TU Clausthal

Clausthal University of Technology

Proceedings of the 11th Workshop on Nonmonotonic Reasoning

Jürgen Dix & Anthony Hunter

IfI Technical Report Series IfI-06-04

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Technical editor of this issue: Tristan Marc Behrens
Contact: jamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Proceedings of the 11th Workshop on Nonmonotonic Reasoning

Jürgen Dix & Anthony Hunter

Clausthal University of Technology and University College London

Abstract

These are the proceedings of the 11th Nonmonotonic Reasoning Workshop. The aim of this series `http://www.kr.org/NMR/` is to bring together active researchers in the broad area of nonmonotonic reasoning, including belief revision, reasoning about actions, planning, logic programming, argumentation, causality, probabilistic and possibilistic approaches to KR, and other related topics.
As part of the program of the 11th workshop, we have assessed the status of the field and discussed issues such as: Significant recent achievements in the theory and automation of NMR; Critical short and long term goals for NMR; Emerging new research directions in NMR; Practical applications of NMR; Significance of NMR to knowledge representation and AI in general.

Preface

This informal proceedings is for the Eleventh International Workshop on Non-Monotonic Reasoning. Its aim is to bring together active researchers in the broad area of nonmonotonic reasoning, including belief revision, reasoning about actions, planning, logic programming, argumentation, causality, probabilistic and possibilistic approaches to KR, and other related topics.

As part of the program we will be considering the status of the field and discussing issues such as: Significant recent achievements in the theory and automation of NMR; Critical short and long term goals for NMR; Emerging new research directions in NMR; Practical applications of NMR; Significance of NMR to knowledge representation and AI in general.

The workshop programme is chaired by Jürgen Dix and Anthony Hunter, and the programme is composed of the following sessions (with session chairs).

1. Answer Set Programming (Ilkka Niemela and Mirek Truszczynski),
2. Theory of NMR and Uncertainty (Salem Benferhat and Gabriele Kern-Isberner),
3. NMR Systems and Applications (Jim Delgrande and Torsten Schaub),
4. Action and Change (Antonis Kakas and Gerhard Lakemeyer),
5. Belief Change and Updates (Andreas Herzig and Maurice Pagnucco),
6. Argumentation, Dialogue, and Decision Making (Leila Amgoud and Guillermo Simari).

Authors have been invited to submit papers directly to any of the above sessions, and all papers have been reviewed by two or three experts in the field. The programme chairs are very grateful to the session chairs for organizing each session, and for arranging the reviewing of the submissions. The programme chairs are also very grateful to the reviewers for their hard work in assessing the submissions and for providing excellent feedback to the authors.

We would also like to thank Mirek Truszczynski for his financial support for the workshop.

Our special thanks go to Tristan Marc Behrens, who put these Proceedings together. This turned out to be an enormous effort and we appreciate his work very much.

May 2006

Jürgen Dix (Germany)
dix@tu-clausthal.de
Anthony Hunter (United Kingdom)
a.hunter@cs.ucl.ac.uk

1 Answer Set Programming

The papers in this collection were presented at the Special Session on Answer Set Programming. This one-day event was a part of the 11th Non-monotonic Reasoning Workshop (NMR 2006) held in collocation with the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006) in the Lake District area of the UK on May 30 - June 1, 2006.

In the 1980s researchers working in the area of nonmonotonic reasoning discovered that their formalisms could be used to describe the behavior of negation as failure in Prolog. This work resulted in logic programming systems of a new kind — *answer-set solvers*, and led to the emergence of a new approach to solving search problems, called *answer-set programming* or ASP, for short. The aim of the session on ASP at NMR 2006 was to facilitate interactions between researchers designing and implementing ASP languages and solvers, and researchers working in the areas of knowledge representation and nonmonotonic reasoning.

The program included 11 papers selected after a review process out of 18 submissions. We thank the program committee members and additional reviewers for careful and unbiased evaluation of the submitted papers. We also want to acknowledge Lengning Liu for his help with the preparation of the papers for the proceedings.

Session chairs

Ilkka Niemelä, Helsinki University of Technology
(Ilkka.Niemela@tkk.fi)

Mirosław Truszczyński, University of Kentucky
(mirek@cs.uky.edu)

Program committee

Marc Denecker, K.U.Leuven, Belgium
(Marc.Denecker@cs.kuleuven.ac.be)

Wolfgang Faber, University of Calabria, Italy
(wf@wfaber.com)

Tomi Janhunen, Helsinki University of Technology, Finland
(ttj@tcs.hut.fi)

Fangzhen Lin, Hong Kong University of Science and Technology, Hong Kong
(flin@cs.ust.hk)

Inna Pivkina, New Mexico State University, USA
(ipivkina@cs.nmsu.edu)

Chiaki Sakama, Wakayama University, Japan
(sakama@sys.wakayama-u.ac.jp)

Hans Tompits, Technische Universität Wien, Austria
(tompits@kr.tuwien.ac.at)

Kewen Wang, Griffith University, Australia
(K.Wang@cit.gu.edu.au)

Additional reviewers

Huan Chen
Yin Chen
Álvaro Cortés Calabuig
Phan Minh Dung
Thomas Eiter
Katsumi Inoue
Kathrin Konczak
Marco Maratea
Maarten Mariën
Emilia Oikarinen
Hou Ping
Francesco Ricca
Roman Schindlauer
Joost Vennekens
Johan Wittocx
Stefan Woltran
Yuting Zhao

Schedule

Wednesday 31 May 2006 (Thirlmere-Wastwater Room)
Session Chairs: I. Niemelä and M. Truszczyński

- 10.30 E. Oikarinen and T. Janhunen, Modular Equivalence for Normal Logic Programs
- 11:00 J. Oetsch, M. Seidl, H. Tompits and S. Woltran, A Tool for Advanced Correspondence Checking in Answer-Set Programming
- 11:30 J. Gressmann, T. Janhunen, R.Mercer, T. Schaub, S. Thiele and R. Ticky, On Probing and Multi-Threading in Platypus
- 12.00 T. Eiter, G. Ianni, R. Schindlauer and H. Tompits, Towards Efficient Evaluation of HEX-Programs
- 12.30 Lunch
- 14.00 M. Gebser and T. Schaub, Tableaux Calculi for Answer Set Programming
- 14.30 C. Anger, M. Gebser and T. Schaub, Approaching the Core of Unfounded Sets
- 15.00 M. Gebser, J. Lee and Y. Lierler, Elementary Sets for Logic Programs
- 15.30 Coffee
- 16.00 T. Syrjänen, Debugging inconsistent answer set programs
- 16.30 T. Eiter and K. Wang Forgetting and Conflict Resolving in Disjunctive Logic Programming
- 17.00 J. Vennekens and M. Denecker Analysing the Structure of Definitions in ID-logic
- 17.30 M. Caminada Well-Founded semantics for Semi-Normal Extended Logic Programs

1.1 Modular Equivalence for Normal Logic Programs

Modular Equivalence for Normal Logic Programs*

Emilia Oikarinen† and **Tomi Janhunen**
Department of Computer Science and Engineering
Helsinki University of Technology (TKK)
P.O. Box 5400, FI-02015 TKK, Finland
Emilia.Oikarinen@tkk.fi and Tomi.Janhunen@tkk.fi

Abstract

A Gaifman-Shapiro-style architecture of program modules is introduced in the case of normal logic programs under stable model semantics. The composition of program modules is suitably limited by module conditions which ensure the compatibility of the module system with stable models. The resulting module theorem properly strengthens Lifschitz and Turner's splitting set theorem (1994) for normal logic programs. Consequently, the respective notion of equivalence between modules, i.e. modular equivalence, proves to be a congruence relation. Moreover, it is analyzed (i) how the translation-based verification technique from (Janhunen & Oikarinen 2005) is accommodated to the case of modular equivalence and (ii) how the verification of weak/visible equivalence can be reorganized as a sequence of module-level tests and optimized on the basis of modular equivalence.

Introduction

Answer set programming (ASP) is a very promising constraint programming paradigm (Niemelä 1999; Marek & Truszczyński 1999; Gelfond & Leone 2002) in which problems are solved by capturing their solutions as *answer sets* or *stable models* of logic programs. The development and optimization of logic programs in ASP gives rise to a meta-level problem of verifying whether subsequent programs are equivalent. To solve this problem, a translation-based approach has been proposed and extended further (Janhunen & Oikarinen 2002; Turner 2003; Oikarinen & Janhunen 2004; Woltran 2004). The underlying idea is to combine two logic programs P and Q under consideration into two logic programs $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ which have no stable models iff P and Q are *weakly equivalent*, i.e. have the same stable models. This enables the use of the same ASP solver, such as SMODELS (Simons, Niemelä, & Soininen 2002), DLV (Leone *et al.* 2006) or GNT (Janhunen *et al.* 2006), for the equivalence verification problem as for the search of stable models in general. First experimental results (Janhunen & Oikarinen 2002; Oikarinen & Janhunen 2004) suggest that the translation-based method can be effective and sometimes much faster than performing a simple cross-check of stable models.

As a potential limitation, the translation-based method as described above treats programs as integral entities and therefore no computational advantage is sought by breaking programs into smaller parts, say *modules* of some kind. Such an optimization strategy is largely preempted by the fact that weak equivalence, denoted by $\equiv$, fails to be a *congruence relation* for $\cup$, i.e. weak equivalence is not preserved under substitutions in unions of programs. More formally put, $P \equiv Q$ does not imply $P \cup R \equiv Q \cup R$ in general. The same can be stated about *uniform equivalence* (Sagiv 1987) but not about *strong equivalence* (Lifschitz, Pearce, & Valverde 2001) which admits substitutions by definition.

From our point of view, strong equivalence seems inappropriate for *fully modularizing* the verification task of weak equivalence. This is simply because two programs P and Q may be weakly equivalent even if they build on respective modules $P_i \subseteq P$ and $Q_i \subseteq Q$ which are not strongly equivalent. For the same reason, program transformations that are known to preserve strong equivalence (Eiter *et al.* 2004) do not provide an inclusive basis for reasoning about weak equivalence. Nevertheless, there are cases where one can utilize the fact that strong equivalence implies weak equivalence. For instance, if P and Q are composed of strongly equivalent pairs of modules P_i and Q_i for all i, then P and Q can be directly inferred to be strongly and weakly equivalent. These observations about strong equivalence motivate the strive for a weaker congruence relation that is compatible with weak equivalence at program-level.

To address the lack of a suitable congruence relation in the context of ASP, we propose a new design in this article. The design superficially resembles that of Gaifman and Shapiro (1989) but stable model semantics (Gelfond & Lifschitz 1988) and special module conditions are incorporated. The feasibility of the design is crystallized in a *module theorem* which shows the module system fully compatible with stable models. In fact, the module theorem established here is a proper strengthening of the splitting set theorem established by Lifschitz and Turner (1994) in the case of normal logic programs. The main difference is that our result allows negative recursion between modules. Moreover, it enables the introduction of a notion of

*The research reported in this paper has been partially funded by the Academy of Finland (project #211025).

†The financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, and Finnish Cultural Foundation is gratefully acknowledged.

equivalence, i.e. *modular equivalence*, which turns out to be a proper congruence relation and reduces to weak equivalence for program modules which have a completely specified input and no hidden atoms. This kind of modules correspond to normal logic programs with completely visible Herbrand base. If normal programs P and Q are composed of modularly equivalent modules P_i and Q_i for all i, then P and Q are modularly equivalent or equivalently stated weakly equivalent. The notion of modular equivalence opens immediately new prospects as regards the translation-based verification method (Janhunen & Oikarinen 2002; Oikarinen & Janhunen 2004). First of all, the method can be tuned for the task of verifying modular equivalence by attaching a *context generator* to program modules in analogy to (Woltran 2004). Second, we demonstrate how the verification of weak equivalence can be reorganized as a sequence of tests, each of which concentrates on a pair of respective modules in the programs subject to the verification task.

The plan for the rest of this article is as follows. As a preparatory step, we briefly review the syntax and semantics of normal logic programs and define notions of equivalence addressed in the sequel. After that we specify program modules as well as establish the module theorem discussed above. Next, we define the notion of modular equivalence, prove the congruence property for it, and give a brief account of computational complexity involved in the respective verification problem. Connections between modular equivalence and the translation-based method for verifying visible equivalence (Janhunen & Oikarinen 2005) are also worked out. Finally, we briefly contrast our work with earlier approaches and present our conclusions.

Normal Logic Programs

We will consider *propositional normal logic programs* in this paper.

Definition 1 *A normal logic program (NLP) is a (finite) set of rules of the form $h \leftarrow B^+, {\sim}B^-$, where h is an atom, B^+ and B^- are sets of atoms, and ${\sim}B = \{{\sim}b \mid b \in B\}$ for any set of atoms B.*

The symbol "$\sim$" denotes *default negation* or *negation as failure* to prove (Clark 1978). Atoms a and their default negations ${\sim}a$ are called *default literals*. A rule consists of two parts: h is the *head* and the rest is the *body*. Let $\mathrm{Head}(P)$ denote the set of head atoms appearing in P, i.e.

$$\mathrm{Head}(P) = \{h \mid h \leftarrow B^+, {\sim}B^- \in P\}.$$

If the body of a rule is empty, the rule is called a *fact* and the symbol "$\leftarrow$" can be omitted. If $B^- = \emptyset$, the rule is *positive*. A program consisting only of positive rules is a *positive logic program*.

Usually the *Herbrand base* $\mathrm{Hb}(P)$ of a normal logic program P is defined to be the set of atoms appearing in the rules of P. We, however, use a revised definition: $\mathrm{Hb}(P)$ *is any fixed set of atoms containing all atoms appearing in the rules of* P. Under this definition the Herbrand base of P can be extended by atoms having no occurrences in P. This aspect is useful e.g. when P is obtained as a result of optimization and there is a need to keep track of the original Herbrand base. Moreover, $\mathrm{Hb}(P)$ is supposed to be finite whenever P is.

Given a normal logic program P, an *interpretation* M of P is a subset of $\mathrm{Hb}(P)$ defining which atoms of $\mathrm{Hb}(P)$ are true ($a \in M$) and which are false ($a \notin M$). An interpretation $M \subseteq \mathrm{Hb}(P)$ is a *(classical) model* of P, denoted by $M \models P$ iff $B^+ \subseteq M$ and $B^- \cap M = \emptyset$ imply $h \in M$ for each rule $h \leftarrow B^+, {\sim}B^- \in P$. For a positive program P, $M \subseteq \mathrm{Hb}(P)$ is the (unique) *least model* of P, denoted by $\mathrm{LM}(P)$, iff there is no $M' \models P$ such that $M' \subset M$. *Stable models* as proposed by Gelfond and Lifschitz (1988) generalize least models for normal logic programs.

Definition 2 *Given a normal logic program P and an interpretation $M \subseteq \mathrm{Hb}(P)$ the Gelfond-Lifschitz reduct*

$$P^M = \{h \leftarrow B^+ \mid h \leftarrow B^+, {\sim}B^- \in P \text{ and } M \cap B^- = \emptyset\},$$

and M is a stable model of P iff $M = \mathrm{LM}(P^M)$.

Stable models are not necessarily unique in general: a normal logic program may have several stable models or no stable models at all. The set of stable models of a NLP P is denoted by $\mathrm{SM}(P)$.

We define a *positive dependency relation* $\leq \subseteq \mathrm{Hb}(P) \times \mathrm{Hb}(P)$ as the reflexive and transitive closure of a relation $\leq_1$ defined as follows. Given $a, b \in \mathrm{Hb}(P)$, we say that b depends directly on a, denoted $a \leq_1 b$, iff there is a rule $b \leftarrow B^+, {\sim}B^- \in P$ such that $a \in B^+$. The *positive dependency graph* of P, $\mathrm{Dep}^+(P)$, is a graph with $\mathrm{Hb}(P)$ as the set of vertices and $\{\langle a, b\rangle \mid a, b \in \mathrm{Hb}(P) \text{ and } a \leq b\}$ as the set of edges. The *negative dependency graph* $\mathrm{Dep}^-(P)$ can be defined analogously. A *strongly connected component* of $\mathrm{Dep}^+(P)$ is a maximal subset $C \subseteq \mathrm{Hb}(P)$ such that for all $a, b \in C$, $\langle a, b\rangle$ is in $\mathrm{Dep}^+(P)$. Thus strongly connected components of $\mathrm{Dep}^+(P)$ partition $\mathrm{Hb}(P)$ into equivalence classes. The dependency relation $\leq$ can then be generalized for the strongly connected components: $C_i \leq C_j$, i.e. C_j depends on C_i, iff $c_i \leq c_j$ for any $c_i \in C_i$ and $c_j \in C_j$.

A *splitting set* for a NLP P is any set $U \subseteq \mathrm{Hb}(P)$ such that for every rule $h \leftarrow B^+, {\sim}B^- \in P$, if $h \in U$ then $B^+ \cup B^- \subseteq U$. The set of rules $h \leftarrow B^+, {\sim}B^- \in P$ such that $\{h\} \cup B^+ \cup B^- \subseteq U$ is the *bottom* of P relative to U, denoted by $\mathrm{b}_U(P)$. The set $\mathrm{t}_U(P) = P \setminus \mathrm{b}_U(P)$ is the *top* of P relative to U. The top can be partially evaluated with respect to an interpretation $X \subseteq U$ resulting a program $\mathrm{e}(\mathrm{t}_U(P), X)$ that contains a rule $h \leftarrow (B^+ \setminus U), {\sim}(B^- \setminus U)$ for each $h \leftarrow B^+, {\sim}B^- \in \mathrm{t}_U(P)$ such that $B^+ \cap U \subseteq X$ and $(B^- \cap U) \cap X = \emptyset$. Given a splitting set U for a NLP P, a *solution* to P with respect to U is a pair $\langle X, Y\rangle$ such that $X \subseteq U$, $Y \subseteq \mathrm{Hb}(P) \setminus U$, $X \in \mathrm{SM}(\mathrm{b}_U(P))$, and $Y \in \mathrm{SM}(\mathrm{e}(\mathrm{t}_U(P), X))$. The *splitting set theorem* relates solutions with stable models.

Theorem 1 *(Lifschitz & Turner 1994) Let U be a splitting set for a NLP P and $M \subseteq \mathrm{Hb}(P)$. Then $M \in \mathrm{SM}(P)$ iff the pair $\langle M \cap U, M \setminus U\rangle$ is a solution to P with respect to U.*

Notions of Equivalence

The notion of *strong equivalence* was introduced by Lifschitz, Pearce and Valverde (2001) whereas *uniform equiva-*

lence has its roots in the database community (Sagiv 1987); cf. (Eiter & Fink 2003) for the case of stable models.

Definition 3 *Normal logic programs P and Q are (weakly) equivalent, denoted $P \equiv Q$, iff $\mathrm{SM}(P) = \mathrm{SM}(Q)$; strongly equivalent, denoted $P \equiv_\mathrm{s} Q$, iff $P \cup R \equiv Q \cup R$ for any normal logic program R; and uniformly equivalent, denoted $P \equiv_\mathrm{u} Q$, iff $P \cup F \equiv Q \cup F$ for any set of facts F.*

Clearly, $P \equiv_\mathrm{s} Q$ implies $P \equiv_\mathrm{u} Q$, and $P \equiv_\mathrm{u} Q$ implies $P \equiv Q$, but not vice versa (in both cases). Strongly equivalent logic programs are semantics preserving substitutes of each other and the relation $\equiv_\mathrm{s}$ can be understood as a *congruence relation* among normal programs, i.e. if $P \equiv_\mathrm{s} Q$, then $P \cup R \equiv_\mathrm{s} Q \cup R$ for all normal programs R. On the other hand, uniform equivalence is not a congruence, as shown in Example 1 below. Consequently, the same applies to weak equivalence and thus $\equiv$ and $\equiv_\mathrm{u}$ are best suited for the comparison of complete programs, and not for modules.

Example 1 *(Eiter* et al. *2004, Example 1) Consider normal logic programs $P = \{a.\}$ and $Q = \{a \leftarrow \sim b.\ a \leftarrow b.\}$. It holds $P \equiv_\mathrm{u} Q$, but $P \cup R \not\equiv Q \cup R$ for $R = \{b \leftarrow a.\}$. Thus $P \not\equiv_\mathrm{s} Q$ and $\equiv_\mathrm{u}$ is not a congruence relation for $\cup$.*

For $P \equiv Q$ to hold, the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ have to be identical subsets of $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$, respectively. The same can be stated about strong and uniform equivalence. This makes these notions of equivalence less useful if $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$ differ by some atoms which are not trivially false in all stable models. Such atoms might, however, be of use when formalizing some auxiliary concepts. Following the ideas from (Janhunen 2003) we partition $\mathrm{Hb}(P)$ into two parts $\mathrm{Hb}_\mathrm{v}(P)$ and $\mathrm{Hb}_\mathrm{h}(P)$ which determine the *visible* and the *hidden* parts of $\mathrm{Hb}(P)$, respectively. In *visible equivalence* the idea is that the hidden atoms in $\mathrm{Hb}_\mathrm{h}(P)$ and $\mathrm{Hb}_\mathrm{h}(Q)$ are local to P and Q and negligible as regards the equivalence of the two programs.

Definition 4 *(Janhunen 2003) Normal logic programs P and Q are visibly equivalent, denoted by $P \equiv_\mathrm{v} Q$, iff $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q)$ and there is a bijection $f : \mathrm{SM}(P) \to \mathrm{SM}(Q)$ such that for all interpretations $M \in \mathrm{SM}(P)$, $M \cap \mathrm{Hb}_\mathrm{v}(P) = f(M) \cap \mathrm{Hb}_\mathrm{v}(Q)$.*

Note that the number of stable models is preserved under $\equiv_\mathrm{v}$. Such a strict correspondence of models is much dictated by the ASP methodology: the stable models of a program usually correspond to the solutions of the problem being solved and thus $\equiv_\mathrm{v}$ preserves the number of solutions, too. In the fully visible case, i.e. $\mathrm{Hb}_\mathrm{h}(P) = \mathrm{Hb}_\mathrm{h}(Q) = \emptyset$, the relation $\equiv_\mathrm{v}$ becomes very close to $\equiv$. The only difference is the additional requirement $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ insisted by $\equiv_\mathrm{v}$. This is of little importance as Herbrand bases can always be extended to meet $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. Since weak equivalence is not a congruence, visible equivalence cannot be a congruence either.

The *relativized variants of strong and uniform equivalence* introduced by Woltran (2004) allow the context to be constrained using a set of atoms A.

Definition 5 *Normal logic programs P and Q are strongly equivalent relative to A, denoted by $P \equiv_\mathrm{s}^A Q$, iff $P \cup R \equiv Q \cup R$ for all normal logic programs R over the set of atoms A; uniformly equivalent relative to A, denoted by $P \equiv_\mathrm{u}^A Q$, iff $P \cup F \equiv Q \cup F$ for all sets of facts $F \subseteq A$.*

Setting $A = \emptyset$ in the above reduces both relativized notions to weak equivalence, and thus neither is a congruence.

Eiter *et al.* (2005) introduce a very general framework based on *equivalence frames* to capture various kinds of equivalence relations. Most of the notions of equivalence defined above can be defined using the framework. Visible equivalence is exceptional in the sense that it does not fit into equivalence frames based on *projected answer sets*. A projective variant of Definition 4 would simply equate $\{M \cap \mathrm{Hb}_\mathrm{v}(P) \mid M \in \mathrm{SM}(P)\}$ to $\{N \cap \mathrm{Hb}_\mathrm{v}(Q) \mid N \in \mathrm{SM}(Q)\}$. As a consequence, the number of answer sets may not be preserved which we find somewhat unsatisfactory because of the general nature of ASP as discussed after Definition 4. Consider, for instance programs $P = \{a \leftarrow \sim b.\ b \leftarrow \sim a.\ \}$ and $Q_n = P \cup \{c_i \leftarrow \sim d_i.\ d_i \leftarrow \sim c_i.\ \mid 0 < i \leq n\}$ with $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q_n) = \{a, b\}$. Whenever $n > 0$ these programs are not visibly equivalent but they would be equivalent under the projective definition. With sufficiently large values of n it is no longer feasible to count the number of different stable models (i.e. solutions) if Q_n is used.

Modular Logic Programs

We define a *logic program module* similarly to Gaifman and Shapiro (1989), but consider the case of normal logic programs instead of positive (disjunctive) logic programs.

Definition 6 *A triple $\mathbb{P} = (P, I, O)$ is a (propositional logic program) module, if*

1. *P is a finite set of rules of the form $h \leftarrow B^+, \sim B^-$;*
2. *I and O are sets of propositional atoms such that $I \cap O = \emptyset$; and*
3. *$\mathrm{Head}(P) \cap I = \emptyset$.*

The Herbrand base of module $\mathbb{P}$, $\mathrm{Hb}(\mathbb{P})$, is the set of atoms appearing in P combined with $I \cup O$. Intuitively the set I defines the *input* of a module and the set O is the *output*. The input and output atoms are considered visible, i.e. the visible Herbrand base of module $\mathbb{P}$ is $\mathrm{Hb}_\mathrm{v}(\mathbb{P}) = I \cup O$. Notice that I and O can also contain atoms not appearing in P, similarly to the possibility of having additional atoms in the Herbrand bases of normal logic programs. All other atoms are hidden, i.e. $\mathrm{Hb}_\mathrm{h}(\mathbb{P}) = \mathrm{Hb}(\mathbb{P}) \setminus \mathrm{Hb}_\mathrm{v}(\mathbb{P})$.

As regards the composition of modules, we follow (Gaifman & Shapiro 1989) and take the union of the disjoint sets of rules involved in them. The conditions given by Gaifman and Shapiro are not yet sufficient for our purposes, and we impose a further restriction denying positive recursion between modules.

Definition 7 *Consider modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ and let $C_1, \ldots, C_n$ be the strongly connected components of $\mathrm{Dep}^+(P_1 \cup P_2)$. There is a positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$, if $C_i \cap O_1 \neq \emptyset$ and $C_i \cap O_2 \neq \emptyset$ for some component C_i.*

The idea is that all inter-module dependencies go through the input/output interface of the modules, i.e. the output of one module can serve as the input for another and hidden atoms are local to each module. Now, if there is a strongly connected component C_i in $\mathrm{Dep}^+(P_1 \cup P_2)$ containing atoms from both O_1 and O_2, we know that, if programs P_1 and P_2 are combined, some output atom a of $\mathbb{P}_1$ depends positively on some output atom b of $\mathbb{P}_2$ which again depends positively on a. This yields a positive recursion.

Definition 8 *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be modules such that*

1. $O_1 \cap O_2 = \emptyset$;
2. $\mathrm{Hb_h}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) = \mathrm{Hb_h}(\mathbb{P}_2) \cap \mathrm{Hb}(\mathbb{P}_1) = \emptyset$; *and*
3. *there is no positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$.*

Then the join of $\mathbb{P}_1$ and $\mathbb{P}_2$, denoted by $\mathbb{P}_1 \sqcup \mathbb{P}_2$, is defined, and $\mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2)$.

Remark. Condition 1 in Definition 8 is actually redundant as it is implied by condition 3. Also, condition 2 can be circumvented in practice using a suitable scheme, e.g. based on module names, to rename the hidden atoms uniquely for each module.

Some observations follow. Since each atom is defined in one module, the sets of rules in $\mathbb{P}_1$ and $\mathbb{P}_2$ are disjoint, i.e. $P_1 \cap P_2 = \emptyset$. Also,

$$\begin{aligned} \mathrm{Hb}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb}(\mathbb{P}_1) \cup \mathrm{Hb}(\mathbb{P}_2), \\ \mathrm{Hb_v}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb_v}(\mathbb{P}_1) \cup \mathrm{Hb_v}(\mathbb{P}_2), \text{ and} \\ \mathrm{Hb_h}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb_h}(\mathbb{P}_1) \cup \mathrm{Hb_h}(\mathbb{P}_2). \end{aligned}$$

Note that the module conditions above impose no restrictions on *negative* dependencies or on positive dependencies *inside* modules. The input of $\mathbb{P}_1 \sqcup \mathbb{P}_2$ might be smaller than the union of inputs of individual modules. This is illustrated by the following example.

Example 2 *Consider modules $\mathbb{P} = (\{a \leftarrow \sim b.\}, \{b\}, \{a\})$ and $\mathbb{Q} = (\{b \leftarrow \sim a.\}, \{a\}, \{b\})$. The join of $\mathbb{P}$ and $\mathbb{Q}$ is defined, and $\mathbb{P} \sqcup \mathbb{Q} = (\{a \leftarrow \sim b.\ b \leftarrow \sim a.\}, \emptyset, \{a, b\})$.*

The following hold for the intersections of Herbrand bases under the conditions 1 and 2 in Definition 8:

$$\begin{aligned} &\mathrm{Hb_v}(\mathbb{P}_1) \cap \mathrm{Hb_v}(\mathbb{P}_2) \\ &\qquad = \mathrm{Hb}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) \\ &\qquad = (I_1 \cap I_2) \cup (I_1 \cap O_2) \cup (I_2 \cap O_1), \text{ and} \\ &\mathrm{Hb_h}(\mathbb{P}_1) \cap \mathrm{Hb_h}(\mathbb{P}_2) = \emptyset. \end{aligned}$$

Join operation $\sqcup$ has the following properties:

- Identity: $\mathbb{P} \sqcup (\emptyset, \emptyset, \emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup \mathbb{P} = \mathbb{P}$ for all $\mathbb{P}$.
- Commutativity: $\mathbb{P}_1 \sqcup \mathbb{P}_2 = \mathbb{P}_2 \sqcup \mathbb{P}_1$ for all modules $\mathbb{P}_1$ and $\mathbb{P}_2$ such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined.
- Associativity: $(\mathbb{P}_1 \sqcup \mathbb{P}_2) \sqcup \mathbb{P}_3 = \mathbb{P}_1 \sqcup (\mathbb{P}_2 \sqcup \mathbb{P}_3)$ for all modules $\mathbb{P}_1, \mathbb{P}_2$ and $\mathbb{P}_3$ such that the joins are defined.

Note that equality sign "=" used here denotes syntactical equivalence, whereas semantical equivalence will be defined in the next section.

The stable semantics of a module is defined with respect to a given input, i.e. a subset of the input atoms of the module. Input is seen as a set of facts (or a database) to be added to the module.

Definition 9 *Given a module $\mathbb{P} = (P, I, O)$ and a set of atoms $A \subseteq I$ the instantiation of $\mathbb{P}$ with the input A is $\mathbb{P}(A) = \mathbb{P} \sqcup \mathbb{F}_A$, where $\mathbb{F}_A = (\{a. \mid a \in A\}, \emptyset, I)$.*

Note that $\mathbb{P}(A) = (P \cup \{a. \mid a \in A\}, \emptyset, I \cup O)$ is essentially a normal logic program with $I \cup O$ as the visible Herbrand base. We can thus generalize the stable model semantics for modules. In the sequel we identify $\mathbb{P}(A)$ with the respective set of rules $P \cup F_A$, where $F_A = \{a. \mid a \in A\}$. In the following $M \cap I$ acts as a particular input with respect to which the module is instantiated.

Definition 10 *An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a stable model of a module $\mathbb{P} = (P, I, O)$, denoted by $M \in \mathrm{SM}(\mathbb{P})$, iff $M = \mathrm{LM}(P^M \cup F_{M \cap I})$.*

We define a concept of *compatibility* to describe when a stable model M_1 of module $\mathbb{P}_1$ can be combined with a stable model M_2 of another module $\mathbb{P}_2$. This is exactly when M_1 and M_2 share the common (visible) part.

Definition 11 *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules, and $M_1 \in \mathrm{SM}(\mathbb{P}_1)$ and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ their stable models which are compatible, iff $M_1 \cap \mathrm{Hb_v}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb_v}(\mathbb{P}_1)$.*

If a program (module) consists of several modules, its stable models are locally stable for the respective submodules; and on the other hand, local stability implies global stability as long as the stable models of the submodules are compatible.

Theorem 2 *(Module theorem). Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. Now, $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ iff $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2)$, and M_1 and M_2 are compatible.*

Proof sketch. "$\Rightarrow$" M_1 and M_2 are clearly compatible and it is straightforward to show that conditions 1 and 2 in Definition 8 imply $M_1 \in \mathrm{SM}(\mathbb{P}_1)$ and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$.
"$\Leftarrow$" Consider $\mathbb{P}_1 = (P_1, I_1, O_1)$, $\mathbb{P}_2 = (P_2, I_2, O_2)$ and their join $\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 = (P, I, O)$. Let $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ be compatible and define $M = M_1 \cup M_2$. There is a strict total ordering $<$ for the strongly connected components C_i of $\mathrm{Dep}^+(P)$ such that if $C_i < C_j$, then $C_i \leq C_j$ and $C_j \not\leq C_i$; or $C_i \not\leq C_j$ and $C_j \not\leq C_i$. Let $C_1 < \cdots < C_n$ be such an ordering. Show that exactly one of the following holds for each C_i: (i) $C_i \subseteq I$, (ii) $C_i \subseteq O_1 \cup \mathrm{Hb_h}(\mathbb{P}_1)$, or (iii) $C_i \subseteq O_2 \cup \mathrm{Hb_h}(\mathbb{P}_2)$. Finally, show by induction that

$$M \cap (\bigcup_{i=1}^{k} C_i) = \mathrm{LM}(P^M \cup F_{M \cap I}) \cap (\bigcup_{i=1}^{k} C_i)$$

holds for $0 \leq k \leq n$ by applying the splitting set theorem (Lifschitz & Turner 1994). □

Example 3 shows that condition 3 in Definition 8 is necessary to guarantee that local stability implies global stability.

Example 3 *Consider $\mathbb{P}_1 = (\{a \leftarrow b.\}, \{b\}, \{a\})$ and $\mathbb{P}_2 = (\{b \leftarrow a.\}, \{a\}, \{b\})$ with $\mathrm{SM}(\mathbb{P}_1) = \mathrm{SM}(\mathbb{P}_2) = \{\emptyset, \{a, b\}\}$. The join of $\mathbb{P}_1$ and $\mathbb{P}_2$ is not defined because of positive recursion (conditions 1 and 2 in Definition 8 are satisfied, however). For a NLP $P = \{a \leftarrow b.\ b \leftarrow a.\}$, we get $\mathrm{SM}(P) = \{\emptyset\}$. Thus, the positive dependency between a and b excludes $\{a, b\}$ from $\mathrm{SM}(P)$.*

Theorem 2 is strictly stronger than the splitting set theorem (Lifschitz & Turner 1994) for normal logic programs. If U is a splitting set for a NLP P, then

$$P = \mathbb{B} \sqcup \mathbb{T} = (\mathrm{b}_U(P), \emptyset, U) \sqcup (\mathrm{t}_U(P), U, \mathrm{Hb}(P) \setminus U),$$

and it follows from Theorems 1 and 2 that $M_1 \in \mathrm{SM}(\mathbb{B})$ and $M_2 \in \mathrm{SM}(\mathbb{T})$ iff $\langle M_1, M_2 \setminus U \rangle$ is a solution for P with respect to U. On the other hand the splitting set theorem cannot be applied to e.g. $\mathbb{P} \sqcup \mathbb{Q}$ from Example 2, since neither $\{a\}$ nor $\{b\}$ is a splitting set. Our theorem also strengthens a module theorem given in (Janhunen 2003, Theorem 6.22) to cover normal programs that involve positive body literals, too. Moreover, Theorem 2 can easily be generalized for modules consisting of several submodules. Consider a collection of modules $\mathbb{P}_1, \ldots, \mathbb{P}_n$ such that the join $\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ is defined (recall that $\sqcup$ is associative). We say that a collection of stable models $\{M_1, \ldots, M_n\}$ for modules $\mathbb{P}_1, \ldots, \mathbb{P}_n$, respectively, is *compatible*, iff M_i and M_j are pairwise compatible for all $1 \leq i, j \leq n$.

Corollary 1 *Let $\mathbb{P}_1, \ldots, \mathbb{P}_n$ be a collection of modules such that $\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ is defined. Now $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n)$ iff $M_i = M \cap \mathrm{Hb}(\mathbb{P}_i) \in \mathrm{SM}(\mathbb{P}_i)$ for all $1 \leq i \leq n$, and the set of stable models $\{M_1, \ldots, M_n\}$ is compatible.*

Corollary 1 enables the computation of stable models on a module-by-module basis, but it leaves us the task of excluding mutually incompatible combinations of stable models.

Example 4 *Consider modules*

$$\begin{aligned} \mathbb{P}_1 &= (\{a \leftarrow {\sim}b.\}, \{b\}, \{a\}), \\ \mathbb{P}_2 &= (\{b \leftarrow {\sim}c.\}, \{c\}, \{b\}), \text{ and} \\ \mathbb{P}_3 &= (\{c \leftarrow {\sim}a.\}, \{a\}, \{c\}). \end{aligned}$$

The join $\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 \sqcup \mathbb{P}_3$ is defined,

$$\mathbb{P} = (\{a \leftarrow {\sim}b.\ b \leftarrow {\sim}c.\ c \leftarrow {\sim}a.\}, \emptyset, \{a, b, c\}).$$

Now $\mathrm{SM}(\mathbb{P}_1) = \{\{a\}, \{b\}\}$, $\mathrm{SM}(\mathbb{P}_2) = \{\{b\}, \{c\}\}$ and $\mathrm{SM}(\mathbb{P}_3) = \{\{a\}, \{c\}\}$. To apply Corollary 1 for finding $\mathrm{SM}(\mathbb{P})$, one has to find a compatible triple of stable models M_1, M_2, and M_3 for $\mathbb{P}_1$, $\mathbb{P}_2$, and $\mathbb{P}_3$, respectively.

- *Now $\{a\} \in \mathrm{SM}(\mathbb{P}_1)$ and $\{c\} \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, since $\{a\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2) = \emptyset = \{c\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1)$. However, $\{a\} \in \mathrm{SM}(\mathbb{P}_3)$ is not compatible with $\{c\} \in \mathrm{SM}(\mathbb{P}_2)$, since $\{c\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_3) = \{c\} \neq \emptyset = \{a\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2)$. On the other hand, $\{c\} \in \mathrm{SM}(\mathbb{P}_3)$ is not compatible with $\{a\} \in \mathrm{SM}(\mathbb{P}_1)$, since $\{a\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_3) = \{a\} \neq \emptyset = \{c\} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1)$.*
- *Also $\{b\} \in \mathrm{SM}(\mathbb{P}_1)$ and $\{b\} \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, but $\{b\} \in \mathrm{SM}(\mathbb{P}_1)$ is incompatible with $\{a\} \in \mathrm{SM}(\mathbb{P}_3)$. Nor is $\{b\} \in \mathrm{SM}(\mathbb{P}_2)$ compatible with $\{c\} \in \mathrm{SM}(\mathbb{P}_3)$.*

Thus it is impossible to select $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ and $M_3 \in \mathrm{SM}(\mathbb{P}_3)$ such that $\{M_1, M_2, M_3\}$ is compatible, which is understandable as $\mathrm{SM}(\mathbb{P}) = \emptyset$.

Modular Equivalence

The definition of *modular equivalence* combines features from relativized uniform equivalence (Woltran 2004) and visible equivalence (Janhunen 2003).

Definition 12 *Logic program modules $\mathbb{P} = (P, I_P, O_P)$ and $\mathbb{Q} = (Q, I_Q, O_Q)$ are modularly equivalent, denoted by $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$, iff*

1. *$I_P = I_Q = I$ and $O_P = O_Q = O$, and*
2. *$\mathbb{P}(A) \equiv_\mathrm{v} \mathbb{Q}(A)$ for all $A \subseteq I$.*

Modular equivalence is very close to visible equivalence defined for modules. As a matter a fact, if Definition 4 is generalized for program modules, the second condition in Definition 12 can be revised to $\mathbb{P} \equiv_\mathrm{v} \mathbb{Q}$. However, $\mathbb{P} \equiv_\mathrm{v} \mathbb{Q}$ is not enough to cover the first condition in Definition 12, as visible equivalence only enforces $\mathrm{Hb}_\mathrm{v}(\mathbb{P}) = \mathrm{Hb}_\mathrm{v}(\mathbb{Q})$. If $I = \emptyset$, modular equivalence coincides with visible equivalence. If $O = \emptyset$, then $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$ means that $\mathbb{P}$ and $\mathbb{Q}$ have the same number of stable models on each input.

Furthermore, if one considers the *fully visible case*, i.e. $\mathrm{Hb}_\mathrm{h}(\mathbb{P}) = \mathrm{Hb}_\mathrm{h}(\mathbb{Q}) = \emptyset$, modular equivalence can be seen as a special case of A-uniform equivalence for $A = I$. Recall, however, the restrictions $\mathrm{Head}(P) \cap I = \mathrm{Head}(Q) \cap I = \emptyset$ imposed by module structure. With a further restriction $I = \emptyset$, modular equivalence coincides with weak equivalence because $\mathrm{Hb}(\mathbb{P}) = \mathrm{Hb}(\mathbb{Q})$ can always be satisfied by extending Herbrand bases. Basically, setting $I = \mathrm{Hb}(\mathbb{P})$ would give us uniform equivalence, but the additional condition $\mathrm{Head}(P) \cap I = \emptyset$ leaves room for the empty module only.

Since $\equiv_\mathrm{v}$ is not a congruence relation for $\cup$, neither is modular equivalence. The situation changes, however, if one considers the join operation $\sqcup$ which suitably restricts possible contexts. Consider for instance the programs P and Q given in Example 1. We can define modules based on them: $\mathbb{P} = (P, \{b\}, \{a\})$ and $\mathbb{Q} = (Q, \{b\}, \{a\})$. Now $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$ and it is not possible to define a module $\mathbb{R}$ based on $R = \{b \leftarrow a.\}$ such that $\mathbb{Q} \sqcup \mathbb{R}$ is defined.

Theorem 3 *Let $\mathbb{P}, \mathbb{Q}$ and $\mathbb{R}$ be logic program modules. If $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$ and both $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined, then $\mathbb{P} \sqcup \mathbb{R} \equiv_\mathrm{m} \mathbb{Q} \sqcup \mathbb{R}$.*

Proof. Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be modules such that $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$. Let $\mathbb{R} = (R, I_R, O_R)$ be an arbitrary module such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. Consider an arbitrary $M \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$. By Theorem 2, $M_P = M \cap \mathrm{Hb}(\mathbb{P}) \in \mathrm{SM}(\mathbb{P})$ and $M_R = M \cap \mathrm{Hb}(\mathbb{R}) \in \mathrm{SM}(\mathbb{R})$. Since $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$, there is a bijection $f : \mathrm{SM}(\mathbb{P}) \to \mathrm{SM}(\mathbb{Q})$ such that $M_P \in \mathrm{SM}(\mathbb{P}) \iff f(M_P) \in \mathrm{SM}(\mathbb{Q})$, and

$$M_P \cap (O \cup I) = f(M_P) \cap (O \cup I). \tag{1}$$

Let $M_Q = f(M_P)$. Clearly, M_P and M_R are compatible. Since (1) holds, also M_Q and M_R are compatible. Applying Theorem 2 we get $M_Q \cup M_R \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$. Define function $g : \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R}) \to \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$ as

$$g(M) = f(M \cap \mathrm{Hb}(\mathbb{P})) \cup (M \cap \mathrm{Hb}(\mathbb{R})).$$

Clearly, g restricted to the visible part is an identity function, i.e. $M \cap (I \cup I_R \cup O \cup O_R) = g(M) \cap (I \cup I_R \cup O \cup O_R)$. Function g is a bijection, since

- g is an injection: $M \neq N$ implies $g(M) \neq g(N)$ for all $M, N \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$, since $f(M \cap \mathrm{Hb}(\mathbb{P})) \neq f(N \cap \mathrm{Hb}(\mathbb{P}))$ or $M \cap \mathrm{Hb}(\mathbb{R}) \neq N \cap \mathrm{Hb}(\mathbb{R})$.

- g is a surjection: for any $M \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$, $N = f^{-1}(M \cap \mathrm{Hb}(\mathbb{Q})) \cup (M \cap \mathrm{Hb}(\mathbb{R})) \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$ and $g(N) = M$, since f is a surjection.

The inverse function $g^{-1} : \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R}) \to \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$ can be defined as $g^{-1}(N) = f^{-1}(N \cap \mathrm{Hb}(\mathbb{Q})) \cup (N \cap \mathrm{Hb}(\mathbb{R}))$. Thus $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$. □

It is instructive to consider a potentially stronger variant of modular equivalence defined in analogy to strong equivalence (Lifschitz *et al.* 2001): $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$ holds for all $\mathbb{R}$ such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. However, Theorem 3 implies that $\equiv^{\mathrm{s}}_{\mathrm{m}}$ adds nothing to $\equiv_{\mathrm{m}}$ since $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

Complexity Remarks

Let us then make some observations about the computational complexity of verifying modular equivalence of normal logic programs. In general, deciding $\equiv_{\mathrm{m}}$ is **coNP**-hard, since deciding the weak equivalence $P \equiv Q$ reduces to deciding $(P, \emptyset, \mathrm{Hb}(P)) \equiv_{\mathrm{m}} (Q, \emptyset, \mathrm{Hb}(Q))$. In the fully visible case $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}_{\mathrm{h}}(Q) = \emptyset$, deciding $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ can be reduced to deciding relativized uniform equivalence $P \equiv^{I}_{\mathrm{u}} Q$ (Woltran 2004) and thus deciding $\equiv_{\mathrm{m}}$ is **coNP**-complete in this restricted case. In the other extreme, $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = \emptyset$ and $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P}$ and $\mathbb{Q}$ have the same number of stable models. This suggests a much higher computational complexity of verifying $\equiv_{\mathrm{m}}$ in general because classical models can be captured with stable models (Niemelä 1999) and counting stable models cannot be easier than #SAT which is #**P**-complete (Valiant 1979).

A way to govern the computational complexity of verifying $\equiv_{\mathrm{m}}$ is to limit the use of hidden atoms as done in the case of $\equiv_{\mathrm{v}}$ by Janhunen and Oikarinen (2005). Therefrom we adopt the property of having *enough visible atoms* (the EVA property for short) defined as follows. For a normal program P and an interpretation $M_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$ for the visible part of P, the *hidden part* $P_{\mathrm{h}}/M_{\mathrm{v}}$ of P relative M_{v} contains for each rule $h \leftarrow B^+, {\sim}B^- \in P$ such that $h \in \mathrm{Hb}_{\mathrm{h}}(P)$ and $M_{\mathrm{v}} \models B^+_{\mathrm{v}} \cup {\sim}B^-_{\mathrm{v}}$, the respective hidden part $h \leftarrow B^+_{\mathrm{h}}, {\sim}B^-_{\mathrm{h}}$. The construction of the hidden part $P_{\mathrm{h}}/M_{\mathrm{v}}$ is closely related to the simplification operation $\mathrm{simp}(P, T, F)$ proposed by Cholewinski and Truszczyński (1999), but restricted in the sense that T and F are subsets of $\mathrm{Hb}_{\mathrm{v}}(P)$ rather than $\mathrm{Hb}(P)$. More precisely put, we have $P_{\mathrm{h}}/M_{\mathrm{v}} = \mathrm{simp}(P, M_{\mathrm{v}}, \mathrm{Hb}_{\mathrm{v}}(P) - M_{\mathrm{v}})$ for any program P.

Definition 13 *A normal logic program P has enough visible atoms iff $P_{\mathrm{h}}/M_{\mathrm{v}}$ has a unique stable model for every interpretation $M_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$.*

The intuition behind Definition 13 is that the interpretation of $\mathrm{Hb}_{\mathrm{h}}(P)$ is uniquely determined for each interpretation of $\mathrm{Hb}_{\mathrm{v}}(P)$ if P has the EVA property. Consequently, the stable models of P can be distinguished on the basis of their visible parts. By the EVA assumption (Janhunen & Oikarinen 2005), the verification of $\equiv_{\mathrm{v}}$ becomes a **coNP**-complete problem for SMODELS programs[1] involving hidden atoms. This complexity result enables us to generalize the translation-based method from (Janhunen & Oikarinen 2002) for deciding $\equiv_{\mathrm{v}}$. Although verifying the EVA property can be hard in general, there are syntactic subclasses of normal programs (e.g. those for which $P_{\mathrm{h}}/M_{\mathrm{v}}$ is always stratified) with the EVA property. It should be stressed that the use of visible atoms remains unlimited and thus the full expressiveness of normal rules remains at our disposal.

So far we have discussed the role of the EVA assumption in the verification of $\equiv_{\mathrm{v}}$. It is equally important in conjunction with $\equiv_{\mathrm{m}}$. This becomes evident once we work out the interconnections of the two relations in the next section.

Application Strategies

The objective of this section is to describe ways in which modular equivalence can be exploited in the verification of visible/weak equivalence. One concrete step in this respect is to reduce the problem of verifying $\equiv_{\mathrm{m}}$ to that of $\equiv_{\mathrm{v}}$ by introducing a special module $\mathbb{G}_I$ that acts as a context generator. A similar technique is used by Woltran (2004) in the case of relativized uniform equivalence.

Theorem 4 *Let $\mathbb{P}$ and $\mathbb{Q}$ be program modules such that $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = O \cup I$. Then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ where $\mathbb{G}_I = (\{a \leftarrow {\sim}\overline{a}.\ \overline{a} \leftarrow {\sim}a \mid a \in I\}, \emptyset, I)$ is a module generating all possible inputs for $\mathbb{P}$ and $\mathbb{Q}$.*

Proof sketch. Note that $\mathbb{G}_I$ has $2^{|I|}$ stable models of the form $A \cup \{\overline{a} \mid a \in I \setminus A\}$ for each $A \subseteq I$. Thus $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{P} \sqcup \mathbb{G}_I$ and $\mathbb{Q} \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ follow by Definitions 2 and 4 and Theorem 2. It follows that $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ iff $\mathbb{P}(A) \equiv_{\mathrm{v}} \mathbb{Q}(A)$ for all $A \subseteq I$ iff $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$. □

As a consequence of Theorem 4, the translation-based technique from (Janhunen & Oikarinen 2005, Theorem 5.4) can be used to verify $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ given that $\mathbb{P}$ and $\mathbb{Q}$ have enough visible atoms ($\mathbb{G}_I$ has the EVA property trivially). More specifically, the task is to show that $\mathrm{EQT}(\mathbb{P} \sqcup \mathbb{G}_I, \mathbb{Q} \sqcup \mathbb{G}_I)$ and $\mathrm{EQT}(\mathbb{Q} \sqcup \mathbb{G}_I, \mathbb{P} \sqcup \mathbb{G}_I)$ have no stable models.

The introduction of modular equivalence was much motivated by the need of modularizing the verification of weak equivalence[2]. We believe that such a modularization could be very effective in a setting where Q is an optimized version of P. Typically Q is obtained by making some local modifications to P. In the following, we propose a further strategy to utilize modular equivalence in the task of verifying the visible/weak equivalence of P and Q.

An essential prerequisite is to identify a module structure for P and Q. Basically, there are two ways to achieve this: either the programmer specifies modules explicitly or strongly connected components of $\mathrm{Dep}^+(P)$ and $\mathrm{Dep}^+(Q)$ are computed to detect them automatically. Assuming the relationship of P and Q as described above, it is likely that these components are pairwise compatible and we can partition P and Q so that $P = \mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ and $Q = \mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_n$ where the respective modules $\mathbb{P}_i$ and $\mathbb{Q}_i$ have the same input and output. Note that $\mathbb{P}_i$ and $\mathbb{Q}_i$ can be the same for a number of i's under the locality assumption.

In this setting, the verification of $\mathbb{P}_i \equiv_{\mathrm{m}} \mathbb{Q}_i$ for each pair of modules $\mathbb{P}_i$ and $\mathbb{Q}_i$ is not of interest as $\mathbb{P}_i \not\equiv_{\mathrm{m}} \mathbb{Q}_i$ does

[1] This class of programs includes normal logic programs.

[2] Recall that $\equiv_{\mathrm{v}}$ coincides with $\equiv$ for programs P and Q having equal and fully visible Herbrand bases.

not necessarily imply $P \not\equiv_{\mathrm{v}} Q$. However, the verification of $P \equiv_{\mathrm{v}} Q$ can still be organized as a sequence of n tests at the level of modules, i.e. it is sufficient to show

$$\mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_{i-1} \sqcup \mathbb{P}_i \sqcup \cdots \sqcup \mathbb{P}_n \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_i \sqcup \mathbb{P}_{i+1} \sqcup \cdots \sqcup \mathbb{P}_n \quad (2)$$

for each $1 \leq i \leq n$ and the resulting chain of equalities conveys $P \equiv_{\mathrm{v}} Q$ under the assumption that P and Q have a completely specified input. If not, then $\equiv_{\mathrm{m}}$ can be addressed using a similar chaining technique based on (2).

Example 5 *Consider normal logic programs P and Q both consisting of two submodules, i.e. $P = \mathbb{P}_1 \sqcup \mathbb{P}_2$ and $Q = \mathbb{Q}_1 \sqcup \mathbb{Q}_2$ where $\mathbb{P}_1, \mathbb{P}_2, \mathbb{Q}_1$, and $\mathbb{Q}_2$ are defined by*

$$\begin{aligned} \mathbb{P}_1 &= (\{c \leftarrow {\sim}a.\}, \{a, b\}, \{c\}), \\ \mathbb{P}_2 &= (\{a \leftarrow b.\}, \emptyset, \{a, b\}), \\ \mathbb{Q}_1 &= (\{c \leftarrow {\sim}b.\}, \{a, b\}, \{c\}), \text{ and} \\ \mathbb{Q}_2 &= (\{b \leftarrow a.\}, \emptyset, \{a, b\}). \end{aligned}$$

Now, $\mathbb{P}_1 \not\equiv_{\mathrm{m}} \mathbb{Q}_1$, but $\mathbb{P}_1$ and $\mathbb{Q}_1$ are visibly equivalent in all contexts produced by both $\mathbb{P}_2$ and $\mathbb{Q}_2$ (in this case actually $\mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_2$ holds, but that is not necessary). Thus

$$\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{Q}_2,$$

which verifies $P \equiv_{\mathrm{v}} Q$ as well as $P \equiv Q$.

It should be stressed that the programs involved in each test (2) differ in $\mathbb{P}_i$ and $\mathbb{Q}_i$ for which the other modules form a common context, say $\mathbb{C}_i$. A way to optimize the verification of $\mathbb{P}_i \sqcup \mathbb{C}_i \equiv_{\mathrm{m}} \mathbb{Q}_i \sqcup \mathbb{C}_i$ is to view $\mathbb{C}_i$ as a module generating input for $\mathbb{P}_i$ and $\mathbb{Q}_i$ and to adjust the translation-based method from (Janhunen & Oikarinen 2005) for such generators. More specifically, we seek computational advantage from using $\mathrm{EQT}(\mathbb{P}_i, \mathbb{Q}_i) \sqcup \mathbb{C}_i$ rather than $\mathrm{EQT}(\mathbb{P}_i \sqcup \mathbb{C}_i, \mathbb{Q}_i \sqcup \mathbb{C}_i)$ and especially when the context $\mathbb{C}_i$ is clearly larger than the modules $\mathbb{P}_i$ and $\mathbb{Q}_i$. By symmetry, the same strategy is applicable to $\mathbb{Q}_i$ and $\mathbb{P}_i$.

Related Work

The notion of modular equivalence is already contrasted with other equivalence relations in previous sections.

Bugliesi, Lamma and Mello (1994) present an extensive survey of modularity in conventional logic programming. Two mainstream programming disciplines can be identified: *programming-in-the-large* where programs are composed with algebraic operators (O'Keefe 1985) and *programming-in-the-small* with abstraction mechanisms (Miller 1986). Our approach can be classified in the former discipline due to resemblance to that of Gaifman and Shapiro (1989). But stable model semantics and the denial of positive recursion between modules can be pointed out as obvious differences in view of their approach.

A variety of conditions on modules have also been introduced. For instance, in contrast to our work, Maher (1993) forbids all recursion between modules and considers Przymusinski's *perfect models* rather than stable models. Brogi et al. (1994) employ operators for program composition and visibility conditions that correspond to the second item in Definition 8. However, their approach covers only positive programs and the least model semantics. Etalle and Gabbrielli (1996) restrict the composition of *constraint logic program* (CLP) modules with a condition that is close to ours: $\mathrm{Hb}(P) \cap \mathrm{Hb}(Q) \subseteq \mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$ but no distinction between input and output is made; e.g. $O_P \cap O_Q \neq \emptyset$ is allowed according to their definitions. They also strive for congruence relations but in the case of CLPs.

Eiter, Gottlob, and Mannila (1997) consider the class of *disjunctive Datalog* used as query programs π over relational databases. As regards syntax, such programs are disjunctive programs which cover normal programs (involving variables though) as a special case. The rough idea is that π is instantiated with respect to an input database D for the given input schema $\mathbf{R}$. The resulting models of $\pi[D]$, which depend on the semantics chosen for π, are projected with respect to an output schema $\mathbf{S}$. To link this approach to ours, it is possible to view π as a program module $\mathbb{P}$ with input I and output O based on $\mathbf{R}$ and $\mathbf{S}$, respectively. Then $\pi[D]$ is obtained as $\mathbb{P}(D)$. In contrast to our work, their module architecture is based on both *positive and negative dependencies* and no recursion between modules is tolerated. These constraints enable a straightforward generalization of the splitting set theorem for the architecture.

Faber *et al.* (2005) apply the *magic set method* in the evaluation of Datalog programs with negation, i.e. effectively normal programs. This involves the concept of an *independent set* S of a program P which is a specialization of a splitting set (recall Theorem 1). Roughly speaking, the idea is that the semantics of an independent set S is not affected by the rest of P and thus S gives rise to a *module* $T = \{h \leftarrow B^+, {\sim}B^- \in P \mid h \in S\}$ of P so that $T \subseteq P$ and $\mathrm{Head}(T) = S$. Due to close relationship to splitting sets, independent sets are not that flexible as regards parceling normal programs. For instance, the splittings demonstrated in Examples 2 and 4 are impossible with independent sets. In certain cases, the distinction of *dangerous rules* in the definition of independent sets pushes negative recursion inside modules which is unnecessary in view of our results. Finally, the module theorem of Faber *et al.* (2005) is weaker than Theorem 2.

Eiter, Gottlob and Veith (1997) address modularity within ASP and view program modules as *generalized quantifiers* the definitions of which are allowed to nest, i.e. P can refer to another module Q by using it as a generalized quantifier. This is an abstraction mechanism typical to programming-in-the-small approaches.

Conclusion

In this article, we a propose a module architecture for logic programs in answer set programming. The compatibility of the module system and stable models is achieved by allowing positive recursion to occur inside modules only. The current design gives rise to a number of interesting results. First, the splitting set theorem by Lifschitz and Turner (1994) is generalized to the case where negative recursion is allowed between modules. Second, the resulting notion of *modular equivalence* is a proper congruence relation for the join operation between modules. Third, the verification

of modular equivalence can be accomplished with existing methods so that specialized solvers need not be developed. Last but not least, we have a preliminary understanding how the task of verifying weak equivalence can be modularized using modular equivalence.

Yet the potential gain from the modular verification strategy has to be evaluated by conducting experiments. A further theoretical question is how the existing model theory based on *SE-models* and *UE-models* (Eiter & Fink 2003) is tailored to the case of modular equivalence. There is also a need to expand the module architecture and module theorem proposed here to cover other classes of logic programs such as e.g. weight constraint programs, disjunctive programs, and nested programs.

References

Brogi, A.; Mancarella, P.; Pedreschi, D.; and Turini, F. 1994. Modular logic programming. *ACM Transactions on Programming Languages and Systems* 16(4):1361–1398.

Bugliesi, M.; Lamma, E.; and Mello, P. 1994. Modularity in logic programming. *Journal of Logic Programming* 19/20:443–502.

Cholewinski, P., and Truszczyński, M. 1999. Extremal problems in logic programming and stable model computation. *Journal of Logic Programming* 38(2):219–242.

Clark, K. L. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum Press. 293–322.

Eiter, T., and Fink, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Proc. of the 19th International Conference on Logic Programming*, volume 2916 of *LNCS*, 224–238. Mumbay, India: Springer.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2004. Simplifying logic programs under uniform and strong equivalence. In *Proc. of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *LNAI*, 87–99. Fort Lauderdale, USA: Springer.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems* 22(3):364–418.

Eiter, T.; Gottlob, G.; and Veith, H. 1997. Modular logic programming and generalized quantifiers. In *Proc. of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *LNCS*, 290–309. Dagstuhl, Germany: Springer.

Eiter, T.; Tompits, H.; and Woltran, S. 2005. On solution correspondences in answer-set programming. In *Proc. of 19th International Joint Conference on Artificial Intelligence*, 97–102. Edinburgh, UK: Professional Book Center.

Etalle, S., and Gabbrielli, M. 1996. Transformations of CLP modules. *Theoretical Computer Science* 166(1–2):101–146.

Faber, W.; Greco, G.; and Leone, N. 2005. Magic sets and their application to data integration. In *Proc. of 10th International Conference on Database Theory, ICDT'05*, volume 3363 of *LNCS*, 306–320. Edinburgh, UK: Springer.

Gaifman, H., and Shapiro, E. 1989. Fully abstract compositional semantics for logic programs. In *Proc. of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 134–142. Austin, Texas, USA: ACM Press.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation — the A-Prolog perspective. *Artificial Intelligence* 138:3–38.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th International Conference on Logic Programming*, 1070–1080. Seattle, Washington: MIT Press.

Janhunen, T., and Oikarinen, E. 2002. Testing the equivalence of logic programs under stable model semantics. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence*, volume 2424 of *LNAI*, 493–504. Cosenza, Italy: Springer.

Janhunen, T., and Oikarinen, E. 2005. Automated verification of weak equivalence within the SMODELS system. Submitted to Theory and Practice of Logic Programming.

Janhunen, T.; Niemelä, I.; Seipel, D.; Simons, P.; and You, J.-H. 2006. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic* 7(1):1–37.

Janhunen, T. 2003. Translatability and intranslatability results for certain classes of logic programs. Series A: Research report 82, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*. Accepted for publication.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Proc. of the 11th International Conference on Logic Programming*, 23–37. Santa Margherita Ligure, Italy: MIT Press.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Maher, M. J. 1993. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science* 110(2):377–403.

Marek, W., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer-Verlag. 375–398.

Miller, D. 1986. A theory of modules for logic programming. In *Proc. of the 1986 Symposium on Logic Programming*, 106–114. Salt Lake City, USA: IEEE Computer Society Press.

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Math. and Artificial Intelligence* 25(3-4):241–273.

Oikarinen, E., and Janhunen, T. 2004. Verifying the equivalence of logic programs in the disjunctive case. In *Proc. of the 7th International Conference on Logic Programming*

and Nonmonotonic Reasoning, volume 2923 of *LNAI*, 180–193. Fort Lauderdale, USA: Springer.

O'Keefe, R. A. 1985. Towards an algebra for constructing logic programs. In *Proc. of the 1985 Symposium on Logic Programming*, 152–160.

Sagiv, Y. 1987. Optimizing datalog programs. In *Proc. of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 349–362. San Diego, USA: ACM Press.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1–2):181–234.

Turner, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming* 3(4-5):609–622.

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8(3):410–421.

Woltran, S. 2004. Characterizations for relativized notions of equivalence in answer set programming. In *Proc. of the 9th European Conference on Logics in Artificial Intelligence*, volume 3229 of *LNAI*, 161–173. Lisbon, Portugal: Springer.

1.2 A Tool for Advanced Correspondence Checking in Answer-Set Programming

A Tool for Advanced Correspondence Checking in Answer-Set Programming*

Johannes Oetsch
Institut für Informationssysteme 184/3,
Technische Universität Wien,
Favoritenstraße 9-11,
A-1040 Vienna, Austria
oetsch@kr.tuwien.ac.at

Martina Seidl
Institut für Softwaretechnik 188/3,
Technische Universität Wien,
Favoritenstraße 9-11,
A-1040 Vienna, Austria
seidl@big.tuwien.ac.at

Hans Tompits and **Stefan Woltran**
Institut für Informationssysteme 184/3,
Technische Universität Wien,
Favoritenstraße 9-11,
A-1040 Vienna, Austria
{tompits,stefan}@kr.tuwien.ac.at

Abstract

In previous work, a general framework for specifying correspondences between logic programs under the answer-set semantics has been defined. The framework allows to define different notions of equivalence, including well-known notions like *strong equivalence* as well as refined ones based on the *projection* of answer sets, where not all parts of an answer set are of relevance (like, e.g., removal of auxiliary letters). In the general case, deciding the correspondence of two programs lies on the fourth level of the polynomial hierarchy and therefore this task can (presumably) not be efficiently reduced to answer-set programming. In this paper, we describe an implementation to verify program correspondences in this general framework. The system, called cc$\top$, relies on linear-time constructible reductions to *quantified propositional logic* using extant solvers for the latter language as back-end inference engines. We provide some preliminary performance evaluation which shed light on some crucial design issues.

Introduction

Nonmonotonic logic programs under the answer-set semantics (Gelfond & Lifschitz 1991), with which we are dealing with in this paper, represent the canonical and, due to the availability of efficient answer-set solvers, arguably most widely used approach to answer-set programming (ASP). The latter paradigm is based on the idea that problems are encoded in terms of theories such that the solutions of a given problem are determined by the models ("answer sets") of the corresponding theory. Logic programming under the answer-set semantics has become an important host for solving many AI problems, including planning, diagnosis, and inheritance reasoning (cf. Gelfond & Leone (2002) for an overview).

To support engineering tasks of ASP solutions, an important issue is to determine the equivalence of different problem encodings. To this end, various notions of equivalence between programs under the answer-set semantics have been studied in the literature, including the recently proposed framework by Eiter, Tompits, & Woltran (2005), which subsumes most of the previously introduced notions. Within this framework, correspondence between two programs, P and Q, holds iff the answer sets of $P \cup R$ and $Q \cup R$ satisfy certain criteria, for any program R in a specified class, called the *context*. We shall focus here on correspondence problems where both the context and the comparison between answer sets are determined in terms of *alphabets*. This kind of program correspondence includes, as special instances, the well-known notions of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001), *uniform equivalence* (Eiter & Fink 2003), its relativised variants thereof (Woltran 2004), as well as the practicably important case of program comparison under *projected* answer sets. In the last setting, not a whole answer set of a program is of interest, but only its intersection on a subset of all letters; this includes, in particular, removal of auxiliary letters.

For illustration, consider the following two programs which both express the selection of exactly one of the atoms a, b. An atom can only be selected if it can be derived together with the context:

$$\begin{aligned} P = \{\ & sel(b) \leftarrow b, not\ \ out(b); \\ & sel(a) \leftarrow a, not\ \ out(a); \\ & out(a) \vee out(b) \leftarrow a, b; \}. \end{aligned}$$

$$\begin{aligned} Q = \{\ & fail \leftarrow sel(a), not\, a, not\, fail; \\ & fail \leftarrow sel(b), not\, b, not\, fail; \\ & sel(a) \vee sel(b) \leftarrow a; \\ & sel(a) \vee sel(b) \leftarrow b; \}. \end{aligned}$$

Both programs use "local" atoms, $out(\cdot)$ and $fail$, respectively, which are expected not to appear in the context. In order to compare the programs, we could specify an alphabet, A, for the context, for instance $A = \{a, b\}$, or, more generally, any set A of atoms not containing the local atoms $out(a)$, $out(b)$, and $fail$. On the other hand, we want to check whether, for each addition of a context program over A, the answer sets correspond when taking only atoms from $B = \{sel(a),\ sel(b)\}$ into account.

In this paper, we report about an implementation of such correspondence problems together with some initial experimental results. The overall approach of the system, which

*This work was partially supported by the Austrian Science Fund (FWF) under grant P18019; the second author was also supported by the Austrian Federal Ministry of Transport, Innovation, and Technology (BMVIT) and the Austrian Research Promotion Agency (FFG) under grant FIT-IT-810806.

we call cc$\top$ ("correspondence-checking tool"), is to reduce the problem of correspondence checking to the satisfiability problem of *quantified propositional logic*, an extension of classical propositional logic characterised by the condition that its sentences, usually referred to as *quantified Boolean formulas* (QBFs), are permitted to contain quantifications over atomic formulas.

The motivation to use such an approach is twofold. First, complexity results (Eiter, Tompits, & Woltran 2005) show that correspondence checking within this framework is hard, lying on the fourth level of the polynomial hierarchy. This indicates that implementations of such checks cannot be realised in a straightforward manner using ASP systems themselves. In turn, it is well known that decision problems from the polynomial hierarchy can be efficiently represented in terms of QBFs in such a way that determining the validity of the resultant QBFs is not computationally harder than checking the original problem. In previous work (Tompits & Woltran 2005), such translations from correspondence checking to QBFs have been developed; moreover, they are constructible in *linear time and space*. Second, various practicably efficient solvers for quantified propositional logic are currently available (see, e.g., Le Berre *et al.* (2005)). Hence, such tools are used as back-end inference engines in our system to verify the correspondence problems under consideration.

We note that reduction methods to QBFs have been successfully applied already in the field of nonmonotonic reasoning (Egly *et al.* 2000; Delgrande *et al.* 2004), paraconsistent reasoning (Besnard *et al.* 2005; Arieli & Denecker 2003), and planning (Rintanen 1999).

Previous systems implementing different forms of equivalence, being special cases of correspondence notions in the framework of Eiter, Tompits, & Woltran (2005), also based on a reduction approach, are SELP (Chen, Lin, & Li 2005) and DLPEQ (Oikarinen & Janhunen 2004). Concerning SELP, here the problem of checking strong equivalence is reduced to propositional logic, making use of SAT solvers as back-end inference engines. Our system generalises SELP in the sense that cc$\top$ handles a correspondence problem which coincides with a test for strong equivalence by the same reduction as used in SELP. The system DLPEQ, on the other hand, is capable of comparing disjunctive logic programs under ordinary equivalence. Here, the reduction of a correspondence problem results in further logic programs such that the latter have no answer set iff the encoded problem holds. Hence, this system uses answer-set solvers themselves in order to check for equivalence.

The methodologies of both of the above systems have in common that their range of applicability is restricted to very special forms of program correspondences, while our new system cc$\top$ provides a wide range of more fine-grained equivalence notions, allowing practical comparisons useful for debugging and modular programming.

The outline of the paper is as follows. We start with recapitulating the basic facts about logic programs under the answer-set semantics and quantified propositional logic. In describing how to implement correspondence problems, we first give a detailed review of the encodings, followed by a discussion how these encodings (and thus the present system) behave in the case the specified correspondence coincides with special equivalence notions. Then, we address some technical questions which arise when applying the encodings to QBF solvers which require its input to be in a certain normal form. Finally, we present the concrete system cc$\top$ and illustrate its usage. The penultimate section is devoted to experimental evaluation and comparisons. We conclude with some final remarks and pointers to future work.

Preliminaries

Throughout the paper, we use the following notation: For an interpretation I (i.e., a set of atoms) and a set $\mathcal{S}$ of interpretations, we write $\mathcal{S}|_I = \{Y \cap I \mid Y \in \mathcal{S}\}$. For a singleton set $\mathcal{S} = \{Y\}$, we write $Y|_I$ instead of $\mathcal{S}|_I$, if convenient.

Logic Programs

We are concerned with *propositional disjunctive logic programs* (DLPs) which are finite sets of rules of form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n, \quad (1)$$

$n \geq m \geq l \geq 0$, where all a_i are propositional atoms from some fixed universe $\mathcal{U}$ and not denotes default negation. If all atoms occurring in a program P are from a given set $A \subseteq \mathcal{U}$ of atoms, we say that P is a program *over* A. The set of all programs over A is denoted by $\mathcal{P}_A$.

Following Gelfond & Lifschitz (1991), an interpretation I is an *answer set* of a program P iff it is a minimal model of the *reduct* P^I, resulting from P by

- deleting all rules containing default negated atoms $not\, a$ such that $a \in I$, and
- deleting all default negated atoms in the remaining rules.

The collection of all answer sets of a program P is denoted by $\mathcal{AS}(P)$.

In order to semantically compare programs, different notions of equivalence have been introduced in the context of the answer-set semantics. Besides *ordinary equivalence* between programs, which checks whether two programs have the same answer sets, the more restrictive notions of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001) and *uniform equivalence* (Eiter & Fink 2003) have been introduced. Two programs, P and Q, are strongly equivalent iff $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$, for any program R, and they are uniformly equivalent iff $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$, for any set R of *facts*, i.e., rules of form $a \leftarrow$, for some atom a. Also, relativised equivalence notions, taking the alphabet of the extension set R into account, have been defined (Woltran 2004).

In abstracting from these notions, Eiter, Tompits, & Woltran (2005) introduced a general framework for specifying differing notions of program correspondence. In this framework, one parameterises, on the one hand, the *context*, i.e., the class of programs used to be added to the programs under consideration, and, on the other hand, the relation that has to hold between the collection of answer sets of the extended programs. More formally, the following definition has been introduced:

Definition 1 *A* correspondence frame $\mathcal{F}$*, is a triple* $(\mathcal{U}, \mathcal{C}, \rho)$*, where* $\mathcal{U}$ *is a set of atoms, called the* universe of $\mathcal{F}$*,* $\mathcal{C} \subseteq \mathcal{P}_\mathcal{U}$*, called the* context of $\mathcal{F}$*, and* $\rho \subseteq 2^{2^\mathcal{U}} \times 2^{2^\mathcal{U}}$*.*

Two programs $P, Q \in \mathcal{P}_\mathcal{U}$ *are called* $\mathcal{F}$-corresponding*, in symbols* $P \simeq_\mathcal{F} Q$*, iff, for all* $R \in \mathcal{C}$*,* $(\mathcal{AS}(P \cup R), \mathcal{AS}(Q \cup R)) \in \rho$*.*

Clearly, the equivalence notions mentioned above are special cases of $\mathcal{F}$-correspondence. Indeed, for any universe $\mathcal{U}$ and any $A \subseteq \mathcal{U}$, strong equivalence relative to A coincides with $(\mathcal{U}, \mathcal{P}_A, =)$-correspondence, and ordinary equivalence coincides with $(\mathcal{U}, \{\emptyset\}, =)$-correspondence.

Following Eiter, Tompits, & Woltran (2005), we are concerned with correspondence frames of form $(\mathcal{U}, \mathcal{P}_A, \subseteq_B)$ and $(\mathcal{U}, \mathcal{P}_A, =_B)$, where $A, B \subseteq \mathcal{U}$ are sets of atoms and $\subseteq_B$ and $=_B$ are projections of the standard subset and set-equality relation, respectively, defined as follows: for any set $\mathcal{S}, \mathcal{S}'$ of interpretations, $\mathcal{S} \subseteq_B \mathcal{S}'$ iff $\mathcal{S}|_B \subseteq \mathcal{S}'|_B$, and $\mathcal{S} =_B \mathcal{S}'$ iff $\mathcal{S}|_B = \mathcal{S}'|_B$.

A *correspondence problem*, Π, (*over* $\mathcal{U}$) is a quadruple $(P, Q, \mathcal{C}, \rho)$, where $P, Q \in \mathcal{P}_\mathcal{U}$ and $(\mathcal{U}, \mathcal{C}, \rho)$ is a correspondence frame. We say that Π *holds* iff $P \simeq_{(\mathcal{U},\mathcal{C},\rho)} Q$ holds. For a correspondence problem $\Pi = (P, Q, \mathcal{C}, \rho)$ over $\mathcal{U}$, we usually leave $\mathcal{U}$ implicit, assuming that it consists of all atoms occurring in P, Q, and $\mathcal{C}$. We call Π an *equivalence problem* if ρ is given by $=_B$, and an *inclusion problem* if ρ is given by $\subseteq_B$, for some $B \subseteq \mathcal{U}$. Note that $(P, Q, \mathcal{C}, =_B)$ holds iff $(P, Q, \mathcal{C}, \subseteq_B)$ and $(Q, P, \mathcal{C}, \subseteq_B)$ jointly hold.

The next proposition summarises the complexity landscape within this framework (Eiter, Tompits, & Woltran 2005; Pearce, Tompits, & Woltran 2001; Woltran 2004).

Proposition 1 *Given programs* P *and* Q*, sets of atoms* A *and* B*, and* $\rho \in \{\subseteq_B, =_B\}$*, deciding whether a correspondence problem* $(P, Q, \mathcal{P}_A, \rho)$ *holds is:*

1. Π_4^P*-complete, in general;*
2. Π_3^P*-complete, for* $A = \emptyset$*;*
3. Π_2^P*-complete, for* $B = \mathcal{U}$*;*
4. coNP*-complete for* $A = \mathcal{U}$*.*

While Case 1 provides the result in the general setting, for the other cases we have the following: Case 2 amounts to *ordinary equivalence with projection*, i.e., the answer sets of two programs relative to a specified set B of atoms are compared. Case 3 amounts to *strong equivalence relative to* A and includes, as a special case, viz. for $A = \emptyset$, *ordinary equivalence*. Finally, Case 4 includes *strong equivalence* (for $B = \mathcal{U}$) as well as strong equivalence with projection.

The Π_4^P-hardness result shows that, in general, checking the correspondence of two programs cannot (presumably) be efficiently encoded in terms of ASP, which has its basic reasoning tasks located at the second level of the polynomial hierarchy (i.e., they are contained in Σ_2^P or Π_2^P). However, correspondence checking can be efficiently encoded in terms of *quantified propositional logic*, whose basic concepts we recapitulate next.

Quantified Propositional Logic

Quantified propositional logic is an extension of classical propositional logic in which formulas are permitted to contain quantifications over propositional variables. In particular, this language contains, for any atom p, unary operators of form $\forall p$ and $\exists p$, called *universal* and *existential quantifiers*, respectively, where $\exists p$ is defined as $\neg\forall p\neg$. Formulas of this language are also called *quantified Boolean formulas* (QBFs), and we denote them by Greek upper-case letters.

Given a QBF $\mathsf{Q}p\,\Psi$, for $\mathsf{Q} \in \{\exists, \forall\}$, we call Ψ the *scope* of $\mathsf{Q}p$. An occurrence of an atom p is *free* in a QBF Φ if it does not occur in the scope of a quantifier $\mathsf{Q}p$ in Φ. In what follows, we tacitly assume that every subformula $\mathsf{Q}p\,\Phi$ of a QBF contains a free occurrence of p in Φ, and for two different subformulas $\mathsf{Q}p\,\Phi$, $\mathsf{Q}q\,\Psi$ of a QBF, we require $p \neq q$. Moreover, given a finite set P of atoms, $\mathsf{Q}P\,\Psi$ stands for any QBF $\mathsf{Q}p_1\mathsf{Q}p_2 \ldots \mathsf{Q}p_n\Psi$ such that the variables $p_1, \ldots, p_n$ are pairwise distinct and $P = \{p_1, \ldots, p_n\}$. Finally, for an atom p (resp., a set P of atoms) and a set I of atoms, $\Phi[p/I]$ (resp., $\Phi[P/I]$) denotes the QBF resulting from Φ by replacing each free occurrence of p (resp., each $p \in P$) in Φ by $\top$ if $p \in I$ and by $\bot$ otherwise.

For an interpretation I and a QBF Φ, the relation $I \models \Phi$ is inductively defined as in classical propositional logic, whereby universal quantifiers are evaluated as follows:

$$I \models \forall p\ \Phi \text{ iff } I \models \Phi[p/\{p\}] \text{ and } I \models \Phi[p/\emptyset].$$

A QBF Φ is *true under* I iff $I \models \Phi$, otherwise Φ is *false under* I. A QBF is *satisfiable* iff it is true under at least one interpretation. A QBF is *valid* iff it is true under any interpretation. Note that a *closed* QBF, i.e., a QBF without free variable occurrences, is either true under any interpretation or false under any interpretation.

A QBF Φ is said to be in *prenex normal form* (PNF) iff it is closed and of the form

$$\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1\, \phi, \tag{2}$$

where $n \geq 0$, ϕ is a propositional formula, $\mathsf{Q}_i \in \{\exists, \forall\}$ such that $\mathsf{Q}_i \neq \mathsf{Q}_{i+1}$ for $1 \leq i \leq n-1$, $(P_1, \ldots, P_n)$ is a partition of the propositional variables occurring in ϕ, and $P_i \neq \emptyset$, for each $1 \leq i \leq n$. We say that Φ is in *prenex conjunctive normal form* (PCNF) iff Φ is of the form (2) and ϕ is in conjunctive normal form. Furthermore, a QBF of form (2) is also referred to as an (n, Q_n)*-QBF*. Any closed QBF Φ is easily transformed into an equivalent QBF in prenex normal form such that each quantifier occurrence from the original QBF corresponds to a quantifier occurrence in the prenex normal form. Let us call such a QBF the *prenex normal form of* Φ. However, there are different ways to obtain an equivalent prenex QBF (cf. Egly *et al.* (2004) for more details on this issue). The following property is essential:

Proposition 2 *For every* $k \geq 0$*, deciding the truth of a given* $(k, \exists)$*-QBF (resp.,* $(k, \forall)$*-QBF) is* Σ_k^P*-complete (resp.,* Π_k^P*-complete).*

Hence, any decision problem $\mathcal{D}$ in Σ_k^P (resp., Π_k^P) can be mapped in polynomial time to a $(k, \exists)$-QBF (resp., $(k, \forall)$-QBF) Φ such that $\mathcal{D}$ holds iff Φ is valid. In particular any correspondence problem $(P, Q, \mathcal{P}_A, \rho)$, for $\rho \in \{\subseteq_B, =_B\}$, can be reduced in polynomial time to a $(4, \forall)$-QBF. Our implemented tool, described next, relies on two such mappings, which are actually constructible in *linear space and time*.

Computing Correspondence Problems

We now describe the system cc$\top$, which allows to verify the correspondence of two programs. It relies on efficient reductions from correspondence problems to QBFs as developed by Tompits & Woltran (2005). These encodings are presented in the first subsection. Then, we discuss how the encodings behave if the specified correspondence problem coincides with special forms of inclusion or equivalence problems, viz. those restricted cases discussed in Proposition 1. Afterwards, we give details concerning the transformation of the resultant QBFs into PCNF, which is necessary because most extant QBF solvers rely on input of this form. Finally, we give some details concerning the general syntax and invocation of the cc$\top$ tool.

Basic Encodings

Following Tompits & Woltran (2005), we consider two different reductions from inclusion problems to QBFs, $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, where $\mathsf{T}[\cdot]$ can be seen as an explicit optimisation of $\mathsf{S}[\cdot]$. Recall that equivalence problems can be decided by the composition of two inclusion problems. Thus, a composed encoding for equivalence problems is easily obtained via a conjunction of two particular instantiations of $\mathsf{S}[\cdot]$ (or $\mathsf{T}[\cdot]$).

For our encodings, we use the following building blocks. The idea hereby is to use sets of globally new atoms in order to refer to different assignments of the atoms from the compared programs within a single formula. More formally, given an indexed set V of atoms, we assume (pairwise) disjoint copies $V_i = \{v_i \mid v \in V\}$, for every $i \geq 1$. Furthermore, we introduce the following abbreviations:

1. $(V_i \leq V_j) := \bigwedge_{v \in V}(v_i \rightarrow v_j)$;
2. $(V_i < V_j) := (V_i \leq V_j) \wedge \neg(V_j \leq V_i)$; and
3. $(V_i = V_j) := (V_i \leq V_j) \wedge (V_j \leq V_i)$.

Observe that the latter is equivalent to $\bigwedge_{v \in V}(v_i \leftrightarrow v_j)$.

Roughly speaking, these three "operators" allow us to compare different subsets of atoms from a common set, V, under subset inclusion, proper-subset inclusion, and equality, respectively. The comparison takes place within a *single* interpretation while evaluating a formula. As an example, consider $V = \{v, w, u\}$ and an interpretation $I = \{v_1, v_2, w_2\}$, implicitly representing sets $X = \{v\}$ (via the relation $I|_{V_1} = \{v_1\}$) and $Y = \{v, w\}$ (via the relation $I|_{V_2} = \{v_2, w_2\}$). Then, we have that $(V_1 \leq V_2)$ as well as $(V_1 < V_2)$ are true under I which matches the observation that X is indeed a proper subset of Y, while $(V_1 = V_2)$ is false under I reflecting the fact that $X \neq Y$.

In accordance to this renaming of atoms, we use subscripts as a general renaming schema for formulas and rules. That is, for each $i \geq 1$, α_i expresses the result of replacing each occurrence of an atom p in α by p_i, where α is any formula or rule. Furthermore, for a rule r of form (1), we define $H(r) = a_1 \vee \cdots \vee a_l$, $B^+(r) = a_{l+1} \wedge \cdots \wedge a_m$, and $B^-(r) = \neg a_{m+1} \wedge \cdots \wedge \neg a_n$. We identify empty disjunctions with $\bot$ and empty conjunctions with $\top$. Finally, for a program P, we define

$$P_{i,j} = \bigwedge_{r \in P} \big((B^+(r_i) \wedge B^-(r_j)) \rightarrow H(r_i)\big).$$

Formally, we have the following relation: Let P be a program over atoms V, I an interpretation, and $X, Y \subseteq V$ such that, for some i, j, $I|_{V_i} = X_i$ and $I|_{V_j} = Y_j$. Then, $X \models P^Y$ iff $I \models P_{i,j}$. Hence, we are able to characterise models of P (in case that $i = j$) as well as models of certain reducts of P (in case that $i \neq j$).

Having defined these building blocks, we proceed with the first encoding.

Definition 2 *Let P, Q be programs over V, let $A, B \subseteq V$, and let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem. Then,*

$$\mathsf{S}[\Pi] := \neg\exists V_1 \Big(P_{1,1} \wedge \mathsf{S}^1(P, A) \wedge \forall V_3\big(\mathsf{S}^2(Q, A, B) \rightarrow \mathsf{S}^3(P, Q, A)\big)\Big),$$

where

$$\mathsf{S}^1(P, A) := \forall V_2\big(((A_2 = A_1) \wedge (V_2 < V_1)) \rightarrow \neg P_{2,1}\big),$$
$$\mathsf{S}^2(Q, A, B) := \big((A \cup B)_3 = (A \cup B)_1\big) \wedge Q_{3,3}, \text{ and}$$
$$\mathsf{S}^3(P, Q, A) := \exists V_4\big((V_4 < V_3) \wedge Q_{4,3} \wedge \big((A_4 < A_1) \rightarrow \forall V_5(((A_5 = A_4) \wedge (V_5 \leq V_1)) \rightarrow \neg P_{5,1})\big)\big).$$

In fact, the scope, Φ, of $\exists V_1$ encodes the conditions for deciding whether a so-called *partial spoiler* (Eiter, Tompits, & Woltran 2005) for the inclusion problem Π exists. Such spoilers test certain relations on the reducts of the two programs involved, in order to avoid an explicit enumeration of all $R \in \mathcal{P}_A$ for deciding whether Π holds. Such a spoiler for Π exists iff Π does *not* hold. Hence, the resulting encoding Φ is unsatisfiable iff Π holds, and thus the closed QBF $\mathsf{S}[\Pi] = \neg\exists V_1 \Phi$ is valid iff Π holds.

In more concrete terms, given a correspondence problem Π and its encoding $\mathsf{S}[\Pi] = \neg\exists V_1 \Phi$, the general task of the QBF Φ is to test, for an answer-set candidate X of P, that no Y with $Y|_B = X|_B$ becomes an answer set of Q under some implicitly considered extension (in fact, it is sufficient to check only potential candidates Y of the form $Y|_{A \cup B} = X|_{A \cup B}$). Now, the subformula $P_{1,1} \wedge \mathsf{S}^1(P, A)$ tests whether X is such a candidate for P, with X being represented by V_1. In the remaining part of the encoding, $\mathsf{S}^2(Q, A, B)$ returns as its models those potential candidates Y (represented by V_3) for being answer set of Q. These candidates are now checked to be non-minimal and whether there is a further model (represented by V_4) of the reduct of Q with respect to Y surviving an extension of Q, for which X turns into an answer set of the extension of P.

In what follows, we review a more compact encoding which, in particular, reduces the number of universal quantifications. The idea is to save on the fixed assignments, as, e.g., in $\mathsf{S}^2(Q, A, B)$, where we have $(A \cup B)_3 = (A \cup B)_1$. That is, in $\mathsf{S}^2(Q, A, B)$, we implicitly ignore all assignments to V_3 where atoms from A or B have different truth values as the corresponding assignments to V_1. Therefore, it makes sense to consider only atoms from $V_3 \setminus (A_3 \cup B_3)$ and using $A_1 \cup B_1$ instead of $A_3 \cup B_3$ in $Q_{3,3}$.

This calls for a more subtle renaming schema for programs, however. Let $\mathcal{V}$ be a set of indexed atoms, and let r be a rule. Then, $r^{\mathcal{V}}_{i,k}$ results from r by replacing each atom x in r by x_i, providing $x_i \in \mathcal{V}$, and by x_k otherwise. For a

program P, we define

$$P^{\mathcal{V}}_{i,j,k} := \bigwedge_{r \in P} \big((B^+(r^{\mathcal{V}}_{i,k}) \wedge B^-(r^{\mathcal{V}}_{j,k})) \rightarrow H(r^{\mathcal{V}}_{i,k})\big).$$

Moreover, for every $i \geq 1$, every set V of atoms, and every set C, $V_i^C := (V \setminus C)_i$.

Definition 3 *Let P, Q be programs over V and $A, B \subseteq V$. Furthermore, let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem and $\mathcal{V} = V_1 \cup V_2^A \cup V_3^{A \cup B} \cup V_4 \cup V_5^A$. Then,*

$$\mathsf{T}[\Pi] := \neg\exists V_1 \Big(P_{1,1} \wedge \mathsf{T}^1(P, A, \mathcal{V}) \wedge \forall V_3^{A\cup B} \big(Q^{\mathcal{V}}_{3,3,1} \rightarrow \mathsf{T}^3(P, Q, A, \mathcal{V})\big)\Big),$$

where

$$\mathsf{T}^1(P, A, \mathcal{V}) := \forall V_2^A \big((V_2^A < V_1^A) \rightarrow \neg P^{\mathcal{V}}_{2,1,1}\big) \text{ and}$$
$$\mathsf{T}^3(P, Q, A, \mathcal{V}) := \exists V_4 \big(\big(V_4 < ((A\cup B)_1 \cup V_3^{A\cup B})\big) \wedge Q^{\mathcal{V}}_{4,3,1} \wedge \big((A_4 < A_1) \rightarrow \forall V_5^A ((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4})\big)\big).$$

Note that the subformula $V_4 < ((A\cup B)_1 \cup V_3^{A\cup B})$ in $\mathsf{T}^3(P, Q, A, \mathcal{V})$ denotes

$$\big(((A \cup B)_4 \leq (A \cup B)_1) \wedge (V_4^{A\cup B} \leq V_3^{A\cup B})\big) \wedge \neg\big(((A \cup B)_1 \leq (A \cup B)_4) \wedge (V_3^{A\cup B} \leq V_4^{A\cup B})\big).$$

Also note that, compared to our first encoding $\mathsf{S}[\Pi]$, we do not have a pendant to subformula S^2 here, which reduces simply to $Q^{\mathcal{V}}_{3,3,1}$ due to the new renaming schema.

Proposition 3 (Tompits & Woltran 2005) *For any inclusion problem Π, the following statements are equivalent: (i) Π holds; (ii) $\mathsf{S}[\Pi]$ is valid; and (iii) $\mathsf{T}[\Pi]$ is valid.*

In what follows, let, for every equivalence problem $\Pi = (P, Q, \mathcal{P}_A, =_B)$, Π' and Π'' denote the associated inclusion problems $(P, Q, \mathcal{P}_A, \subseteq_B)$ and $(Q, P, \mathcal{P}_A, \subseteq_B)$, respectively.

Corollary 1 *For any equivalence problem Π, the following statements are equivalent: (i) Π holds; (ii) $\mathsf{S}[\Pi'] \wedge \mathsf{S}[\Pi'']$ is valid; and (iii) $\mathsf{T}[\Pi'] \wedge \mathsf{T}[\Pi'']$ is valid.*

Special Cases

We now analyse how our encodings behave in certain instances of the equivalence framework which are located at lower levels of the polynomial hierarchy (cf. Proposition 1). We point out that the following simplifications are correspondingly implemented within our system.

In the case of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001), i.e., problems of form $\Pi = (P, Q, \mathcal{P}_A, =_A)$ with $A = \mathcal{U}$, the encodings $\mathsf{T}[\Pi']$ and $\mathsf{T}[\Pi'']$ can be drastically simplified, since $V_2^A = V_3^A = V_5^A = \emptyset$. In particular, $\mathsf{T}[\Pi']$ is equivalent to

$$\neg\exists V_1 \Big(P_{1,1} \wedge \big(Q_{1,1} \rightarrow \exists V_4 \big((V_4 < V_1) \wedge Q_{4,1} \wedge \neg P_{4,1}\big)\big)\Big).$$

Now, the composed encoding for strong equivalence, i.e., the QBF $\mathsf{T}[\Pi'] \wedge \mathsf{T}[\Pi'']$, amounts to a single propositional unsatisfiability test, witnessing the coNP-completeness complexity for checking strong equivalence (Pearce, Tompits, & Woltran 2001; Lin 2002). This holds also for problems of the form $(P, Q, \mathcal{P}_\mathcal{U}, =_B)$ with arbitrary B. One can show that similar reductions (Pearce, Tompits, & Woltran 2001; Lin 2002) for testing strong equivalence in terms of propositional logic are simple variants thereof. Indeed, the methodology of the tool SELP (Chen, Lin, & Li 2005) is basically mirrored in our approach, in case the parameterisation of the given problem corresponds to a test for strong equivalence.

Strong equivalence *relative* to a set A of atoms (Woltran 2004), i.e., problems of form $(P, Q, \mathcal{P}_A, =_B)$ with $B = \mathcal{U}$, also yields simplifications within $\mathsf{T}[\Pi']$ and $\mathsf{T}[\Pi'']$, since $V_3^{A\cup B} = \emptyset$. In fact, $\mathsf{T}[\Pi']$ can be rewritten to

$$\neg\exists V_1 \big(P_{1,1} \wedge \forall V_2^A \big((V_2^A < V_1^A) \rightarrow \neg P^{\mathcal{V}}_{2,1,1}\big) \wedge \big(Q_{1,1} \rightarrow \exists V_4 \big(\big(V_4 < V_1\big) \wedge Q_{4,1} \wedge \big((A_4 < A_1) \rightarrow \forall V_5^A ((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4})\big)\big)\big)\big).$$

When putting this QBF into prenex normal form (see below), it turns out that the resulting QBF amounts to a $(2, \forall)$-QBF, again reflecting the complexity of the encoded task. Notice that for equivalence problems $(P, Q, \mathcal{P}_A, =_B)$ with $A \cup B = \mathcal{U}$, we also have that $V_3^{A\cup B} = \emptyset$. Thus, the same simplifications also apply for this special case.

The case of ordinary equivalence, i.e., considering problems of form $\Pi = (P, Q, \mathcal{P}_A, =)$ with $A = \emptyset$, is, indeed, a special case of relativised strong equivalence. As an additional optimisation we can drop the subformula

$$(A_4 < A_1) \rightarrow \forall V_5^A \big((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4}\big) \qquad (3)$$

from part T^3 of $\mathsf{T}[\Pi']$. This is because $A = \emptyset$, and therefore

$$(A_4 < A_1) := \bigwedge_{a \in A} (a_4 \rightarrow a_1) \wedge \neg \bigwedge_{a \in A} (a_1 \rightarrow a_4)$$

reduces to $\top \wedge \neg\top$, and thus to $\bot$. Hence, the validity of the implication (3) follows. However, this does not affect the number of quantifier alternations compared to the case of relativised strong equivalence. Indeed, this is in accord with the Π_2^P-completeness for ordinary equivalence. Putting things together, and observing that for $A = \emptyset$ we have $V_2^A = V_2$, the encoding $\mathsf{T}[\Pi']$ results for ordinary equivalence in

$$\neg\exists V_1 \Big(P_{1,1} \wedge \forall V_2 ((V_2 < V_1) \rightarrow \neg P_{2,1}) \wedge (Q_{1,1} \rightarrow \exists V_4 ((V_4 < V_1) \wedge Q_{4,1}))\Big).$$

This encoding is related to encodings for computing answer sets via QBFs, as discussed by Egly *et al.* (2000). Indeed, taking the two main conjuncts from $\mathsf{T}[\Pi']$, viz.

$$P_{1,1} \wedge \forall V_2 ((V_2 < V_1) \rightarrow \neg P_{2,1}) \text{ and} \qquad (4)$$
$$Q_{1,1} \rightarrow \exists V_4 ((V_4 < V_1) \wedge Q_{4,1}), \qquad (5)$$

we get, for any assignment $Y_1 \subseteq V_1$, that Y_1 is a model of (4) iff Y is an answer set of P, and Y_1 is a model of (5) only if Y is not an answer set of Q.

Finally, we discuss the case of ordinary equivalence with projection, i.e., problems of form $(P, Q, \mathcal{P}_A, =_B)$ with $A = \emptyset$. Problems of this form are Π_3^P-complete, and thus we expect our system (after transformation to prenex form) to

yield $(3, \forall)$-QBFs. Here, the only simplification is to get rid off the subformula (3). We can do this for the same reason, viz. since $A = \emptyset$, as above. The simplifications are then as follows (once again using the fact that $V_2^A = V_2$ as well as $V_3^{A \cup B} = V_3^B$):

$$\neg\exists V_1 \big(P_{1,1} \wedge \forall V_2 \big((V_2 < V_1) \rightarrow \neg P_{2,1}\big) \wedge \forall V_3^B \big(Q^{\mathcal{V}}_{3,3,1} \rightarrow \exists V_4 \big(\big(V_4 < (B_1 \cup V_3^B)\big) \wedge Q^{\mathcal{V}}_{4,3,1}\big)\big)\big).$$

Compared to the case of relativised equivalence, as discussed above, this time we have $V_3^{A \cup B} \neq \emptyset$ and thus an additional quantifier alternation "survives" the simplification. After bringing the encoding into its prenex form, we therefore get $(3, \forall)$-QBFs, once again reflecting the intrinsic complexity of the encoded problem.

For the encoding $\mathsf{T}[\cdot]$, the structure of the resulting QBF always reflects the complexity of the correspondence problem according to Proposition 1. This does not hold for formulas stemming from $\mathsf{S}[\cdot]$, however. In any case, our tool implements both encodings in order to provide interesting benchmarks for QBF solvers with respect to their capability to find implicit optimisations for equivalent QBFs.

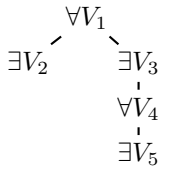
Transformations into Normal Forms

Most available QBF solvers require its input formula to be in a certain normal form, viz. in prenex conjunctive normal form (PCNF). Hence, in order to employ these solvers for our tool, the translations described above have to be transformed by a further two-phased normalisation step:

1. translation of the QBF into prenex normal form (PNF);
2. translation of the propositional part of the formula in PNF into CNF.

Both steps require to address different design issues. In what follows, we describe the fundamental problems, and then briefly provide our solutions in some detail.

First, the step of prenexing is not deterministic. As shown by Egly *et al.* (2004), there are numerous so-called *prenexing strategies*. The concrete selection of such a strategy (also depending on the concrete solver used) crucially influences the running times (see also our results below). In prenexing a QBF, certain *dependencies* between quantifiers have to be respected, when combining the quantifiers of different subformulas to one linear prefix. For our encodings, these dependencies are rather simple and analogous for both encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$. First, observe, however, that both encodings have a negation as their main connective which has to be shifted into the formula by applying the usual equivalence preserving transformations as known from first-order logic. In what follows, we implicitly assume that this step has already been performed. This allows us to consider the quantifier dependencies cleansed with respect to their polarities. The dependencies for the encoding $\mathsf{S}[\cdot]$ can then be illustrated as follows:



Here, the left branch results from the subformula S^1 and the right one results from the subformula $\forall V_3(\mathsf{S}^2(Q, A, B) \rightarrow \mathsf{S}^3(P, Q, A))$.

Inspecting these quantifier dependencies, we can group $\exists V_2$ either together with $\exists V_3$ or with $\exists V_5$. This yields the following two basic ways for prenexing our encodings:

$$\uparrow\colon\ \forall V_1 \exists (V_2 \cup V_3) \forall V_4 \exists V_5; \text{ and } \downarrow\colon\ \forall V_1 \exists V_3 \forall V_4 \exists (V_5 \cup V_2).$$

Together with the two encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, we thus get four different alternatives to represent an inclusion problem in terms of a prenex QBF; we will denote them by $\mathsf{S}_\uparrow[\cdot]$, $\mathsf{S}_\downarrow[\cdot]$, $\mathsf{T}_\uparrow[\cdot]$, and $\mathsf{T}_\downarrow[\cdot]$, respectively. Our experiments below show their different performance behaviour (relative to the employed QBF solver and the benchmarks).

Concerning the transformation of the propositional part of a prenex QBF into CNF, we use a method following Tseitin (1968) in which new atoms are introduced abbreviating subformula occurrences and which has the property that the resultant CNFs are always polynomial in the size of the input formula. Recall that a standard translation of a propositional formula into CNF based on distributivity laws yields formulas of exponential size in the worst case. However, the normal form translation into CNF using labels is not validity preserving like the one based on distributivity laws but only *satisfiability equivalent*. In the case of closed QBFs, the following result holds:

Proposition 4 *Let $\Phi = \mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \phi$, for $\mathsf{Q}_i \in \{\exists, \forall\}$ and $n > 0$, be either an $(n, \forall)$-QBF with n being even or an $(n, \exists)$-QBF with n being odd. Furthermore let ϕ' be the CNF resulting from the propositional part ϕ of Φ by introducing new labels following Tseitin (1968). Then, Φ and $\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \exists V \phi'$ are logically equivalent, where V are the new labels introduced by the CNF transformation.*

Note that for Φ as in the above proposition, we have that $\mathsf{Q}_1 = \exists$. Hence, in this case, $\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \exists V \phi'$ is the desired PCNF, equivalent to Φ, used as input for QBF solvers requiring PCNF format for evaluating Φ. In order to transform a QBF $\Psi = \mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \psi$ which is an $(n, \forall)$-QBF with n being odd or an $(n, \exists)$-QBF with n being even, we just apply the above proposition to $\overline{\mathsf{Q}}_n P_n \ldots \overline{\mathsf{Q}}_1 P_1 \neg\psi$, where $\overline{\mathsf{Q}}_i = \exists$ if $\mathsf{Q}_i = \forall$ and $\overline{\mathsf{Q}}_i = \forall$ otherwise, which is equivalent to $\neg\Psi$. That is, in order to evaluate Ψ by means of a QBF solver requiring PCNF input, we compute $\overline{\mathsf{Q}}_n P_n \ldots \overline{\mathsf{Q}}_1 P_1 \neg\psi$ and "reverse" the output. This is accommodated in cc$\top$ that either the original correspondence problem or the complementary problem is encoded whenever an input yields a QBF like Ψ.

For the entire normal-form transformation, one can use the quantifier-shifting tool `qst` (Zolda 2004). It accepts arbitrary QBFs in `boole` format (see below) as input and returns an equivalent PCNF QBF in *qdimacs* format, which is nowadays a de-facto standard for PCNF-QBF solvers. The tool `qst` implements 14 different strategies (among them $\uparrow$ and $\downarrow$ we use here) to obtain a PCNF and uses the mentioned structure-preserving normal-form transformation for the transformation to CNF.

The Implemented Tool

The system cc$\top$ implements the reductions from inclusion problems $(P, Q, \mathcal{P}_A, \subseteq_B)$ and equivalence problems $(P, Q, \mathcal{P}_A, =_B)$ to corresponding QBFs, together with the potential simplifications discussed above. It takes as input two programs, P and Q, and two sets of atoms, A and B, where A specifies the alphabet of the context and B the set of atoms for projection on the correspondence relation. The reduction (S$[\cdot]$ or T$[\cdot]$) and the type of correspondence problem ($\subseteq_B$ or $=_B$) are specified via command-line arguments: `-S`, `-T` to select the kind of reduction; and `-i`, `-e` to check for inclusion or equivalence between the two programs.

In general, the syntax to specify the programs in cc$\top$ corresponds to the basic DLV syntax.[1] Propositional DLV programs can be passed to cc$\top$ and programs processible for cc$\top$ can be handled by DLV. Considering the example from the introduction, the two programs would be expressed as:

```
P: sel(b) :- b, not out(b).
   sel(a) :- a, not out(a).
   out(a) v out(b) :- a, b.

Q: fail :- sel(a), not a, not fail.
   fail :- sel(b), not b, not fail.
   sel(a) v sel(b) :- a.
   sel(a) v sel(b) :- b.
```

We suppose that file `P.dl` contains the code for program P and, accordingly, file `Q.dl` contains the code for Q. If we want to check whether P is equivalent to Q with respect to the projection to the output predicate $sel(\cdot)$, and restricting the context to programs over $\{a, b\}$, then we need to specify

- the context set, stored in a file, say A, containing the string "`(a, b)`", and
- the projection set, also stored in a file, say B, containing the string "`(sel(a), sel(b))`".

The invocation syntax for cc$\top$ is as follows:

```
ccT -e P.dl Q.dl A B.
```

By default, the encoding T$[\cdot]$ is chosen. Note that the order of the arguments is important: first, the programs P and Q appear, then the context set A, and at last the projection set B. An alternative call of cc$\top$ for our example would be

```
cc⊤ -e -A "(a,b)" -B "(sel(a),sel(b))"
P.dl Q.dl
```

specifying sets A and B directly from the command line. After invocation, the resulting QBF is written to the standard output device and can be processed further by QBF solvers. The output can be piped, e.g., directly to the BDD-based QBF solver `boole`[2] by means of the command

```
ccT -e P.dl Q.dl A B | boole
```

which yields `0` or `1` as answer for the correspondence problem (in our case, the correspondence holds and the output is `1`). To employ further QBF solvers, the output has to be processed according to their input syntax.

If the set A (resp., B) is omitted in invocation, then each variable occurring in P or Q is assumed to be in A (resp., B); if "`0`" is passed instead of a filename, then the empty set is assumed for set A (resp., B). Thus, checking for strong equivalence between P and Q is done by

```
ccT -e P.dl Q.dl | boole
```

while ordinary equivalence (with projection over B) by

```
ccT -e P.dl Q.dl 0 B | boole.
```

We developed cc$\top$ entirely in *ANSI C*; hence, it is highly portable. The parser for the input data was written using *LEX* and *YACC*. The complete package in its current version consists of more than 2000 lines of code. For further information about cc$\top$ and the benchmarks below, see

`http://www.kr.tuwien.ac.at/research/eq/`.

Experimental Results

Our experiments were conducted to determine the behaviour of different QBF solvers in combination with the encodings S$[\cdot]$ and T$[\cdot]$ for inclusion checking, or, if the employed QBF solver requires the input in prenex form, with S$_\uparrow[\cdot]$, S$_\downarrow[\cdot]$, T$_\uparrow[\cdot]$, and T$_\downarrow[\cdot]$. To this end, we implemented a generator of inclusion problems which emanate from the proof of the Π_4^P-hardness of inclusion checking (Eiter, Tompits, & Woltran 2005), and thus provides us with benchmark problems capturing the intrinsic complexity of this task.

The strategy to generate such instances is as follows:

1. generate a $(4, \forall)$-QBF Φ in PCNF by random;
2. reduce Φ to an inclusion problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ such that Π holds iff Φ is valid;
3. apply cc$\top$ to derive the corresponding encoding Ψ for Π.

Incidentally, this procedure also yields a simple method for verifying the correctness of the overall implementation by simply checking whether Ψ is equivalent to Φ. We use here a parameterisation for the generation of random QBFs such that the benchmark set yields a nearly 50% distribution between the true and false instances. Therefore, the set is neither over- nor underconstrained and thus presumably located in a hard region, having easy-hard-easy patterns in mind.

The reduction from the generated QBF Φ to the corresponding inclusion problem is obtained as follows: Consider Φ of form $\forall W \exists X \forall Y \exists Z \phi$, where $\phi = \bigwedge_{i=1}^{n} C_i$ is a formula in CNF over atoms $V = (W \cup X \cup Y \cup Z)$ with $C_i = c_{i,1} \vee \cdots \vee c_{i,k_i}$. Now, let $\bar{V} = \{\bar{v} \mid v \in V\}$ be a set of new atoms, and define $C_i^* = c_{i,1}^*, \ldots, c_{i,k_i}^*$, $v^* = \bar{v}$, and $(\neg v)^* = v$. We generate

$$
\begin{aligned}
P = \{&v \vee \bar{v} \leftarrow \mid v \in V\} \cup \\
&\{v \leftarrow u, \bar{u};\ \bar{v} \leftarrow u, \bar{u} \mid v, u \in V \backslash W\} \cup \\
&\{\leftarrow not\, v;\ \leftarrow not\, \bar{v} \mid v \in V \backslash W\} \cup \\
&\{v \leftarrow C_i^*;\ \bar{v} \leftarrow C_i^* \mid v \in V \backslash W;\ 1 \leq i \leq n\}.
\end{aligned}
$$

For program Q we use further atoms $X' = \{x' \mid x \in X\}$, $\bar{X}' = \{\bar{x}' \mid x \in X\}$ and generate:

[1] See `http://www.dlvsystem.com/` for more information about DLV.

[2] This solver is available at `http://www.cs.cmu.edu/~modelcheck/bdd.html`.

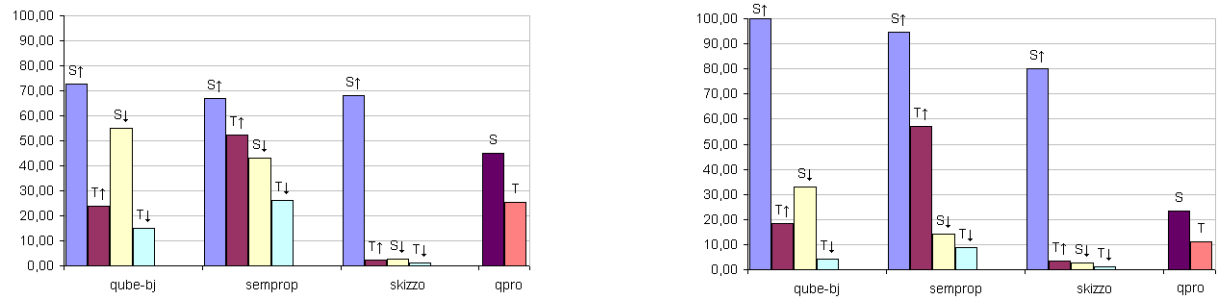


Figure 1: Results for true (left chart) and false (right chart) problem instances subdivided by solvers and encodings.

$$
\begin{aligned}
Q = \{&v \vee \bar{v} \leftarrow \mid v \in X \cup Y\} \cup \\
&\{v \leftarrow u, \bar{u};\ \bar{v} \leftarrow u, \bar{u} \mid v, u \in X \cup Y\} \cup \\
&\{\leftarrow x', \bar{x}';\ \leftarrow not\, x', not\, \bar{x}' \mid x \in X\} \cup \\
&\{v \leftarrow x';\ \bar{v} \leftarrow x';\\
&v \leftarrow \bar{x}';\ \bar{v} \leftarrow \bar{x}' \mid v \in X \cup Y, x \in X\} \cup \\
&\{x' \leftarrow \bar{x}, not\, \bar{x}';\ \bar{x}' \leftarrow x, not\, x' \mid x \in X\}.
\end{aligned}
$$

Finally, sets A and B are defined as:

$$A = B = \{X \cup \bar{X} \cup Y \cup \bar{Y}\}.$$

It can be shown that Φ is valid iff $(P, Q, \mathcal{P}_A, \subseteq_B)$ holds.

We have set up a test series comprising 1000 instances of inclusion problems generated that way (465 of them evaluating to true), where the first program P has 620 rules, the second program Q has 280 rules, using a total of 40 atoms, and the sets A and B of atoms are chosen to contain 16 atoms. After employing cc$\top$, the resulting QBFs possess, in case of translation $\mathsf{S}[\cdot]$, 200 atoms and, in case of translation $\mathsf{T}[\cdot]$, 152 atoms. The additional prenexing step (together with the translation of the propositional part into CNF) yields, in case of $\mathsf{S}[\cdot]$, QBFs with 6575 clauses over 2851 atoms and, in case of $\mathsf{T}[\cdot]$, QBFs with 6216 clauses over 2555 atoms.

We compared four different state-of-the-art QBF solvers, namely `qube-bj` (Giunchiglia, Narizzano, & Tacchella 2003), `semprop` (Letz 2002), `skizzo` (Benedetti 2005), and `qpro` (Egly, Seidl, & Woltran 2006). The former three require QBFs in PCNF as input (thus, we tested them using encodings $\mathsf{S}_{\uparrow}[\cdot]$, $\mathsf{S}_{\downarrow}[\cdot]$, $\mathsf{T}_{\uparrow}[\cdot]$, and $\mathsf{T}_{\downarrow}[\cdot]$), while `qpro` admits arbitrary QBFs as input (we tested it with the non-prenex encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$). Our results are depicted in Figure 1. The y-axis shows the (arithmetically) average running time in seconds (time-out was 100 seconds) for each solver (with respect to the chosen translation and prenexing strategy).

As expected, for all solvers, the more compact encodings of form $\mathsf{T}[\cdot]$ were evaluated faster than the QBFs stemming from encodings of form $\mathsf{S}[\cdot]$. The performance of the prenex-form solvers `qube-bj`, `semprop`, and `skizzo` is highly dependent on the prenexing strategy, and $\downarrow$ dominates $\uparrow$.

For the special case of ordinary equivalence, we compared our approach against the system DLPEQ (Oikarinen & Janhunen 2004) which is based on a reduction to disjunctive logic programs, using `gnt` (Janhunen *et al.* 2006) as answer-set solver. The benchmarks rely on randomly generated $(2, \exists)$-QBFs using Model A (Gent & Walsh 1999). Each QBF is reduced to a program following Eiter & Gottlob (1995), such that the latter possesses an answer set iff the original QBF is valid. The idea of the benchmarks is to compare each such program with one in which one randomly selected rule is dropped, simulating a "sloppy" programmer, in terms of ordinary equivalence.

Average running times are shown in Table 1. The number n of variables in the original QBF varies from 10 to 24, and, for each n, 100 such program comparisons are generated for which the portion of cases where equivalence holds is between 40% and 50% (for details about the benchmarks, cf. Oikarinen & Janhunen (2004)). We set a time-out of 120 seconds, and both the one-phased mode (DLPEQ1) as well as the two-phased mode (DLPEQ2) of DLPEQ were tested. For cc$\top$, we compared the same back-end solvers as above, using encoding $\mathsf{T}[\cdot]$. Recall that for ordinary equivalence cc$\top$ provides $(2, \forall)$-QBFs, thus we can resign on the distinction between prenexing strategies. The dedicated DLPEQ approach turns out to be faster, but, interestingly, among the tested QBF solvers, `qpro` is the most competitive one, while the PCNF-QBF solvers perform bad even for small instances. This result is encouraging as regards further development of the non-normal form approach of QBF solvers.

Conclusion

In this paper, we discussed an implementation for advanced program comparison in answer-set programming via encodings into quantified propositional logic. This approach was motivated by the high computational complexity we have to face for correspondence checking, making a direct realisation via ASP hard to accomplish. Since currently practicably efficient solvers for quantified propositional logic are available, they can be employed as back-end inference engines to verify the correspondence problems under consideration using the proposed encodings. Moreover, since such problems are one of the few natural ones lying above the second level of the polynomial hierarchy, yet still part of the polynomial hierarchy, we believe that our encodings also provide valuable benchmarks for evaluating QBF solvers, for which there is currently a lack of structured problems with more than one quantifier alternation (cf., Le Berre *et al.* (2005)).

References

Arieli, O., and Denecker, M. 2003. Reducing Preferential Paraconsistent Reasoning to Classical Entailment. *Journal*

	qube-bj	semprop	skizzo	qpro	DLPEQ1	DLPEQ2
10	120.00	120.00	14.71	0.05	0.05	0.04
12	120.00	120.00	18.45	0.17	0.06	0.06
14	120.00	120.00	48.70	0.51	0.09	0.08
16	120.00	120.00	120.00	1.54	0.13	0.11
18	120.00	120.00	120.00	4.85	0.19	0.15
20	120.00	120.00	120.00	15.07	0.31	0.25
22	120.00	120.00	120.00	46.23	0.50	0.39
24	120.00	120.00	120.00	120.00	0.84	0.64

	qube-bj	semprop	skizzo	qpro	DLPEQ1	DLPEQ2
10	0.29	56.00	12.27	0.01	0.03	0.03
12	1.49	65.06	18.24	0.02	0.05	0.03
14	5.35	69.35	33.17	0.07	0.05	0.04
16	25.48	86.53	120.00	0.23	0.07	0.06
18	46.10	65.74	120.00	0.50	0.09	0.07
20	82.06	90.34	120.00	1.95	0.20	0.15
22	76.77	86.95	120.00	6.11	0.20	0.15
24	83.68	92.43	120.00	14.81	0.40	0.34

Table 1: Comparing cc$\top$ against DLPEQ on true (left table) and false (right table) problem instances subdivided by solvers.

of Logic and Computation 13(4):557–580.

Benedetti, M. 2005. sKizzo: A Suite to Evaluate and Certify QBFs. In *Proc. CADE'05*, volume 3632 of *LNCS*, 369–376. Springer.

Besnard, P.; Schaub, T.; Tompits, H.; and Woltran, S. 2005. Representing Paraconsistent Reasoning via Quantified Propositional Logic. In *Inconsistency Tolerance*, volume 3300 of *LNCS*, 84–118. Springer.

Chen, Y.; Lin F.; and Li, L. 2005. SELP - A System for Studying Strong Equivalence Between Logic Programs. In *Proc. LPNMR'05*, volume 3552 of *LNAI*, 442–446. Springer.

Delgrande, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2004. On Computing Solutions to Belief Change Scenarios. *Journal of Logic and Computation* 14(6):801–826.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In *Proc. AAAI'00*, 417–422. AAAI Press.

Egly, U.; Seidl, M.; Tompits, H.; Woltran, S.; and Zolda, M. 2004. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. SAT'03. Selected Revised Papers*, volume 2919 of *LNCS*, 214–228. Springer.

Egly, U.; Seidl, M.; and Woltran, S. 2006. A Solver for QBFs in Nonprenex Form. In *Proc. ECAI'06*.

Eiter, T., and Fink, M. 2003. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proc. ICLP'03*, volume 2916 of *LNCS*, 224–238. Springer.

Eiter, T., and Gottlob, G. 1995. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence* 15(3/4):289–323.

Eiter, T.; Tompits, H.; and Woltran, S. 2005. On Solution Correspondences in Answer Set Programming. In *Proc. IJCAI'05*, 97–102.

Gelfond, M., and Leone, N. 2002. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

Gent, I., and Walsh, T. 1999. Beyond NP: The QSAT Phase Transition. In *Proc. AAAI'99*, 648–653. AAAI Press.

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2003. Backjumping for Quantified Boolean Logic Satisfiability. *Artificial Intelligence* 145:99–120.

Janhunen, T.; Niemelä, I.; Seipel, D.; and Simons, P. 2006. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM Transactions on Computational Logic* 7(1):1–37.

Le Berre, D.; Narizzano, M.; Simon, L.; and Tacchella, A. 2005. The Second QBF Solvers Comparative Evaluation. In *Proc. SAT'04. Revised Selected Papers*, volume 3542 of *LNCS*, 376–392. Springer.

Letz, R. 2002. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proc. TABLEAUX'02*, volume 2381 of *LNCS*, 160–175. Springer.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Lin, F. 2002. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. KR'02*, 170–176. Morgan Kaufmann.

Oikarinen, E.; and Janhunen, T. 2004. Verifying the Equivalence of Logic Programs in the Disjunctive Case. *Proc. LPNMR'04*, volume 2923 of *LNCS*, 180–193. Springer.

Pearce, D.; Tompits, H.; and Woltran, S. 2001. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA'01*, volume 2258 of *LNCS*, 306–320. Springer.

Rintanen, J. 1999. Constructing Conditional Plans by a Theorem Prover. *JAIR* 10:323–352.

Tompits, H., and Woltran, S. 2005. Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In *Proc. ICLP'05*, volume 3668 of *LNCS*, 189–203. Springer.

Tseitin, G. S. 1968. On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*. 234–259.

Woltran, S. 2004. Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In *Proc. JELIA'04*, volume 3229 of *LNCS*, 161–173. Springer.

Zolda, M. 2004. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. Master's Thesis, Vienna University of Technology.

1.3 On Probing and Multi-Threading in Platypus

On Probing and Multi-Threading in PLATYPUS

Jean Gressmann
Institut für Informatik
Universität Potsdam
Postfach 900327
D-14439 Potsdam
Germany

Tomi Janhunen
Helsinki University of Technology
Department of Computer Science and Engineering
P.O. Box 5400
FI-02015 TKK
Finland

Robert E. Mercer
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Ontario
Canada N6A 5B7

Torsten Schaub *
Institut für Informatik
Universität Potsdam
Postfach 900327
D-14439 Potsdam, Germany

Sven Thiele
Institut für Informatik
Universität Potsdam
Postfach 900327
D-14439 Potsdam, Germany

Richard Tichy
Institut für Informatik
Universität Potsdam
Postfach 900327
D-14439 Potsdam, Germany

Abstract

The PLATYPUS approach offers a generic platform for distributed answer set solving, accommodating a variety of different modes for distributing the search for answer sets over different processes and/or processors. In this paper, we describe two major extensions of PLATYPUS. First, we present its *probing* approach which provides a controlled non-linear traversal of the search space. Second, we present its new *multi-threading* architecture allowing for intra-process distribution. Both contributions are underpinned by experimental results illustrating their computational impact.

Introduction

The success of Answer Set Programming (ASP) has been greatly enhanced by the availability of highly efficient ASP-solvers (Simons, Niemelä, & Soininen 2002; Leone *et al.* 2006). But, more complex applications are requiring computationally more powerful devices. Distributing parts of the search space among cooperating sequential solvers performing independent searches can provide increased computational power. To accomplish this distribution of the problem solving process, we have proposed a generic approach to distributed answer set solving, called PLATYPUS (Gressmann *et al.* 2005).[1]

The PLATYPUS approach differs from other pioneering work in distributed answer set solving (Finkel *et al.* 2001; Hirsimäki 2001; Pontelli, Balduccini, & Bermudez 2003), by accommodating in a single design a variety of different architectures for distributing the search for answer sets over different processes and processors. The resulting platform,[2] `platypus`, allows one to exploit the increased computational power of clustered and/or multi-processor machines via different types of inter- and intra-process distribution techniques like MPI (Gropp, Lusk, & Thakur 1999), Unix' fork mechanism, and (as discussed in the sequel) multi-threading. In addition, the generic approach permits a flexible instantiation of all parts of the design.

More precisely, the PLATYPUS design incorporates two distinguishing features: First, it modularises (and is thus independent of) the propagation engine (currently exemplified by `smodels`' and `nomore++`' expansion procedures). Second, the search space is represented explicitly. This representation allows a flexible distribution scheme to be incorporated, thereby accommodating different distribution policies and architectures. For instance, the previous `platypus` system (Gressmann *et al.* 2005) supported a multiple process (by forking) and a multiple processor (by MPI (Gropp, Lusk, & Thakur 1999)) architecture. The two particular contributions discussed in this paper take advantage of these two aspects of the generic design philosophy. The first extension to PLATYPUS, *probing*, refines the encapsulated module for propagation. Probing is akin to the concept of *restarting* in the related areas of satisfiability checking (SAT) (Baptista & Marques-Silva 2000; Gomes, Selman, & Kautz 1998) and constraint processing (CSP) (Gomes *et al.* 2000; Walsh 1999). The introduction of probing demonstrates one aspect of the flexibility in our PLATYPUS design: by having a modularised generic design, we can easily specify parts of the generic design to give different computational properties to the `platypus` system. Our second improvement to `platypus` is the integration of multi-threading into our software package.[3] Multi-threading expands the implemented architectural options for delegating the search space and adds several new features to `platypus`: (1) the single- and multi-threaded versions can take advantage of new hardware innovations such as multi-core processors, as well as primitives to implement lock-free data structures, (2) a hybrid architecture which allows

*Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

[1] *platypus*, small densely furred aquatic monotreme of Australia and Tasmania having a broad bill and tail and webbed feet.

[2] We use `typewriter` font when referring to actual systems.

[3] Available at (platypus, website undated).

the mixing of inter- and intra-process distribution, and (3) the intra-process distribution provides a lighter parallelisation mechanism than forking.

In the remainder of this paper we highlight our two contributions, *probing* and *multi-threading*, by focussing on the appropriate aspects of the abstract PLATYPUS algorithm reproduced from (Gressmann *et al.* 2005) below. As well, their computational impact is exposed in data provided by a series of experiments.

Definitions and notation

In Answer Set Programming, a logic program Π is associated with a set $AS(\Pi)$ of *answer sets*, which are distinguished models of the rules in the program. Since we do not elaborate upon theoretical aspects here, we refer the reader to the literature for a formal introduction to ASP (cf. (Gelfond & Lifschitz 1991; Lifschitz 1996; Baral 2003)).

For computing answer sets, we rely on *partial assignments*, mapping atoms in an alphabet $\mathcal{A}$ onto true, false, or undefined. We represent such assignments as pairs (X, Y) of sets of atoms, in which X contains all true atoms and Y all false ones. An answer set X is then represented by the total assignment $(X, \mathcal{A} \setminus X)$. In general, a partial assignment (X, Y) aims at capturing a subset of the answer sets of a logic program Π, viz.

$$AS_{(X,Y)}(\Pi) = \{Z \in AS(\Pi) \mid X \subseteq Z, Z \cap Y \neq \emptyset\} .$$

The PLATYPUS approach and its *probing* mode

To begin, we recapitulate the major features of the PLATYPUS approach (Gressmann *et al.* 2005). To enable a distributed search for answer sets, the search space is decomposed by means of partial assignments. This method works because partial assignments that differ with respect to atoms not in the undefined set represent different parts of the search space. To this end, Algorithm 1 is based on an explicit representation of the search space in terms of a set S of partial assignments, on which it iterates until S becomes empty. The algorithm relies on the omnipresence of a given logic program Π and the program's alphabet $\mathcal{A}$ as global parameters. Communication between PLATYPUS instances is limited to delegating partial assignments as representatives of parts of the search space. The set of partial assignments provided in the input variable S delineates the search space given to a specific instance of PLATYPUS. Although this explicit representation offers an extremely flexible access to the search space, it must be handled with care since it grows exponentially in the worst case. Without Line 9, Algorithm 1 computes all answer sets in $\bigcup_{(X,Y)\in S} AS_{(X,Y)}(\Pi)$. With Line 9 each PLATYPUS instance generates a subset of the answer sets. CHOOSE and DELEGATE are in principle non-deterministic selection functions: CHOOSE yields a single element, DELEGATE communicates a subset of S to a PLATYPUS instance and returns a subset of S. Clearly, depending on what these subsets are, this algorithm is subject to incomplete and redundant search behaviours. The EXPAND function hosts the deterministic part of Algorithm 1. This function is meant to be implemented with an off-the-shelf ASP-expander that is used as a black-box providing both sufficiently strong as well as efficient propagation operations. See (Gressmann *et al.* 2005) for further details.

Algorithm 1: PLATYPUS

Global : A logic program Π over alphabet $\mathcal{A}$.
Input : A nonempty set S of partial assignments.
Output: Print a subset of the answer sets of Π.

```
repeat
1   |  (X, Y) ← CHOOSE(S)
2   |  S ← S \ {(X, Y)}
3   |  (X', Y') ← EXPAND((X, Y))
4   |  if X' ∩ Y' = ∅ then
5   |  |  if X' ∪ Y' = A then
6   |  |  |  print X'
    |  |  else
7   |  |  |  A ← CHOOSE(A \ (X' ∪ Y'))
8   |  |  |  S ← S∪{ (X'∪{A}, Y'), (X', Y'∪{A}) }
9   |  |  S ← DELEGATE(S)
until S = ∅
```

Let us now turn to specific design issues beyond the generic description of Algorithm 1. To reduce the size of partial assignments and thus that of passed messages, we follow (Pontelli, Balduccini, & Bermudez 2003) in representing partial assignments only by atoms[4] whose truth values were assigned by choice operations (cf. atom A in Lines 7 and 8). Given an assignment (X, Y) along with its subsets $X_c \subseteq X$ and $Y_c \subseteq Y$ of atoms assigned by a choice operation, we have $(X, Y) = \text{EXPAND}((X_c, Y_c))$. Consequently, the expansion of assignment (X, Y) to (X', Y') in Line 3 does not affect the representation of the search space in S.[5] Furthermore, the design includes the option of using a choice proposed by the EXPAND component for implementing Line 7. Additionally, the currently used expanders, `smodels` and `nomore++`, also supply a *polarity*, indicating a preference for assigning true or false.[6]

Thread architecture.

The overall design of the `platypus` platform splits Algorithm 1 into two salient components: the `distribution` and the `core`. While the former encapsulates inter-process distribution, the latter handles intra-process distribution and all (sequential) answer set computation methods. For better hardware adaption, the `core` comes in a *single-* and *multi-threaded* version. A thread amounts to a sequential PLATYPUS instance. Since multi-threading and all other distribution aspects are dealt with in the next section, we concentrate in what follows on the non-distributive features of the `core` (equivalent to the single-threaded version).

Each (answer set computing) thread inside the `core` of a `platypus` process has an explicit representation of its

[4] Assignments are not a priori restricted to atoms. This is exploited when using `nomore++`.

[5] Also, some care must be taken when implementing the tests in Lines 4 and 5; see (Gressmann *et al.* 2005).

[6] We rely on this information in Algorithm 3.

(part of the) search space in its variable S. This set of partial assignments is implemented as a tree. Whenever more convenient, we describe S in terms of a set of assignments or a search tree and its branches. In contrast to stack-based ASP-solvers, like `smodels` or `nomore++`, whose search space contains a single branch at a time, this tree normally contains several independent branches. The two major com-

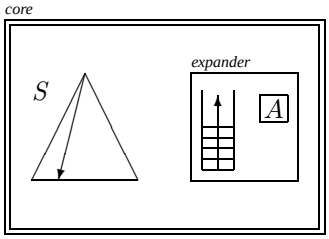


Figure 1: Inner structure of a (single-threaded) `core` module.

ponents of a (single-threaded) `core` along with their interrelationship are depicted in Figure 1. The triangle on the left hand side represents the search tree contained in variable S of Algorithm 1. The vector represents the *active* partial assignment (or branch, respectively) selected in Line 1, and being currently treated by the expander (see below). The square on the right hand side stands for the EXPAND module; the state of the expander is characterised by the contents of its stack, given on the left within the square. The contents of the stack corresponds to the active branch in the search tree (indicated by the usage of an arrow within the stack). While the stack contains the full assignment (X, Y), the search tree's active branch only contains the pair of subsets (X_c, Y_c) having been assigned by choice operations. The box $\boxed{A}$ symbolises the fact that expanders (relying on `smodels` or `nomore++`) also provide a candidate for the choice A made in Line 7 of Algorithm 1.

Probing.

The explicit representation of the (partial) search space, although originally devised to enable the use of a variety of strategies for delegating parts of the search space in the distributed setting, appears to be beneficial in some sequential contexts, as well. Of particular interest, when looking for a single answer set, is limiting fruitless searches in parts of the search tree that are sparsely populated with answer sets. In such cases, it seems advantageous to leave a putatively sparsely populated part and continue at another location in the search space. In `platypus`, this decision is governed by two command line options, $\#c$ and $\#j$. A part of the search is regarded as fruitless, whenever the number of *conflicts* (as encountered in Line 4) exceeds the value of $\#c$. The corresponding conflict counter[7] c is incremented each time a conflict is detected in Line 4 in Algorithm 1. The counter c is *reset* to zero whenever an answer set is found in Line 5 or the active branch in S is switched (and thus the expander is reinitialised; see Algorithm 2). The number of *jumps* in the search space is limited by $\#j$; each jump changes the active branch in the search space. We use a *binary exponential back-off* (cf. (Tanenbaum 2001)) scheme to heed unsuccessful jumps. The idea is as follows. Initially, probing initiates a jump in the search space whenever the initial conflict limit $\#c$ is reached. If no solution is found after $\#j$ jumps, then the problem appears to be harder than expected. In this case, the permissible number of conflicts $\#c$ is doubled and the allowed number of jumps $\#j$ is halved. The former is done to prolong systematic search, the latter to reduce gradually to zero the number of jumps in the search space. We refer to this treatment of the search space as *probing*. Probing is made precise in Algorithm 2, which is a refinement of the CHOOSE operation in Line 1 of Algorithm 1. Note that probing continues until the parameter

Algorithm 2: CHOOSE (in Line 1 of Algorithm 1) in *probing* mode.

Global : Positive integers $\#c, \#j$, initially supplied via command line.
Integers c, j, initially $c = 0$ and $j = \#j$.
Selection policy $\mathcal{P}$, supplied via command line.
Input : A set S of partial assignments with an active assignment $b \in S$.
Output: A partial assignment.

```
begin
    // Counter c is incremented by one in Line 4 of
    Algorithm 1.
    if (c ≤ #c) then // no jumping
        return b
    if (#j = 0) then // no jumping
        return b
    else
        c ← 0
        j ← j − 1
        if (j = 0) then
            #c ← (#c × 2)
            #j ← (#j div 2)
            j ← #j
        let b′ ← SELECT(P, S) in
            make b′ the active partial assignment in S
            return b′
end
```

$\#j$ becomes zero. When probing stops, search proceeds in the usual depth-first manner by considering only one branch at a time by means of the expander's stack. Clearly, this is also the case during the phases when the conflict limit has not been reached ($c \leq \#c$).

At the level of implementation, the expander must be reinitialised whenever the active branch of the search space changes. Reinitialisation is unnecessary when extending the active branch by the choice (obtained in Line 7) in Line 8 of Algorithm 1 or when backtracking is possible in case a conflict or an answer set is obtained. In the first case, the expander's choice (that is, an atom along with a truth value) is simply pushed on top of the expander's stack (and marked as

[7] Each thread has its own conflict and jump counters.

a possible backtracking point). At the same time, the active branch in S is extended by the choice and a copy of the active branch extended by the complementary choice is added to S. The probing refinement of Line 8 in Algorithm 1 is made precise in Algorithm 3.

Algorithm 3: Assignment (in Line 8 of Algorithm 1) in *probing* mode.

Global : A set S of partial assignments with active assignment (X', Y').
Input : An atom A and a constant $P \in \{true, false\}$.
begin
  $S \leftarrow S \cup \{\ (X' \cup \{A\}, Y'),\ (X', Y' \cup \{A\})\ \}$
  if $P = true$ **then**
    make $(X' \cup \{A\}, Y')$ *the active partial assignment in* S
  else
    make $(X', Y' \cup \{A\})$ *the active partial assignment in* S
end

In the case that a conflict occurs or an answer set is obtained, the active branch in S is replaced by the branch corresponding to the expander's stack after backtracking. If it exists, this is the largest branch in S that equals a subbranch of the active branch after switching the truth value of its leaf element. If backtracking is impossible, the active branch is chosen by means of the given policy $\mathcal{P}$.[8] If this, too, is impossible, S must be empty and the PLATYPUS instance terminates.

The policy-driven selection of a branch, expressed by SELECT$(\mathcal{P}, S)$ in Algorithm 2, is governed by another command line option [9] $\#n$ and works in two steps.

1. Among *all* existing branches,[10] the $\#n$ best ones, $b_1, \ldots, b_{\#n}$, are identified according to policy $\mathcal{P}$.

   To be precise, let p be a mapping of branches to ordinal values, used by $\mathcal{P}$ for evaluating branches. For $b \in \{b_1, \ldots, b_{\#n}\}$ and $b' \in S \setminus \{b_1, \ldots, b_{\#n}\}$, we then have that[11] $p(b) \leq p(b')$.

2. A branch b is randomly selected from $\{b_1, \ldots, b_{\#n}\}$.

The random selection from the best $\#n$ branches counteracts the effect of a rigid policy by arbitrarily choosing some close alternatives.

To see that this approach guarantees completeness, it is sufficient to see that no partial assignment is ever eliminated from the search space. Also, when probing, the number of different branches in the search space S cannot exceed twice the number of initially permitted jumps, viz. $2 \times \#j$. For instance, if the command line option sets $\#j$ to 13, we may develop at most $13 + 6 + 3 + 1$ different branches in S, which is bound by 2×13. Thereby, a branch is considered as different if it is not obtainable from another's subbranch by switching the assigned value of a single element.[12]

[8] To this end, `platypus` supports three policies, picking a largest, a smallest, or a random assignment.

[9] Option $\#n$ can be zero, indicating the use of all branches.

[10] This includes all backtracking points.

[11] That is, branches sharing the worst value among the ones in $\{b_1, \ldots, b_{\#n}\}$ may also occur in $S \setminus \{b_1, \ldots, b_{\#n}\}$.

Thread Architecture

In the PLATYPUS algorithm, DELEGATE allows the assigning of answer set computation tasks to other PLATYPUS instances. In the following, we detail the multi-threaded architecture extension to the `platypus` platform which adds intra-process distribution delegation capacities to the existing inter-process distribution delegation capabilities, which are optionally realised via Unix' forking mechanism[13] or MPI (Gropp, Lusk, & Thakur 1999) (described in (Gressmann *et al.* 2005)). This enlarged architecture opens up the possibility of hybrid delegation methods, for instance, delegating `platypus` via MPI on a cluster of multi-processor workstations, with delegation among the multi-processors of the workstation accomplished by means of multi-threading.

The architecture is split into more or less two parts: the `core` and the `distribution` components. The configuration of both components inside a process is depicted in Figure 2. The `core` encapsulates the search for answer sets, and the DELEGATE function is encapsulated in the `distribution` component. The `core` and `distribution` components have well-defined interfaces that localize the communication between the components. This design allows us to incorporate, for instance, single- and multi-threaded cores, as well as inter-process distribution schemes, like MPI and forking, with ease.

Each `platypus` process hosts an instance of the `core`, the core object, which cooperates with one instance of the `distribution` component, the distribution object. Communication is directed from core to distribution objects and is initiated by the core object. During execution the major flow of control lies with the core objects.

The multi-threaded core flow of control works according to the master/slave principle. The master coordinates a number of slave threads (viz. *thread*$_0$ and *thread*$_1$ to *thread*$_n$, respectively, in Figure 2). Each slave thread executes the PLATYPUS algorithm on its thread-local search space, indicated by the respective triangles and boxes as was done in the previous section. The master thread handles communication (through the distribution object) with other `platypus` processes on behalf of the slave threads. Communication between the master thread and its slave threads is based on counters (symbolised by □) and queues (represented by ▥). Similarly to the previous section, we use arrows to indicate partial assignments. Events of interest (e.g. statistics, answer sets, etc.) are communicated by the slave threads to the master thread by incrementing the appropriate counter or adding to the respective queue. The master thread periodically polls the counters and queues for any change. If the change requires information to be transmitted to other `platypus` processes the master thread

[12] This would simply be a backtracking point.

[13] Forking creates duplicate `platypus` processes, collaborating in the search. Communication among them is done using POSIX IPC (handling shared memory and message queues).

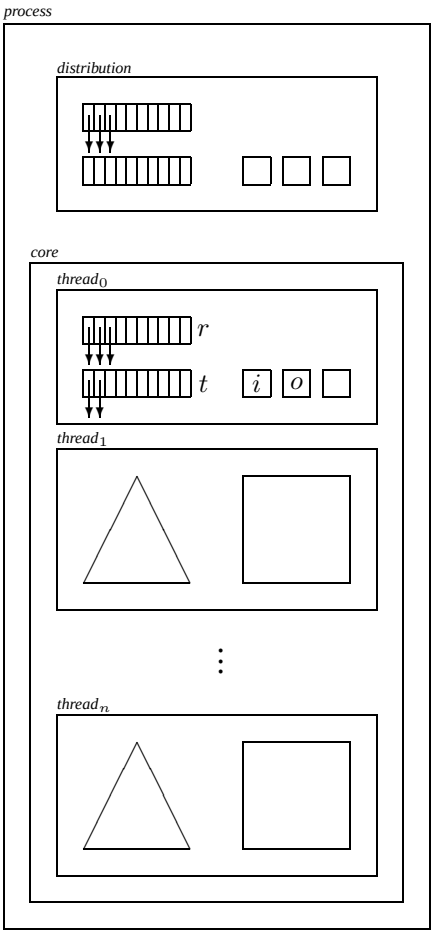


Figure 2: Inner structure of a single process with a multi-threaded *core*.

forwards this information via the distribution object. The search ends (followed by termination of the `platypus` program) if there is agreement among the distribution objects that either all participating processes are in need of work (indicating all the work is done) or the requested number of answer sets has been computed.

Let us now illustrate the communication among core and distribution objects by detailing the major counters and queues. In the `core`, the *idle thread counter* of the master thread (indicated by $\boxed{i}$ in Figure 2) serves two purposes: It indicates the number of idle slave threads in the core object, and it shows the number of partial assignments in the *thread delegation queue* of the master thread (indicated by t). Slave threads share their search space automatically among themselves as long as one thread has some work left. A slave thread running out of work (reaching an empty search space S) checks the availability of work via the idle thread counter and if possible removes a partial assignment from the thread delegation queue. Otherwise, it waits until new work is assigned to it.

A slave thread can become aware of the existence of an idle thread by noting that the idle thread counter exceeds zero during one of its periodic checks. If this is the case, it splits off a subpart of its local search space according to a distribution policy[14], puts the partial assignment that represents the subspace into the thread delegation queue, and decrements the idle thread counter. As this may happen simultaneously in several working slave threads, more partial assignments can end up in the thread delegation queue than there exist idle slaves. These extras are used subsequently by idle threads.

When all slave threads are idle (that is, the idle thread counter equals the number of slave threads.) the master thread initiates communication via the distribution object to acquire more work from other PLATYPUS processes. To this end, the master thread operates in a *polling model*: The master thread periodically queries the associated distribution object for work until it either gets some work or is requested to terminate.[15] Once work is available, the master thread adds it to the thread delegation queue, decrements the idle thread counter,[16] and wakes up a slave thread. The awoken slave thread will find the branch there, take it out, and start working again. From there on, the core enters its normal thread-to-thread mode of work sharing.

Conversely, when a `platypus` process receives notification that another process has run out of work, it attempts to delegate a piece of its search space. To this end, it sets the *other-process-needs-work* flag (indicated by $\boxed{o}$) of the master thread in its core object. All slave threads noticing this flag clear the flag and delegate a piece of their search space according to the delegation policy by adding it to the *remote delegation queue* (indicated by r). The master thread takes one branch out of the queue and forwards it to the requesting `platypus` process (via the distribution object). Because of the multi-threaded nature any number of threads can end up delegating. Items left in the remote delegation queue are used by the master thread to fulfil subsequent requests for work by other `platypus` processes or work requests by its slave threads.

The conceptual difference between the thread delegation and the remote delegation queues is that the former handles intra-core delegations, while the latter deals with extra-core delegation, although non-delegated work can return to the core. This is reflected by the fact that master and slave threads are allowed to insert partial assignments into the thread delegation queue, whereas only slave threads remove items from this queue. In contrast, only the master thread is allowed to eliminate items from the remote delegation queue, while insertions are performed only by slave threads.

Implementation

An important aspect of the multi-threaded core implementation is the use of *lock-free data structures* (Valois 1995; Herlihy 1991; 1993) for synchronizing communication among

[14]Currently, `platypus` supports three policies, picking a largest, a smallest, or a random assignment.

[15]For instance, if the required number of answer sets has already been computed.

[16]The inserting thread is always responsible for decrementing the idle thread counter.

master and slave threads. To be more precise,

- queues (such as the answer set, the thread delegation, and the remote delegation queues) are based on Michael and Scott's FIFO queue (Michael & Scott 1996), and
- counters utilize atomic primitives to implement lock-freedom.

The major benefits of lock-free data structures are that, first, they avoid well-known problems of lock-based approaches such as deadlock, livelock, starvation, and the priority inversion problem (Tanenbaum 2001) and, second, they often provide better performance when contention is high (Michael & Scott 1996). A drawback is that they need hardware support in the form of *universal atomic primitives* (Herlihy 1993). Although not all known data structures have efficient and general-purpose implementations since they require rather powerful atomic primitives (Herlihy 1993), the lock-free data structures used in `platypus` support Intel IA-32, IA-32 with AMD64/EM64T extensions, and SPARC V8/V9 architectures running Linux, Solaris, or Windows, ensuring a broad coverage of major hardware architectures and operating systems.

Experimental Results

The following experiments aim at providing some indications on the computational value of probing and multi-threading. A more detailed empirical evaluation can be found in (Gressmann 2005), being partly mirrored at (platypus, website undated).

All experiments were conducted with some fixed parameters.

- `smodels` (2.28) was used as propagation engine and for delivering the (signed) choice in Line 7 of Algorithm 1,
- the choice in Line 1 of Algorithm 1 was fixed to the policy selecting assignments with the largest number of unassigned atoms,
- all such selections were done in a deterministic way by setting command-line option $\#n$ to 1 (cf. the previous section).

All tests were conducted with `platypus` version 0.2.2 (platypus, website undated). Our results reflect the average times of 5 runs for finding the first or all answer sets, respectively, of the considered instance. Timing excludes parsing and printing. The data was obtained on a quad processor (4 Opteron 2.2GHz processors, 8 GB shared RAM) under `Linux`.

For illustrating the advantage of probing, we have chosen the search for one Hamiltonian cycle in *clumpy graphs*, proposed in (Ward & Schlipf 2004) as a problem set being problematic for systematic backtracking. These benchmarks are available at (platypus, website undated). Table 1 shows the timings for probing running the single-threaded core, with all combinations of settings for the numbers of conflicts $\#c$ (10, 50, 100, 200) and jumps $\#j$ (32, 64, 128, 256, 512), respectively. The entries give the aforementioned average time. For comparison, we also provide the corresponding `smodels` times.[17] as well as the ones for single-threaded `platypus` without probing in the first two columns, labelled *sm* and *st*. The remaining columns are labelled with the used command line options, viz. $\#c, \#j$. A blank entry represents a timeout after 240 seconds.

First of all, we notice that the systems using standard depth first-search are unable to solve 12 instances within the given time limit, whereas when using probing, apart for a few exceptions, all instances are solved. We see that `platypus` without probing does best 8 times,[18] as indicated in boldface, and worst 24 times, whereas `smodels` does best 2 times and worst 24 times. Compared to each specific probing configuration, `platypus` without probing performs better among 9 to 15 (`smodels`, 6 to 8) times out of 38. In fact, there seems to be no clear pattern indicating a best probing configuration. However, looking at the lower part of Table 1, we observe that `platypus` without probing (`smodels`) times out 12 times, while probing still gives a solution under all but three configurations. In all, we see that probing allows for a significant speed-up for finding the first answer set. This is particularly valuable whenever answer sets are hard to find with a systematic backtracking procedure, as witnessed by the entries in the lower part of Table 1.

This improvement is even more impressive when using multi-threading,[19] where further speed-ups were observed on 20 benchmarks, most of which were among the more substantial ones in the lower part of Table 1. The most significant one was observed on clumpy graph 09,09,04 which was solved in 4.66 and 4.26 seconds, respectively, when setting $\#c, \#j$ to 10,512 and using 3 and 4 slave threads, respectively. Interestingly, even the multi-threaded variant *without* probing cannot solve the last seven benchmarks within the time limit, except for clumpy 09,09,07, which `platypus` with 4 slave threads was able to solve in 13.8 seconds. This illustrates that probing and multi-threading are two complementary techniques that can be used for accelerating the performance of standard ASP-solvers. A way to tackle benchmarks that are even beyond the reach of probing with multi-threading is to use randomisation via command-line option $\#n$. Unlike the search for a single answer set, probing has generally no positive effect on the computation of all answer sets. In fact, on more common benchmarks (cf. (asparagus, website undated)) probing rarely kicks in because the conflict counter is reset to zero whenever an answer set is found.

Table 2 displays the effect of multi-threading. For consistency, we have taken a subset of the benchmarks[20] in (Gressmann *et al.* 2005), used when evaluating the speed-ups obtained with the (initial) forking and MPI variant of `platy-`

[17] These times are only of an indicative nature since they include printing one answer set; this cannot be disabled in `smodels`.

[18] The six cases differ by only 0.01sec which is due to slightly different timing methods (see Footnote 17).

[19] The complete set of tests on multi-threading with and without probing are provided at (platypus, website undated).

[20] These benchmarks stem mainly from (asparagus, website undated).

clumpy	*sm*	*st*	10,32	10,64	10,128	10,256	10,512	50,32	50,64	50,128	50,256	50,512	100,32	100,64	100,128	100,256	100,512	200,32	200,64	200,128	200,256	200,512
06,06,02	0.01	**0.01**	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
06,06,03	0.10	0.10	0.05	0.05	0.05	0.05	0.05	0.07	0.07	0.07	0.07	0.07	0.11	0.11	0.11	0.11	0.11	0.17	0.16	0.16	0.16	0.16
06,06,04	0.61	0.63	0.08	0.08	0.08	0.08	0.08	0.14	0.14	0.14	0.14	0.14	0.24	0.24	0.24	0.24	0.24	0.34	0.34	0.34	0.34	0.34
06,06,05	6.30	6.61	1.24	1.79	0.95	0.84	0.84	0.78	0.66	0.66	0.66	0.66	0.96	0.96	0.96	0.96	0.96	2.29	2.14	2.14	2.14	2.14
06,06,06	0.38	0.39	0.05	0.05	0.05	0.05	0.05	0.04	0.04	0.04	0.04	0.04	0.06	0.06	0.06	0.06	0.06	0.10	0.10	0.10	0.10	0.10
06,06,07	0.04	**0.03**	0.14	0.14	0.14	0.14	0.14	0.08	0.08	0.08	0.08	0.08	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
06,06,08	0.08	0.08	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.03
06,06,09	11.3	11.8	0.47	0.52	0.62	0.62	0.62	1.07	1.01	1.01	1.01	1.01	2.23	2.06	2.06	2.06	2.06	3.06	3.46	3.46	3.46	3.46
06,06,10	0.06	0.05	0.03	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.03	0.03	0.05	0.05	0.05	0.05	0.05
07,07,01	0.02	**0.01**	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
07,07,02	0.05	**0.04**	0.61	0.74	0.71	0.71	0.71	1.76	1.45	1.45	1.45	1.45	2.01	2.92	2.91	2.91	2.90	0.04	0.04	0.04	0.04	0.04
07,07,03	8.98	9.60	18.7	9.56	14.5	3.75	3.26	4.79	4.72	16.9	6.11	6.05	5.02	33.8	18.4	9.71	10.3	23.3	9.75	22.1	14.5	14.5
07,07,04	1.37	1.38	0.98	2.05	2.01	3.49	3.38	1.57	1.79	1.54	1.54	1.53	2.87	2.19	2.19	2.20	2.19	2.76	3.30	3.30	3.30	3.28
07,07,05	0.03	**0.02**	0.04	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.02	0.02
07,07,06	**0.38**	**0.38**	0.41	0.38	0.38	0.38	0.38	0.61	0.61	0.61	0.61	0.61	0.69	0.69	0.69	0.69	0.69	0.86	0.86	0.86	0.86	0.86
07,07,07	0.04	**0.03**	0.08	0.08	0.08	0.08	0.08	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03
07,07,08	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.14	0.14	0.14	0.14	0.14
07,07,09	0.40	0.40	0.08	0.08	0.08	0.08	0.08	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.55	0.55	0.55	0.55	0.55
07,07,10	124.5	126.4	15.8	6.32	2.17	1.96	1.97	31.7	13.4	6.01	5.27	5.27	59.3	72.0	9.49	8.74	8.74	18.8	21.5	20.4	14.1	14.1
08,08,01			5.07	1.64	2.44	4.68	5.23	22.5	2.84	3.21	3.22	3.20	10.9	4.81	4.76	4.72	4.68	45.1	15.4	10.3	10.2	10.0
08,08,02			7.04	11.1	2.42	2.44	2.43	8.01	6.22	5.61	6.64	6.61	23.0	12.0	9.74	9.05	8.98	44.0	15.5	13.7	13.8	13.7
08,08,03			14.8	9.39	13.1	5.31	5.52	61.9	84.9	7.57	14.0	13.1	105.8	51.8	9.17	8.71	8.66	32.8	205.8	15.9	15.3	15.3
08,08,05	36.7	37.0	231.2		16.1	33.6	43.6	176.6	24.1	36.1	53.5	96.5	48.3	29.2	47.7	84.1	129.2	70.0	39.4	87.3	189	240
08,08,06	8.15	8.22	0.05	0.05	0.05	0.05	0.05	0.10	0.10	0.10	0.10	0.10	0.16	0.17	0.17	0.17	0.16	0.26	0.26	0.26	0.26	0.26
08,08,07	4.17	4.10	0.44	0.44	0.44	0.44	0.43	1.23	1.24	1.23	1.23	1.23	0.48	0.48	0.48	0.48	0.47	0.89	0.90	0.90	0.90	0.89
08,08,08			0.85	71.6	14.5	6.33	13.5	2.16	1.73	1.73	1.72	1.72	3.69	2.77	2.77	2.77	2.76	6.40	4.76	4.76	4.77	4.75
08,08,09			1.29	0.87	0.88	0.88	0.87	1.07	1.08	1.08	1.08	1.07	2.03	2.03	2.03	2.03	2.02	3.02	3.04	3.03	3.03	3.02
08,08,10	**1.66**	**1.67**	17.3	11.5	4.24	4.37	4.02	1.87	2.24	2.24	2.24	2.23	4.93	2.72	2.72	2.72	2.72	5.97	7.41	7.41	7.40	7.37
09,09,01	24.9	25.0	0.34	0.34	0.34	0.34	0.34	0.10	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.11	0.11	0.12	0.12	0.12	0.12	0.12
09,09,02			1.66	1.82	2.84	2.64	2.63	0.85	0.85	0.85	0.85	0.84	1.48	1.49	1.49	1.49	1.48	2.31	2.32	2.33	2.32	2.31
09,09,03			13.3	4.24	7.33		74.3	0.82	0.82	0.82	0.82	0.82	1.67	1.68	1.68	1.68	1.68	2.51	2.52	2.52	2.52	2.51
09,09,04			143.8				50.9					81.6					95.7					
09,09,05			2.60	2.08	2.66	2.66	2.66	4.03	3.98	4.68	4.68	4.67	3.96	4.80	4.81	4.80	4.79	6.49	6.32	6.31	6.33	6.31
09,09,06			4.00	2.59	159.6	6.40	5.89	11.5	8.62	5.51	5.51	5.50	7.35	21.5	6.45	6.46	6.44	12.8	20.1	17.4	17.4	17.4
09,09,07			0.75	28.4	3.23	3.01	3.01	2.16	2.03	2.04	2.03	2.03	3.05	3.07	3.07	3.06	3.05	6.70	5.95	5.95	5.95	5.90
09,09,09			0.73	0.71	0.71	0.71	0.71	1.95	2.40	2.40	2.40	2.39	3.91	3.50	3.51	3.50	3.48	12.5	9.68	9.67	9.69	9.63

Table 1: Experimental results for *probing* (with the single-threaded core).

pus.[21] Unlike above, we measure the average time (of 5 runs) for computing all answer sets. Comparing the sum of the average times, the current platypus variant running multi-threading is 2.64 times faster than its predecessor using forking, reported in (Gressmann *et al.* 2005). In more detail, the columns reflect the times of platypus run with the multi-threaded core restricted to 1, 2, 3, and 4 slave threads, (with probing disabled).[22] When looking at each benchmark, the experiments show a qualitatively consistent 2-, 3-, and 4-times speed-up when doubling, tripling, and quadrupling the number of processors, with only minor exceptions. For instance, the smallest speed-up was observed on *schur-11-5* (1.52, 1.73, 1.75); among the highest speed-ups, we find *schur-19-4* (2.17, 3.43, 4.75) and *pigeon-7-11* (2.24, 3.43, 4.6). The average speed-ups observed on this set of benchmarks is 1.96, 2.89, and 3.75. However, when taking the weighted average, whose weight is given by the respective average time, we obtain even a slightly super-linear speed-up: 2.07, 3.18, 4.24. Such super-linear speed-ups are observed primarily on time-demanding benchmarks and, although less significant, have also been observed in (Gressmann *et al.* 2005) when forking. In all, we observe that the more substantial the benchmark, the more clear-cut the speed-up. Given that the experiments were run on a quad processor, it is worth noting that we observe no drop in performance when increasing the number of slave threads from 3 to 4, despite having a fifth (master) thread. Finally, we note that the multi-threaded core, when restricted to a single slave thread, exhibits only slightly poorer performance than the single-threaded version: the latter is on average about 2% faster than the former.

At last, we would like to mention that the performance of platypus is currently—under similar circumstances—slightly better when using Unix' fork (along with POSIX IPC for communication) than when using multi-threading.

[21] The forking tests were also run on the same machine.

[22] The numbers in column 'mt #1' are comparable with the ones obtained with smodels or the single-threaded core, respectively. To be more precise, when running smodels and platypus in mode 'mt #1' while printing to /dev/null, we observe an overall factor of 1.59 on the benchmarks in Table 2.

problem	mt #1	mt #2	mt #3	mt #4
color-5-10	1.53	0.84	0.62	0.53
color-5-15	60.9	31.1	20.5	15.7
ham_comp_8	3.66	1.99	1.38	1.10
ham_comp_9	85.2	43.6	29.0	22.5
pigeon-7-8	1.38	0.73	0.57	0.48
pigeon-7-9	4.22	2.19	1.46	1.17
pigeon-7-10	13.2	6.31	4.12	3.08
pigeon-7-11	36.5	16.3	10.6	7.94
pigeon-7-12	88.2	39.9	25.8	19.0
pigeon-8-9	11.6	5.77	3.80	2.84
pigeon-8-10	48.3	22.3	14.2	10.4
pigeon-9-10	128.4	61.8	39.5	29.4
schur-14-4	1.00	0.63	0.47	0.42
schur-15-4	2.38	1.30	0.91	0.73
schur-16-4	4.04	2.14	1.41	1.11
schur-17-4	9.13	4.58	3.04	2.28
schur-18-4	16.7	8.34	5.31	3.92
schur-19-4	39.3	18.1	11.5	8.28
schur-20-4	44.1	21.9	13.8	10.1
schur-11-5	0.56	0.37	0.32	0.32
schur-12-5	1.49	0.83	0.63	0.54
schur-13-5	5.69	2.90	1.97	1.51
schur-14-5	18.6	9.05	6.00	4.42

Table 2: Experimental results on *multi-threading*.

We see two reasons for this. First, forking does not need a master. Second, the current implementation of forking also utilises lock-free data structures where possible (and it thus improves over the one described in (Gressmann *et al.* 2005)).

Discussion

At the heart of the PLATYPUS design is its generality and modularity. These two features allow a great deal of flexibility in any instantiation of the algorithm, making it unique among related approaches. Up to now, this flexibility was witnessed by the possibility to use different off-the-shelf solvers, different process-oriented distribution mechanisms, and a variety of choice policies. In this paper we have presented two significant configurable enhancements to `platypus`.

First, we have described its probing mode, relying on an explicit yet restricted representation of the search space. This provides us with a global view of the search space and allows us to have different threads working on different subspaces. Although probing does not aim at a sequential setting, we have experimentally demonstrated its computational value on a specific class of benchmarks, which is problematic for standard ASP-solvers. Probing offers a non-linear[23] exploration of the search space that can be randomised while remaining complete. Unlike restart strategies in SAT, which usually draw on learnt information (Baptista & Marques-Silva 2000; Gomes, Selman, & Kautz 1998), probing keeps previously abandoned parts of the search space, so that they can be revisited subsequently. Hence, the principal difference between our probing scheme and restarting, known from SAT and CSP, is that probing is *complete* in the sense that it allows the enumeration of all solutions and the detection of no solution. Nonetheless, it would be interesting to see how the various restart strategies in SAT and CSP could be adapted for probing. Restart is implemented in `smodels` and investigated in the context of local search in ASP in (Dimopoulos & Sideris 2002). SAT-based ASP-solvers, such as `assat` (Lin & Zhao 2004) and `cmodels` (Giunchiglia, Lierler, & Maratea 2004), can take advantage of restarts via their embedded SAT-solver.

Second, we have presented `platypus`' multi-threaded architecture. Multi-threading complements the previous process-oriented distribution schemes of `platypus` by providing further intra-process distribution capacities. This is of great practical value since it allows us to take advantage of recent hardware developments, offering multi-core processors. In a hybrid setting, consisting of clusters of such machines, we may use multi-threading for distribution on the multi-core processors, while distribution among different workstations is done with previously established distribution techniques in `platypus`, like MPI. Furthermore, the modular implementation of the *core* and *distribution* component allow for easy modifications in view of new distribution concepts, like grid computing, for instance. The `platypus` platform is freely available on the web (platypus, website undated).

Our experiments have concentrated on highlighting the individual merits of probing and multi-threading. Further systematic studies are needed to investigate their interplay in addition to experiments with different strategies which would include approaches similar to those found in SAT and CSP. Similarly, the relationship between our approach and the work described in (Finkel *et al.* 2001; Hirsimäki 2001; Pontelli, Balduccini, & Bermudez 2003) needs to be studied in more detail.

Acknowledgments The first, fourth, fifth, and sixth author was supported by DFG under grant SCHA 550/6-4. All but the third author were also funded by the EC through IST-2001-37004 WASP project. The third and last authors were funded by NSERC (Canada) and SHARCNET.

We are furthermnore grateful to Christian Anger, Martin Brain, Martin Gebser, Benjamin Kaufmann, and the anonymous referees for many helpful suggestions.

References

`http://asparagus.cs.uni-potsdam.de`.

Baptista, L., and Marques-Silva, J. 2000. Using randomization and learning to solve hard real-world instances of satisfiability. In Dechter, R., ed., *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP'00)*, volume 1894 of *Lecture Notes in Computer Science*, 489–494. Springer-Verlag.

[23] That is, the traversal of the search space does not follow a given strategy like depth-first search.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Dimopoulos, Y., and Sideris, A. 2002. Towards local search for answer sets. In Stuckey, P., ed., *Proceedings of the Eighteenth International Conference on Logic Programming (ICLP'02)*, volume 2401 of *Lecture Notes in Computer Science*, 363–377. Springer-Verlag.

Finkel, R.; Marek, V.; Moore, N.; and Truszczynski, M. 2001. Computing stable models in parallel. In Provetti, A., and Son, T., eds., *Proceedings of AAAI Spring Symposium on Answer Set Programming (ASP'01)*, 72–75. AAAI/MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. A SAT-based polynomial space algorithm for answer set programming. In Delgrande, J., and Schaub, T., eds., *Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning (NMR'04)*, 189–196.

Gomes, C. P.; Selman, B.; Crato, N.; and Kautz, H. A. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reasoning* 24(1/2):67–100.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, 431–437. AAAI Press.

Gressmann, J.; Janhunen, T.; Mercer, R.; Schaub, T.; Thiele, S.; and Tichy, R. 2005. Platypus: A platform for distributed answer set solving. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, 227–239. Springer-Verlag.

Gressmann, J. 2005. Design, implementierung und validierung einer modularen mulithreaded architektur fr platypus. Diplomarbeit, Institut für Informatik, Universität Potsdam.

Gropp, W.; Lusk, E.; and Thakur, R. 1999. *Using MPI-2: Advanced Features of the Message-Passing Interface*. The MIT Press.

Herlihy, M. 1991. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* 13(1):124–149.

Herlihy, M. 1993. A methodology for implementing highly concurrent data objects. *ACM Transactions on Programming Languages and Systems* 15(5):745–770.

Hirsimäki, T. 2001. Distributing backtracking search trees. Technical report, Helsinki University of Technology.

Leone, N.; Faber, W.; Pfeifer, G.; Eiter, T.; Gottlob, G.; Koch, C.; Mateis, C.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*. To appear.

Lifschitz, V. 1996. Foundations of logic programming. In Brewka, G., ed., *Principles of Knowledge Representation*. CSLI Publications. 69–127.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Michael, M. M., and Scott, M. L. 1996. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Symposium on Principles of Distributed Computing*, 267–275.

`http://www.cs.uni-potsdam.de/platypus`.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Nonmonotonic reasoning on beowulf platforms. In Dahl, V., and Wadler, P., eds., *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, volume 2562 of *Lecture Notes in Artificial Intelligence*, 37–57.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Tanenbaum, A. S. 2001. *Modern Operating Systems*. New Jersey, USA: Prentice Hall, 2nd edition.

Valois, J. D. 1995. *Lock-Free Data Structures*. Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, New York.

Walsh, T. 1999. Search in a small world. In Dean, T., ed., *IJCAI*, 1172–1177. Morgan Kaufmann.

Ward, J., and Schlipf, J. 2004. Answer set programming with clause learning. In Lifschitz, V., and Niemelä, I., eds., *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, 302–313. Springer-Verlag.

1.4 Towards Efficient Evaluation of HEX-Programs

Towards Efficient Evaluation of HEX-Programs*

Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits
Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A-1040 Vienna, Austria
{eiter, ianni, roman, tompits}@kr.tuwien.ac.at

Abstract

We briefly report on the development status of dlvhex, a reasoning engine for HEX-programs, which are nonmonotonic logic programs with higher-order atoms and external atoms. Higher-order features are widely acknowledged as useful for various tasks and are essential in the context of meta-reasoning. Furthermore, the possibility to exchange knowledge with external sources in a fully declarative framework such as answer-set programming (ASP) is particularly important in view of applications in the Semantic-Web area. Through external atoms, HEX-programs can deal with external knowledge and reasoners of various nature, such as RDF datasets or description logics bases.

Introduction

Nonmonotonic semantics is often requested by Semantic-Web designers in cases where the reasoning capabilities of the *Ontology layer* of the Semantic Web turn out to be too limiting, since they are based on monotonic logics. The widely acknowledged answer-set semantics of nonmonotonic logic programs (Gelfond & Lifschitz 1991), which is arguably the most important instance of the *answer-set programming* (ASP) paradigm, is a natural host for giving nonmonotonic semantics to the *Rules* and *Logic* layers of the Semantic Web.

In order to address problems such as *meta-reasoning* in the context of the Semantic Web and interoperability with other software, in (Eiter *et al.* 2005), we have extended the answer-set semantics to HEX-*programs*, which are *higher-order logic programs* (which accommodate meta-reasoning through *higher-order atoms*) with *external atoms* for software interoperability. Intuitively, a higher-order atom allows to quantify values over predicate names, and to freely exchange predicate symbols with constant symbols, like in the rule

$$C(X) \leftarrow subClassOf(D, C), D(X).$$

An external atom facilitates the assignment of a truth value of an atom through an external source of computation. For instance, the rule

$$t(Sub, Pred, Obj) \leftarrow \&RDF[uri](Sub, Pred, Obj)$$

computes the predicate t taking values from the predicate $\&RDF$. The latter extracts RDF statements from the set of URIs specified by the extension of the predicate uri; this task is delegated to an external computational source (e.g., an external deduction system, an execution library, etc.). External atoms allow for a bidirectional flow of information to and from external sources of computation such as description logics reasoners. By means of HEX-programs, powerful meta-reasoning becomes available in a decidable setting, e.g., not only for Semantic-Web applications, but also for meta-interpretation techniques in ASP itself, or for defining policy languages.

Other logic-based formalisms, like TRIPLE (Sintek & Decker 2002) or F-Logic (Kifer, Lausen, & Wu 1995), feature also higher-order predicates for meta-reasoning in Semantic-Web applications. Our formalism is fully declarative and offers the possibility of nondeterministic predicate definitions with higher complexity in a decidable setting. This proved already useful for a range of applications with inherent nondeterminism, such as ontology merging (Wang *et al.* 2005) or matchmaking, and thus provides a rich basis for integrating these areas with meta-reasoning.

HEX-Programs

Syntax

HEX programs are built on mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of *constant names*, *variable names*, and *external predicate names*, respectively. Unless stated otherwise, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are written with first letter in upper case (resp., lower case), and elements from $\mathcal{G}$ are prefixed with "&". Constant names serve both as individual and predicate names. Importantly, $\mathcal{C}$ may be infinite.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms and $n \geq 0$ is its *arity*. Intuitively, Y_0 is the predicate name; we thus also use the familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant. For example, $(x, rdf{:}type, c)$ and $node(X)$ are ordinary atoms, while $D(a, b)$ is a higher-order atom. An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m), \qquad (1)$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms

*This work was partially supported by the Austrian Science Fund (FWF) under grant P17212-N04, and by the European Commission through the IST Networks of Excellence REWERSE (IST-2003-506779).

(called *input list* and *output list*, respectively), and $\&g$ is an *external predicate name*.

It is possible to specify *molecules* of atoms in F-Logic-like syntax. For instance, $gi[father \rightarrow X, Z \rightarrow iu]$ is a shortcut for the conjunction $father(gi, X)$, $Z(gi, iu)$.

HEX-programs are sets of rules of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_m, \quad (2)$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are higher-order atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms. The operator "not" is *negation as failure* (or *default negation*).

Semantics

The semantics of HEX-programs is given by generalizing the answer-set semantics (Eiter *et al.* 2005). The *Herbrand base* of a program P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule r, $grnd(r)$, is defined accordingly, and the grounding of program P by $grnd(P) = \bigcup_{r \in P} grnd(r)$. An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms.

We say that an interpretation $I \subseteq HB_P$ is a *model* of an atom $a \in HB_P$ iff $a \in I$. Furthermore, I is a model of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ iff $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$, where $f_{\&g}$ is an $(n+m+1)$-ary Boolean function associated with $\&g$, called *oracle function*, assigning each element of $2^{HB_P} \times \mathcal{C}^{n+m}$ either 0 or 1.

Let r be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a *model* of a HEX-program P, denoted $I \models P$, iff $I \models r$ for all $r \in grnd(P)$.

The *FLP-reduct* of P w.r.t. $I \subseteq HB_P$, denoted fP^I, is the set of all $r \in grnd(P)$ such that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set of* P iff I is a minimal model of fP^I. By $ans(P)$ we denote the set of answer sets of P.

Note that the answer-set semantics may yield no, one, or multiple models (i.e., answer sets) in general. Therefore, for query answering, *brave* and *cautious reasoning* (truth in some resp. all models) is considered in practice, depending on the application.

We have seen that the truth value of an external atom is determined with respect to a specific interpretation, via the domain of the associated Boolean function. As a consequence, the satisfiability of an external atom in general cannot be stated apriori, but only regarding an entire model of a program. This implies not only that external atoms influence the truth values of ordinary atoms by occuring in rule bodies, but also that ordinary atoms can have an effect on the evaluation of external atoms. Hence, HEX-programs facilitate a bidirectional flow of knowledge between the answer set program and the external evaluation function.

In practice, it is useful to differentiate between two kinds of input attributes for external atoms. For an external predicate $\&g$ (exploited, say, in an atom $\&g[p](X)$), a term appearing in an attribute position of type *predicate* (in this case, p) means that the outcomes of $f_{\&g}$ are dependent from the current interpretation I, for what the extension of the predicate named p in I is concerned. An input attribute of type *constant* does not imply a dependency of $f_{\&g}$ from some portion of I. An external predicate whose input attributes are all of type constant does not depend from the current interpretation.

Example 1 The external predicate $\&RDF$ introduced before is implemented with a single input argument of type *predicate*, because its associated function finds the RDF-URIs in the extension of the predicate uri:

$$tr(S, P, O) \leftarrow \&RDF[uri](S, P, O),$$
$$uri(\text{“}file\text{://}foaf.rdf\text{”}) \leftarrow .$$

Should the input argument be of type constant, an equivalent program would be:

$$tr(S, P, O) \leftarrow \&RDF[\text{“}file\text{://}foaf.rdf\text{”}](S, P, O).$$

or

$$tr(S, P, O) \leftarrow \&RDF[X](S, P, O), uri(X),$$
$$uri(\text{“}file\text{://}foaf.rdf\text{”}) \leftarrow .$$

Usability of HEX-Programs

An interesting application scenario, where several features of HEX-programs come into play, is *ontology alignment*. Merging knowledge from different sources in the context of the Semantic Web is a crucial task (Calvanese, Giacomo, & Lenzerini 2001) that can be supported by HEX-programs in various ways:

Importing external theories. This can be achieved by fragments of code such as:

$$triple(X, Y, Z) \leftarrow \&RDF[uri](X, Y, Z),$$
$$triple(X, Y, Z) \leftarrow \&RDF[uri2](X, Y, Z),$$
$$proposition(P) \leftarrow triple(P, rdf{:}type, rdf{:}statement).$$

Searching in the space of assertions. In order to choose nondeterministically which propositions have to be included in the merged theory and which not, statements like the following can be used:

$$pick(P) \vee drop(P) \leftarrow proposition(P).$$

Translating and manipulating reified assertions. For instance, it is possible to choose how to put RDF triples (possibly including OWL assertions) in an easier manipulable and readable format, and to make selected propositions true such as in the following way:

$$(X, Y, Z) \leftarrow pick(P), triple(P, rdf{:}subject, X), triple(P, rdf{:}predicate, Y), triple(P, rdf{:}object, Z),$$
$$C(X) \leftarrow (X, rdf{:}type, C).$$

Defining ontology semantics. The semantics of the ontology language at hand can be defined in terms of entailment rules and constraints expressed in the language itself or in terms of external knowledge, like in

$$D(X) \leftarrow subClassOf(D, C), C(X),$$
$$\leftarrow \&inconsistent[pick],$$

where the external predicate $\&inconsistent$ takes a set of assertions as input and establishes through an external reasoner whether the underlying theory is inconsistent.

Inconsistency of the CWA can be checked by pushing back inferred values to the external knowledge base:

$$set\_false(C, X) \leftarrow cwa(C, C'), C'(X),$$
$$inconsistent \leftarrow \&DL1[set\_false](b),$$

where $\&DL1[N](X)$ effects a check whether a knowledge base, augmented with all negated facts $\neg c(a)$ such that $N(c, a)$ holds, entails the empty concept $\bot$ (entailment of $\bot(b)$, for any constant b, is tantamount to inconsistency).

Implementation

The challenge of implementing a reasoner for HEX-programs lies in the interaction between external atoms and the ordinary part of a program. Due to the bidirectional flow of information represented by its input list, an external atom cannot be evaluated prior to the rest of the program. However, the existence of established and efficient reasoners for answer-set programs led us to the idea of splitting and rewriting the program such that an existing answer-set solver can be employed alternatingly with the external atoms' evaluation functions. In the following, we will outline methods that are already implemented in our prototype HEX reasoner dlvhex. We will partly refer to (Eiter *et al.* 2006), modifying the algorithms and concepts presented there where it is appropriate in the view of an actual implementation.

Dependency Information

Taking the dependency between heads and bodies into account is a common tool for devising an operational semantics for ordinary logic programs, e.g., by means of the notions of *stratification* or *local stratification* (Przymusinski 1988), or through *modular stratification* (Ross 1994) or *splitting sets* (Lifschitz & Turner 1994). In (Eiter *et al.* 2006), we defined novel types of dependencies, considering that in HEX programs, dependency between heads and bodies is not the only possible source of interaction between predicates. Contrary to the traditional notion of dependency based on propositional programs, we consider relationships between nonground, higher-order atoms. In the view of an actual implementation of a dependency graph processing algorithm, we will present in the following a generalized definition of atom dependency of (Eiter *et al.* 2006).

Definition 1 *Let P be a program and a, b atoms occurring in some rule of P. Then, a* depends positively *on b ($a \rightarrow_p b$), if one of the following conditions holds:*

1. *There is some rule $r \in P$ such that $a \in H(r)$ and $b \in B^+(r)$.*
2. *There are some rules $r_1, r_2 \in P$ such that $a \in B(r_2)$ and $b \in H(r_1)$ and there exists a partial substitution θ of variables in a such that either $a\theta = b$ or $a = b\theta$. E.g., $H(a, Y)$ unifies with $p(a, X)$.*
3. *There is some rule $r \in P$ such that $a, b \in H(r)$. Note that this relation is symmetric.*
4. *a is an external predicate of form $\&g[\bar{X}](\bar{Y})$ where $\bar{X} = X_1, \ldots, X_n$, and b is of form $p(\bar{Z})$, and, for some i, $X_i = p$ and of type predicate (e.g., $\&count[item](N)$ is externally dependent on $item(X)$).*

Moreover, a depends negatively *on b ($a \rightarrow_n b$), if there is some rule $r \in P$ such that $a \in H(r)$ and $b \in B^-(r)$. We say that a depends on b, if $a \rightarrow b$, where $\rightarrow = \rightarrow_p \cup \rightarrow_n$. The relation $\rightarrow^+$ denotes the transitive closure of $\rightarrow$.*

These dependency relations let us construct a graph, which we call *dependency graph* of the corresponding program.

Example 2 Consider the program of Figure 1, modeling the search for personal contacts that stem from a *FOAF-ontology*,[1] which is accessible by a URL.
The first two facts specify the URLs of the FOAF ontologies we want to query. Rules 3 and 4 ensure that each answer set will be based on a single URL only. Rule 5 extracts all triples from an RDF file specified by the extension of $input$. Rule 6 converts triples that assign names to individuals into the predicate $name$. Finally, the last rule traverses the RDF graph to construct the relation $knows$.

Figure 2 shows the dependency graph of P.[2]

Evaluation Strategy

The principle of evaluation of a HEX-program relies on the theory of *splitting sets*. Intuitively, given a program P, a splitting set S is a set of ground atoms that induce a sub-program $grnd(P') \subset grnd(P)$ whose models $\mathcal{M} = \{M_1, \ldots, M_n\}$ can be evaluated separately. Then, an adequate *splitting theorem* shows how to plug in $\mathcal{M}$ in a modified version of $P \setminus P'$ so that the overall models can be computed. Here, we use a modified notion of splitting set, accomodating non-ground programs and suited to our definition of dependency graph.

Definition 2 *A* global splitting set *for a HEX-program P is a set of atoms A appearing in P, such that whenever $a \in A$ and $a \rightarrow b$ for some atom b appearing in P, then also $b \in A$.*

In (Eiter *et al.* 2006), we already defined an algorithm based on splitting sets. However, there we used a general approach, decomposing P into strongly connected components (SCC in the following), which leads to a potentially large number of splitting sets (considering that a single atom that does not occur in any cycle is a SCC by itself). However, since the evaluation of each splitting set requires an interaction with an answer-set solver (i.e., one or more calls to the solver, depending on the nature of the program associated with the splitting set), in a practical setting the object must be to identify as few splitting sets as possible in order to minimize the number of actual reasoning steps and

[1] "FOAF" stands for "Friend Of A Friend", and is an RDF vocabulary to describe people and their relationships.

[2] Long constant names have been abbreviated for the sake of compactness.

(1) $url(\text{“}http://www.kr.tuwien.ac.at/staff/roman/foaf.rdf\text{”}) \leftarrow;$
(2) $url(\text{“}http://www.mat.unical.it/ianni/foaf.rdf\text{”}) \leftarrow;$
(3) $\neg input(X) \vee \neg input(Y) \leftarrow url(X), url(Y), X \neq Y;$
(4) $input(X) \leftarrow not \neg input(X), url(X);$
(5) $triple(X,Y,Z) \leftarrow \&RDF[A](X,Y,Z), input(A);$
(6) $name(X,Y) \leftarrow triple(X, \text{“}http://xmlns.com/foaf/0.1/name\text{”}, Y);$
(7) $knows(X,Y) \leftarrow name(A,X), name(B,Y), triple(A, \text{“}http://xmlns.com/foaf/0.1/knows\text{”}, B).$

Figure 1: Example program using the $\&RDF$-atom.

increase overall efficiency. Therefore, we now modify and specialize the notions and methods given there.

Definition 3 *A* local splitting set *for a* HEX*-program P is a set of atoms A appearing in P, such that for each atom $a \in A$ there is no atom $b \notin A$ such that $a \rightarrow b$ and $b \rightarrow^{+} a$.*

Thus, contrary to a global splitting set, a local splitting set does not necessarily include the lowest layer of the program, but it never “breaks” a cycle.

Definition 4 *The* bottom *of P w.r.t. set of atoms A is the set of rules $b_A(P) = \{r \in P \mid H(r) \cap A \neq \emptyset\}$.*

We define the concept of *external component*, which represents a part of the dependency graph including at least one external atom. Intuitively, an external component is the minimal local splitting set that contains one or more external atoms. We distinguish between different types of external components, each with a specific procedure of evaluation, i.e., computing its model(s) w.r.t. to a set of ground atoms I. Before these are laid out, we need to introduce some auxiliary notions.

From the viewpoint of program evaluation, it turns out to be impractical to define the semantics of an external predicate by means of a Boolean function. Again restricting the concepts presented in (Eiter *et al.* 2006) for our practical needs, we define $F_{\&g} : 2^{HB_P} \times D_1, \ldots, D_n \rightarrow 2^{R_{\mathcal{C}}^m}$ with $F_{\&g}(I, y_1, \ldots, y_n) = \langle x_1, \ldots, x_m \rangle$ iff $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$, where $R_{\mathcal{C}}^m$ is the set of all tuples of arity m that can be built with symbols from $\mathcal{C}$. If the input list $y_1, \ldots, y_n$ is not ground in the original program, safety restrictions for HEX-programs ensure that its values can be determined from the remaining rule body.

A ground external atom $\&g$ is monotonic providing $I \models \&g$ implies $I' \models \&g$, for $I \subseteq I' \subseteq HB_P$.

With P_{hex}, we denote the ordinary logic program having each external atom $\&g[\overline{y}](\overline{x})$ in P replaced by $d_{\&g}(\overline{y}, \overline{x})$ (we call this kind of atoms *replacement atoms*), where $d_{\&g}$ is a fresh predicate symbol.

The categories of external components we consider are:

- A single external atom $\&g$ that does not occur in any cycle. Its evaluation method returns for each tuple $\langle x_1, \ldots, x_m \rangle$ in $F_{\&g}(I, y_1, \ldots, y_n)$ a ground replacement atom $d_{\&g}(y_1, \ldots, y_n, x_1, \ldots, x_m)$ as result. The external atom in Figure 2, surrounded by a rectangular box, represents such a component.
- A strongly connected component C without any weakly negated atoms and only monotonic external atoms. A simple method for computing the (unique) model of such a component is given by the fixpoint operation of the operator $\Lambda : 2^{HB_P} \rightarrow 2^{HB_P}$, defined by $\Lambda(I) = M(P_{hex} \cup D_P(I)) \cap HB_P$, where:
  - P_{hex} is an ordinary logic program as defined above, with $P = b_C$.
  - $D_P(I)$ is the set of all facts $d_{\&g}(\overline{y}, \overline{c}) \leftarrow$ such that $I \models \&g[\overline{y}](\overline{c})$ for all external atoms $\&g$ in P; and
  - $M(P_{hex} \cup D_P(I))$ is the single answer set of $P_{hex} \cup D_P(I)$; since P_{hex} is stratified, this answer set is guaranteed to exist and to be unique.
- A strongly connected component C with negative dependencies or nonmonotonic external atoms. In this case, we cannot rely on an iterative approach, but are forced to guess the value of each external atom beforehand and validate each guess w.r.t. the remaining atoms:
  - Construct P_{hex} from $P = b_C$ as before and add for each replacement atom $d_{\&g}(\overline{y}, \overline{x})$ all rules

$$d_{\&g}(\overline{y}, \overline{c}) \vee \neg d_{\&g}(\overline{y}, \overline{c}) \leftarrow \tag{3}$$

    such that $\&g[\overline{y}](\overline{c})$ is a ground instance of $\&g[\overline{y}](\overline{x})$. Intuitively, the rules (3) “guess” the truth values of the external atoms of C. Denote the resulting program by P_{guess}.
  - Compute the answer sets $Ans = \{M_1, \ldots, M_n\}$ of P_{guess}.
  - For each answer set $M \in Ans$ of P_{guess}, test whether the original “guess” of the value of $d_{\&g}(\overline{y}, \overline{c})$ is compliant with $f_{\&g}$. That is, for each external atom a, check whether $M \models \&g[\overline{y}](\overline{c})$. If this condition does not hold, remove M from Ans.
  - Each remaining $M \in Ans$ is an answer set of P iff M is a minimal model of fP_{hex}^M.

Note that a cyclic subprogram must preserve certain safety rules in order to bound the number of symbols to be taken into account to a finite extent. To this end, we defined in (Eiter *et al.* 2006) the notion of *expansion-safety*, which avoids a potentially infinite ground program while still allowing external atoms to bring in additional symbols to the program.

The evaluation algorithm in (Figure 3) uses the following subroutines:

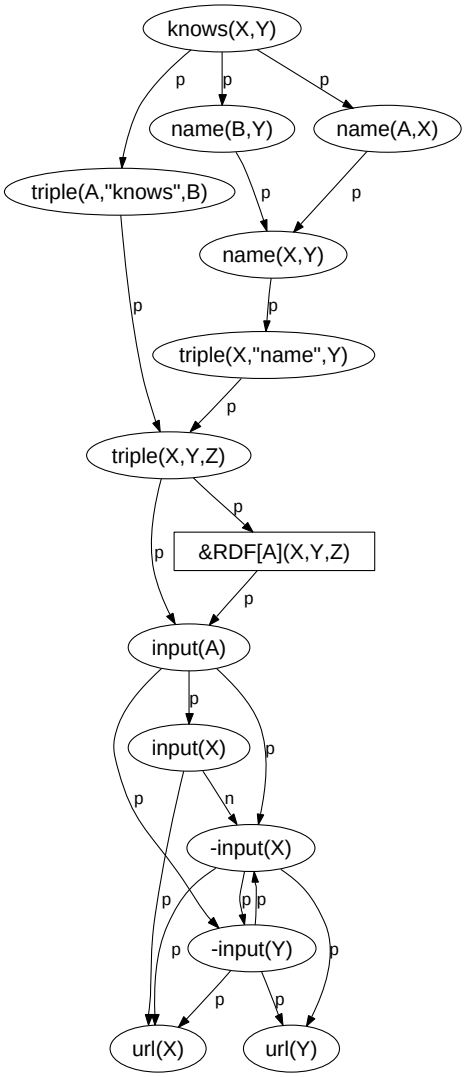


Figure 2: FOAF program graph.

$eval(comp, \mathcal{I})$ Computes the models of an external component $comp$ (which is of one of the types described above) for each interpretation $I \in \mathcal{I}$; each I is added as a set of facts to the respective models.

$solve(P, I)$ Returns the answer sets of $P \cup A$, where P does not contain any external atom and A is the set of facts that corresponds to I.

Intuitively, the algorithm traverses the dependency graph from bottom to top, gradually pruning it while computing the respective models. Step (a) singles out all external components that do not depend on any further atom or component, i.e., that are on the "bottom" of the dependency graph. Those components are evaluated against the current known models in Step (b) and can be removed from the list of external components that are left to be solved. Moreover, Step (b) ensures that all rules of these components are removed from the program. From the remaining part, Step(d) extracts the

EVALUATION ALGORITHM
(Input: a HEX-program P; Output: a set of models $\mathcal{M}$)

1. Determine the dependency graph G for P.
2. Find all external components C_i of P and build $Comp = \{C_1, \ldots, C_n\}$.
3. Set $T := Comp$ and $\mathcal{M} := \{F\}$, where F is the set of all facts originally contained in P. The set $\mathcal{M}$ will eventually contain $ans(P)$ (which is empty, in case inconsistency is detected).
4. While $P \neq \emptyset$ do
   (a) Let $\overline{T} := \{C \in T \mid \forall a \in C : \text{if } \exists a \rightarrow b \text{ then } b \in C\}$.
   (b) Let $\mathcal{M}' := \{\emptyset\}$; for each C from $\overline{T}$:
   - let $\mathcal{M}' := \{\mathcal{A} \bigcup \mathcal{B} \mid (\mathcal{A}, \mathcal{B}) \in \mathcal{M}' \times eval(C, \mathcal{M})\}$,
   - remove C from $Comp$ and
   - let $P := P \setminus b_c(P)$.
   Let $\mathcal{M} := \mathcal{M}'$.
   (c) if $\mathcal{M} = \emptyset$ then halt.
   (d) Let $\mathcal{M} := \bigcup_{M \in \mathcal{M}} solve(P', M)$, where $P' = P_{hex} \setminus b_{\overline{C}}$ and $\overline{C}$ is the set of all nodes u such that either $u \rightarrow^+ c$ with $c \in C$ or $u \in C$ for any $C \in Comp$;
   let $P := P \setminus P'$ and remove all atoms from the graph that are not in $\overline{C}$.

Figure 3: Evaluation algorithm.

largest possible subprogram that does not depend on any remaining external component, computes its models and removes it from the program resp. dependency graph.

Seen from a more general perspective, the iteration traverses the program graph by applying two different evaluation functions each turn. While $eval$ computes minimal subprograms containing external atoms, $comp$ solves maximal non-external subprograms.

Let us exemplarily step through the algorithm with Example 2 as input program P. First, the graph G is constructed as shown in Figure 2. Since P contains only a single external atom, the set $Comp$ constructed in Step 2 contains just one external component C, the $\&RDF$-atom itself. Step (a) extracts those components of $Comp$ that form a global splitting set, i.e., that do not depend on any atom not in the component. Clearly, this is not the case for C and hence, $\overline{T}$ is empty. Step (d) constructs an auxiliary program P' by removing the bottom of $\overline{C}$, which contains each component that is still in $Comp$ and every atom "above" it in the dependency graph:

$\neg input(X) \vee \neg input(Y) \leftarrow url(X), url(Y), X \neq Y;$
$input(X) \leftarrow not \neg input(X), url(X);$

$solve(P', M)$ in Step (d) yields the answer sets of P', where M is the set of the original facts from P (the two URIs). P' is removed from P and $\overline{C}$ from the dependency graph (the resulting subgraph is shown in Figure 4). Continuing with (a), now the external component C is contained in T_c, and therefore in Step (b) evaluated for each set in $\mathcal{M}$. After removing C from $Comp$, $\overline{C}$ is empty in Step (d) and

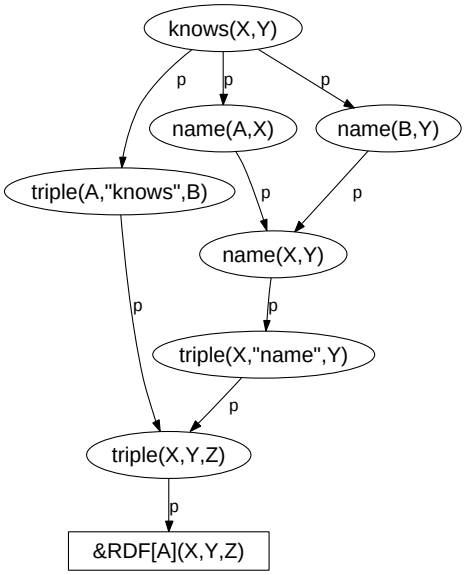


Figure 4: Pruned dependency graph.

$P' = P_{hex}$, i.e., an ordinary, stratified program, which is evaluated against each set in $\mathcal{M}$ - note that these sets now also contain the result of the external atom, represented as ground replacement atoms. At this point, P is empty and the algorithm terminates, having $\mathcal{M}$ as result.

We obtain the following property:

Theorem 1 *Let P be a* HEX*-program and $\mathcal{M}$ the output of the evaluation algorithm from Figure 3. Then, M is an answer set of P iff $M \in \mathcal{M}$.*

Proof 1 (Sketch). The given algorithm is actually a repeated application of the splitting set theorem as introduced in (Lifschitz & Turner 1994) and extended to programs with external atoms in (Eiter *et al.* 2006). Basically, the theorem allows to state that if U is a splitting set for a program P, then, a set A is an answer set of program P iff A is an answer set of $P' = (P \setminus b_U) \cup B$ where B contains the facts corresponding to some answer set of b_U.

Given the current value of P, Step (a) of the algorithm finds splitting sets corresponding to external components of P. The splitting set theorem is applied by computing the answer sets of the bottoms of each of these components. If one of the components is found to be inconsistent, then the entire program must be inconsistent and no answer set exists (Step (c)). Step (d) again applies the splitting set theorem on the remaining program. In this case, the splitting set which is searched for does not contain external atoms. After each iteration of the algorithm, the set of final answer sets is updated, while P is reduced. Finally, all answer sets of P are left.

Available External Atoms

External Atoms are provided by so-called *plugins*, i.e., libraries that define one or more external atom functions. Currently, we implemented the *RDF plugin*, the *Description Logics Plugin* and the *String Plugin*.

The RDF Plugin RDF (Resource Description Framework) is a language for representing information about resources in the World-Wide Web and is intended to represent meta-data about Web resources which is machine-readable and -processable. RDF is based on the idea of identifying objects using Web identifiers (called *Uniform Resource Identifiers*, or URIs), and describing resources in terms of simple properties and property values. The *RDF plugin* provides a single external atom, the $\&RDF$-atom, which enables the user to import RDF-triples from any RDF knowledge base. It takes a single constant as input, which denotes the RDF-source (a file path or Web address).

The Description-Logics Plugin Description logics are an important class of formalisms for expressing knowledge about concepts and concept hierarchies (often denoted as *ontologies*). The basic building blocks are *concepts*, *roles*, and *individuals*. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates interpreted as sets of objects. Roles are interpreted as binary relations between objects. In previous work (Eiter *et al.* 2004), we introduced *dl-programs* as a method to interface description-logic knowledge bases with answer-set programs, allowing a bidirectional flow of information. To model dl-programs in terms of HEX-programs, we developed the *description-logics plugin*, which includes three external atoms (these atoms, in accord to the semantics of dl-programs, also allow for extending a description logic knowledge base, before submitting a query, by means of the atoms' input parameters):

- the $\&dlC$ atom, which queries a concept (specified by an input parameter of the atom) and retrieves its individuals,
- the $\&dlR$ atom, which queries a role and retrieves its individual pairs, and
- the $\&dlConsistent$ atom, which tests the (possibly extended) description logic knowledge base for consistency.

The description-logics plugin can access OWL ontologies, i.e., description logic knowledge bases in the language $\mathcal{SHOIN}(\mathbf{D})$, utilizing the RACER reasoning engine (Haarslev & Möller 2001).

The String Plugin For simple string manipulation routines, we provide the string plugin. It currently consists of two atoms:

- the $\&concat$ atom, which lets the user specify two constant strings in the input list and returns their concatenation as a single output value, and
- the $\&strstr$ atom, which tests two strings for substring inclusion.

Current Prototype

dlvhex has been implemented as a command-line application. It takes one or more HEX-programs as input and directly prints the resultant models as output. Both input and output are given in classical textual logic-programming

notation. For the core reasoning process, dlvhex itself needs the answer-set solver DLV (Leone *et al.* 2005) (and DLT (Ianni *et al.* 2004) if F-Logic syntax is used).

Assuming that the program from Example 2 is represented by the file `rdf.lp`, dlvhex is called as follows:

```
user@host:~> dlvhex --filter=friend rdf.lp
```

The `--filter` switch reduces the output of facts to the given predicate names. The result contains two answer sets:

```
{knows("Giovambattista Ianni",
       "Axel Polleres"),
{knows("Giovambattista Ianni",
       "Francesco Calimeri"),
{knows("Giovambattista Ianni",
       "Wolfgang Faber"),
{knows("Giovambattista Ianni",
       "Roman Schindlauer")}

{knows("Roman Schindlauer",
       "Giovambattista Ianni"),
{knows("Roman Schindlauer",
       "Wolfgang Faber"),
{knows("Roman Schindlauer",
       "Hans Tompits")}
```

We will make dlvhex available both through source and binary packages. To ease becoming familiar with the system, we also offer a simple Web-interface available at

`http://www.kr.tuwien.ac.at/research/dlvhex`.

It allows for entering a HEX-program and filter predicates and displays the resultant models. On the same Web-page, we also supply a toolkit for developing custom plugins, embedded in the GNU autotools environment, which takes care for the low-level, system-specific build process and lets the plugin author concentrate his or her efforts on the implementation of the plugin's actual core functionality.

References

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 2001. A Framework for Ontology Integration. In *Proceedings of the First Semantic Web Working Symposium*, 303–316.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2004. Nonmonotonic Description Logic Programs: Implementation and Experiments. In *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004*, 511–527.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2005. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*. Morgan Kaufmann.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective Integration of Declarative Rules with external Evaluations for Semantic Web Reasoning. In *European Semantic Web Conference 2006, Proceedings*. To appear.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

Haarslev, V., and Möller, R. 2001. RACER System Description. In *Proceedings IJCAR-2001*, volume 2083 of *LNCS*, 701–705.

Ianni, G.; Ielpa, G.; Pietramala, A.; Santoro, M. C.; and Calimeri, F. 2004. Enhancing Answer Set Programming with Templates. In Delgrande, J. P., and Schaub, T., eds., *Proceedings NMR*, 233–239.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical Foundations of Object-Oriented and Frame-Based Languages. *J. ACM* 42(4):741–843.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2005. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*. To appear.

Lifschitz, V., and Turner, H. 1994. Splitting a Logic Program. In *Proceedings ICLP-94*, 23–38. Santa Margherita Ligure, Italy: MIT-Press.

Przymusinski, T. 1988. On the declarative semantics of deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 193–216.

Ross, K. A. 1994. Modular stratification and magic sets for datalog programs with negation. *J. ACM* 41(6):1216–1266.

Sintek, M., and Decker, S. 2002. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *International Semantic Web Conference*, 364–378.

Wang, K.; Antoniou, G.; Topor, R. W.; and Sattar, A. 2005. Merging and Aligning Ontologies in dl-Programs. In Adi, A.; Stoutenburg, S.; and Tabet, S., eds., *Proceedings First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005), Galway, Ireland, November 10-12, 2005*, volume 3791 of *LNCS*, 160–171. Springer.

1.5 Tableaux Calculi for Answer Set Programming

Tableau Calculi for Answer Set Programming

Martin Gebser and **Torsten Schaub**
Institut für Informatik
Universität Potsdam
Postfach 900327
D–14439 Potsdam

Abstract

We introduce a formal proof system based on tableau methods for analyzing computations made in Answer Set Programming (ASP). Our approach furnishes declarative and fine-grained instruments for characterizing operations as well as strategies of ASP-solvers. First, the granulation is detailed enough to capture the variety of propagation and choice operations of algorithms used for ASP; this also includes SAT-based approaches. Second, it is general enough to encompass the various strategies pursued by existing ASP-solvers. This provides us with a uniform framework for identifying and comparing fundamental properties of algorithms. Third, the approach allows us to investigate the proof complexity of algorithms for ASP, depending on choice operations. We show that exponentially different best-case computations can be obtained for different ASP-solvers. Finally, our approach is flexible enough to integrate new inference patterns, so to study their relation to existing ones. As a result, we obtain a novel approach to unfounded set handling based on loops, being applicable to non-SAT-based solvers. Furthermore, we identify backward propagation operations for unfounded sets.

Introduction

Answer Set Programming (ASP; (Baral 2003)) is an appealing tool for knowledge representation and reasoning. Its attractiveness is supported by the availability of efficient off-the-shelf ASP-solvers that allow for computing answer sets of logic programs. However, in contrast to the related area of satisfiability checking (SAT), ASP lacks a formal framework for describing inferences conducted by ASP-solvers, such as the resolution proof theory in SAT-solving (Mitchell 2005). This deficiency led to a great heterogeneity in the description of algorithms for ASP, ranging over procedural (Lin & Zhao 2004; Giunchiglia *et al.* 2004), fixpoint (Simons *et al.* 2002), and operational (Faber 2002; Anger *et al.* 2005) characterizations. On the one hand, this complicates identifying fundamental properties of algorithms, such as soundness and completeness. On the other hand, it almost disables formal comparisons among them.

We address this deficiency by introducing a family of tableau calculi (D'Agostino *et al.* 1999) for ASP. This allows us to view answer set computations as derivations in an inference system: A branch in a tableau corresponds to a successful or unsuccessful computation of an answer set; an entire tableau represents a traversal of the search space. Our approach furnishes declarative and fine-grained instruments for characterizing operations as well as strategies of ASP-solvers. In fact, we relate the approaches of *assat*, *cmodels*, *dlv*, *nomore*$^{++}$, *smodels*, etc. (Lin & Zhao 2004; Giunchiglia *et al.* 2004; Leone *et al.* 2006; Anger *et al.* 2005; Simons *et al.* 2002) to appropriate tableau calculi, in the sense that computations of an aforementioned solver comply with tableau proofs in a corresponding calculus. This provides us with a uniform proof-theoretic framework for analyzing and comparing different algorithms, which is the first of its kind for ASP.

Based on proof-theoretic concepts, we are able to derive general results, which apply to whole classes of algorithms instead of only specific ASP-solvers. In particular, we investigate the proof complexity of different approaches, depending on choice operations. It turns out that, regarding time complexity, exponentially different best-case computations can be obtained for different ASP-solvers. Furthermore, our proof-theoretic framework allows us to describe and study novel inference patterns, going beyond implemented systems. As a result, we obtain a loop-based approach to unfounded set handling, which is not restricted to SAT-based solvers. Also we identify backward propagation operations for unfounded sets.

Our work is motivated by the desire to converge the various heterogeneous characterizations of current ASP-solvers, on the basis of a canonical specification of principles underlying the respective algorithms. The classic example for this is DPLL (Davis & Putnam 1960; Davis *et al.* 1962), the most widely used algorithm for SAT, which is based on resolution proof theory (Mitchell 2005). By developing proof-theoretic foundations for ASP and abstracting from implementation details, we want to enhance the understanding of solving approaches as such. The proof-theoretic perspective also allows us to state results in a general way, rather than in a solver-specific one, and to study inferences by their admissibility, rather than from an implementation point of view.

Our work is inspired by the one of Jarvisalo, Junttila, and Niemelä, who use tableau methods in (Järvisalo *et al.* 2005; Junttila & Niemelä 2000) for investigating Boolean circuit satisfiability checking in the context of symbolic model checking. Although their target is different from ours, both approaches have many aspects in common. First, both use tableau methods for characterizing DPLL-type techniques.

Second, using cut rules for characterizing DPLL-type split operations is the key idea for analyzing the proof complexity of different inference strategies. General investigations in propositional proof complexity, in particular, the one of satisfiability checking (SAT), can be found in (Beame & Pitassi 1998). From the perspective of tableau systems, DPLL is very similar to the propositional version of the KE tableau calculus; both are closely related to weak connection tableau with atomic cut (as pointed out in (Hähnle 2001)). Tableau-based characterizations of logic programming are elaborated upon in (Fitting 1994). Pearce, Guzmán, and Valverde provide in (Pearce *et al.* 2000) a tableau calculus for automated theorem proving in equilibrium logic based on its 5-valued semantics. Other tableau approaches to nonmonotonic logics are summarized in (Olivetti 1999). Bonatti describes in (Bonatti 2001) a resolution method for skeptical answer set programming. Operator-based characterizations of propagation and choice operations in ASP can be found in (Faber 2002; Anger *et al.* 2005; Calimeri *et al.* 2001).

Answer Set Programming

Given an alphabet $\mathcal{P}$, a (normal) *logic program* is a finite set of rules of the form $p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n$, where $n \geq m \geq 0$ and each $p_i \in \mathcal{P}$ $(0 \leq i \leq n)$ is an *atom*. A *literal* is an atom p or its negation $not\ p$. For a rule r, let $head(r) = p_0$ be the *head* of r and $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ be the *body* of r; and let $body^+(r) = \{p_1, \ldots, p_m\}$ and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$. The set of atoms occurring in a program Π is given by $atom(\Pi)$. The set of bodies in Π is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping rule bodies with the same head p, let $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$. A program Π is *positive* if $body^-(r) = \emptyset$ for all $r \in \Pi$. $Cn(\Pi)$ denotes the smallest set of atoms closed under positive program Π. The *reduct*, Π^X, of Π relative to a set X of atoms is defined by $\Pi^X = \{head(r) \leftarrow body^+(r) \mid r \in \Pi,\ body^-(r) \cap X = \emptyset\}$. A set X of atoms is an *answer set* of a logic program Π if $Cn(\Pi^X) = X$. As an example, consider Program $\Pi_1 = \{a \leftarrow;\ c \leftarrow not\ b, not\ d;\ d \leftarrow a, not\ c\}$ and its two answer sets $\{a, c\}$ and $\{a, d\}$.

An *assignment* A is a partial mapping of objects in a program Π into $\{\boldsymbol{T}, \boldsymbol{F}\}$, indicating whether a member of the *domain* of A, $dom(A)$, is true or false, respectively. In order to capture the whole spectrum of ASP-solving techniques, we fix $dom(A)$ to $atom(\Pi) \cup body(\Pi)$ in the sequel. We define $A^{\boldsymbol{T}} = \{v \in dom(A) \mid A(v) = \boldsymbol{T}\}$ and $A^{\boldsymbol{F}} = \{v \in dom(A) \mid A(v) = \boldsymbol{F}\}$. We also denote an assignment A by a set of signed objects: $\{\boldsymbol{T}v \mid v \in A^{\boldsymbol{T}}\} \cup \{\boldsymbol{F}v \mid v \in A^{\boldsymbol{F}}\}$. For instance with Π_1, the assignment mapping body $\emptyset$ of rule $a\leftarrow$ to $\boldsymbol{T}$ and atom b to $\boldsymbol{F}$ is represented by $\{\boldsymbol{T}\emptyset, \boldsymbol{F}b\}$; all other atoms and bodies of Π_1 remain undefined. Following up this notation, we call an assignment *empty* if it leaves all objects undefined.

We define a set U of atoms as an *unfounded set* (van Gelder, Ross, & Schlipf 1991) of a program Π wrt a partial assignment A, if, for every rule $r \in \Pi$ such that $head(r) \in U$, either $(body^+(r) \cap A^{\boldsymbol{F}}) \cup (body^-(r) \cap A^{\boldsymbol{T}}) \neq \emptyset$ or $body^+(r) \cap U \neq \emptyset$. The *greatest unfounded set* of Π wrt A, denoted $GUS(\Pi, A)$, is the union of all unfounded sets of Π wrt A. *Loops* are sets of atoms that circularly depend upon one another in a program's positive atom dependency graph (Lin & Zhao 2004). In analogy to external support (Lee 2005) of loops, we define the *external bodies* of a loop L in Π as $EB(L) = \{body(r) \mid r \in \Pi, head(r) \in L, body^+(r) \cap L = \emptyset\}$. We denote the set of all loops in Π by $loop(\Pi)$.

Tableau calculi

We describe calculi for the construction of answer sets from logic programs. Such constructions are associated with binary trees called *tableaux* (D'Agostino *et al.* 1999). The nodes of the trees are (mainly) *signed propositions*, that is, propositions preceded by either $\boldsymbol{T}$ or $\boldsymbol{F}$, indicating an assumed truth value for the proposition. A *tableau* for a logic program Π and an initial assignment A is a binary tree such that the root node of the tree consists of the rules in Π and all members of A. The other nodes in the tree are *entries* of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, where $v \in dom(A)$, generated by extending a tableau using the rules in Figure 1 in the following standard way (D'Agostino *et al.* 1999): Given a tableau rule and a branch in the tableau such that the prerequisites of the rule hold in the branch, the tableau can be extended by adding new entries to the end of the branch as specified by the rule. If the rule is the Cut rule in *(m)*, then entries $\boldsymbol{T}v$ and $\boldsymbol{F}v$ are added as the left and the right child to the end of the branch. For the other rules, the consequent of the rule is added to the end of the branch. For convenience, the application of tableau rules makes use of two conjugation functions, $\boldsymbol{t}$ and $\boldsymbol{f}$. For a literal l, define:

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \in \mathcal{P} \\ \boldsymbol{F}p & \text{if } l = not\ p \text{ for a } p \in \mathcal{P} \end{cases}$$

$$\boldsymbol{f}l = \begin{cases} \boldsymbol{T}p & \text{if } l = not\ p \text{ for a } p \in \mathcal{P} \\ \boldsymbol{F}l & \text{if } l \in \mathcal{P} \end{cases}$$

Some rule applications are subject to provisos. $(\S)$ stipulates that $B_1, \ldots, B_m$ constitute all bodies of rules with head p. $(\dagger)$ requires that p belongs to the greatest unfounded set induced by all rules whose body is not among $B_1, \ldots, B_m$. $(\ddagger)$ makes sure that p belongs to a loop whose external bodies are $B_1, \ldots, B_m$. Finally, $(\sharp[X])$ guides the application of the Cut rule by restricting cut objects to members of X.[1] Different tableau calculi are obtained from different rule sets. When needed this is made precise by enumerating the tableau rules. The following tableau calculi are of particular interest:

$$\mathcal{T}_{comp} = \{\textit{(a)-(h)}, \textit{Cut}[atom(\Pi) \cup body(\Pi)]\} \quad (1)$$

$$\mathcal{T}_{smodels} = \{\textit{(a)-(i)}, \textit{Cut}[atom(\Pi)]\} \quad (2)$$

$$\mathcal{T}_{noMoRe} = \{\textit{(a)-(i)}, \textit{Cut}[body(\Pi)]\} \quad (3)$$

$$\mathcal{T}_{nomore^{++}} = \{\textit{(a)-(i)}, \textit{Cut}[atom(\Pi) \cup body(\Pi)]\} \quad (4)$$

[1] The Cut rule (*(m)* in Figure 1) may, in principle, introduce more general entries; this would however necessitate additional decomposition rules, leading to extended tableau calculi.

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

(a) Forward True Body (FTB)

$$\frac{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\} \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

(b) Backward False Body (BFB)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{T}\{l_1, \ldots, l_n\}}{\boldsymbol{T}p}$$

(c) Forward True Atom (FTA)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{F}p}{\boldsymbol{F}\{l_1, \ldots, l_n\}}$$

(d) Backward False Atom (BFA)

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \boldsymbol{f}l_i}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

(e) Forward False Body (FFB)

$$\frac{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\boldsymbol{t}l_i}$$

(f) Backward True Body (BTB)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\ (\S)$$

(g) Forward False Atom (FFA)

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\S)$$

(h) Backward True Atom (BTA)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\ (\dagger)$$

(i) Well-Founded Negation (WFN)

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\dagger)$$

(j) Well-Founded Justification (WFJ)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\ (\ddagger)$$

(k) Forward Loop (FL)

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\ddagger)$$

(l) Backward Loop (BL)

$$\frac{}{\boldsymbol{T}v \mid \boldsymbol{F}v}\ (\sharp[X])$$

(m) Cut (Cut$[X]$)

$$\begin{array}{rcl}
(\S) & : & \mathit{body}(p) = \{B_1, \ldots, B_m\} \\
(\dagger) & : & \{B_1, \ldots, B_m\} \subseteq \mathit{body}(\Pi),\ p \in \mathit{GUS}(\{r \in \Pi \mid \mathit{body}(r) \notin \{B_1, \ldots, B_m\}\}, \emptyset) \\
(\ddagger) & : & p \in L, L \in \mathit{loop}(\Pi),\ \mathit{EB}(L) = \{B_1, \ldots, B_m\} \\
(\sharp[X]) & : & v \in X
\end{array}$$

Figure 1: Tableau rules for answer set programming.

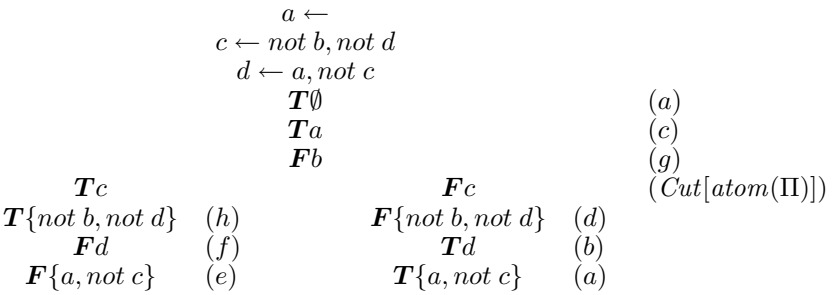


Figure 2: Tableau of $\mathcal{T}_{smodels}$ for Π_1 and the empty assignment.

An exemplary tableau of $\mathcal{T}_{smodels}$ is given in Figure 2, where rule applications are indicated by either letters or rule names, like (a) or $(Cut[atom(\Pi)])$. Both branches comprise Π_1 along with a total assignment for $atom(\Pi_1) \cup body(\Pi_1)$; the left one represents answer set $\{a, c\}$, the right one gives answer set $\{a, d\}$.

A branch in a tableau is *contradictory*, if it contains both entries $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some $v \in dom(A)$. A branch is *complete*, if it is contradictory, or if the branch contains either the entry $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in dom(A)$ and is closed under all rules in a given calculus, except for the Cut rule in *(m)*. For instance, both branches in Figure 2 are non-contradictory and complete.

For each $v \in dom(A)$, we say that entry $\boldsymbol{T}v$ (or $\boldsymbol{F}v$) can be deduced by a set $\mathcal{R}$ of tableau rules in a branch, if the entry $\boldsymbol{T}v$ (or $\boldsymbol{F}v$) can be generated from nodes in the branch by applying rules in $\mathcal{R}$ only. Note that every branch corresponds to a pair (Π, A) consisting of a program Π and an assignment A, and vice versa;[2] we draw on this relationship for identifying branches in the sequel. Accordingly, we let $D_{\mathcal{R}}(\Pi, A)$ denote the set of all entries deducible by rule set $\mathcal{R}$ in branch (Π, A). Moreover, $D^*_{\mathcal{R}}(\Pi, A)$ represents the set of all entries in the smallest branch extending (Π, A) and being closed under $\mathcal{R}$. When dealing with tableau calculi, like $\mathcal{T}$, we slightly abuse notation and write $D_{\mathcal{T}}(\Pi, A)$ (or $D^*_{\mathcal{T}}(\Pi, A)$) instead of $D_{\mathcal{T}\setminus\{(m)\}}(\Pi, A)$ (or $D^*_{\mathcal{T}\setminus\{(m)\}}(\Pi, A)$), thus ignoring *Cut*. We mention that $D^*_{\{(a),(c),(e),(g)\}}(\Pi, A)$ corresponds to Fitting's operator (Fitting 2002). Similarly, we detail in the subsequent sections that $D^*_{\{(a)\text{-}(h)\}}(\Pi, A)$ coincides with unit propagation on a program's completion (Clark 1978; Apt *et al.* 1987), $D^*_{\{(a),(c),(e),(g),(i)\}}(\Pi, A)$ amounts to propagation via well-founded semantics (van Gelder, Ross, & Schlipf 1991), and $D^*_{\{(a)\text{-}(i)\}}(\Pi, A)$ captures *smodels*' propagation (Simons *et al.* 2002), that is, well-founded semantics enhanced by backward propagation. Note that all deterministic rules in Figure 1 are answer set preserving; this also applies to the Cut rule when considering both resulting branches.

A tableau is *complete* if all its branches are complete. A complete tableau for a program and the empty assignment such that all branches are contradictory is called a *refutation* for the program; it means that the program has no answer set, as exemplarily shown next for *smodels*-type tableaux.

Theorem 1 *Let Π be a logic program and let $\emptyset$ denote the empty assignment. Then, the following holds for tableau calculus $\mathcal{T}_{smodels}$:*

1. Π has no answer set iff any complete tableau for Π and $\emptyset$ is a refutation.

2. If Π has an answer set X, then every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch (Π, A) such that $X = A^{\boldsymbol{T}} \cap atom(\Pi)$.

3. If a tableau for Π and $\emptyset$ has a non-contradictory complete branch (Π, A), then $A^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π.

The same results are obtained for other tableau calculi, like $\mathcal{T}_{noMoRe}$ and $\mathcal{T}_{nomore^{++}}$, all of which are sound and complete for ASP.

Characterizing existing ASP-solvers

In this section, we discuss the relation between the tableau rules in Figure 1 and well-known ASP-solvers. As it turns out, our tableau rules are well-suited for describing the approaches of a wide variety of ASP-solvers. In particular, we cover all leading approaches to answer set computation for (normal) logic programs. We start with SAT-based solvers *assat* and *cmodels*, then go on with atom-based solvers *smodels* and *dlv*, and finally turn to *hybrid* solvers, like *nomore*$^{++}$, working on atoms as well as bodies.

SAT-based solvers. The basic idea of SAT-based solvers is to use some SAT-solver as model generator and to afterwards check whether a generated model contains an unfounded *loop*. Lin and Zhao show in (Lin & Zhao 2004) that the answer sets of a logic program Π coincide with the models of the *completion* of Π and the set of all *loop formulas* of Π. The respective propositional logic translation is $Comp(\Pi) \cup LF(\Pi)$, where:[3]

$$\begin{aligned} Comp(\Pi) &= \{p \equiv (\textstyle\bigvee_{k=1..m} \bigwedge_{l \in B_k} l) \mid \\ &\qquad p \in atom(\Pi), body(p) = \{B_1, \ldots, B_m\}\} \\ LF(\Pi) &= \{\neg(\textstyle\bigvee_{k=1..m} \bigwedge_{l \in B_k} l) \rightarrow \bigwedge_{p \in L} \neg p \mid \\ &\qquad L \in loop(\Pi), EB(L) = \{B_1, \ldots, B_m\}\} \end{aligned}$$

[2] Given a branch (Π, A) in a tableau for Π and initial assignment A_0, we have $A_0 \subseteq A$.

[3] Note that a negative default literal $not\ p$ is translated as $\neg p$.

This translation constitutes the backbone of SAT-based solvers *assat* (Lin & Zhao 2004) and *cmodels* (Giunchiglia *et al.* 2004). However, loop formulas $LF(\Pi)$ require exponential space in the worst case (Lifschitz & Razborov 2006). Thus, *assat* adds loop formulas from $LF(\Pi)$ incrementally to $Comp(\Pi)$, whenever some model of $Comp(\Pi)$ not corresponding to an answer set has been generated by the underlying SAT-solver.[4] The approach of *cmodels* avoids storing loop formulas by exploiting the SAT-solver's inner backtracking and learning scheme. Despite the differences between *assat* and *cmodels*, we can uniformly characterize their model generation and verification steps. We first describe tableaux capturing the proceeding of the underlying SAT-solver and then go on with unfounded set checks.

In analogy to Theorem 1, models of $Comp(\Pi)$ correspond to tableaux of $\mathcal{T}_{comp}$.

Theorem 2 *Let Π be a logic program. Then, M is a model of $Comp(\Pi)$ iff every complete tableau of $\mathcal{T}_{comp}$ for Π and $\emptyset$ has a unique non-contradictory branch (Π, A) such that $M = A^{\boldsymbol{T}} \cap atom(\Pi)$.*

Intuitively, tableau rules *(a)-(h)* describe unit propagation on a program's completion, represented in CNF as required by most SAT-solvers. Note that *assat* and *cmodels* introduce propositional variables for bodies in order to obtain a polynomially-sized set of clauses equivalent to a program's completion (Babovich & Lifschitz 2003). Due to the fact that atoms and bodies are represented as propositional variables, allowing both of them as branching variables in $\mathcal{T}_{comp}$ (via $Cut[atom(\Pi) \cup body(\Pi)]$; cf. (1)) makes sense.

Once a model of $Comp(\Pi)$ has been generated by the underlying SAT-solver, *assat* and *cmodels* apply an unfounded set check for deciding whether the model is an answer set. If it fails, unfounded loops whose atoms are true (so-called *terminating loops* (Lin & Zhao 2004)) are determined. Their loop formulas are used to eliminate the generated model. Unfounded set checks, as performed by *assat* and *cmodels*, can be captured by tableau rules *FFB* and *FL* (*(e)* and *(k)* in Figure 1) as follows.

Theorem 3 *Let Π be a logic program, let M be a model of $Comp(\Pi)$, and let $A = \{\boldsymbol{T}p \mid p \in M\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus M\}$. Then, M is an answer set of Π iff $M \cap (D_{\{FL\}}(\Pi, D_{\{FFB\}}(\Pi, A)))^{\boldsymbol{F}} = \emptyset$.*

With SAT-based approaches, sophisticated unfounded set checks, able to detect unfounded loops, are applied only to non-contradictory complete branches. Unfortunately, programs may yield exponentially many loops (Lifschitz & Razborov 2006). This can lead to exponentially many models of a program's completion that turn out to be no answer sets (Giunchiglia & Maratea 2005). In view of Theorem 3, it means that exponentially many branches may have to be completed by final unfounded set checks.

Atom-based solvers. We now describe the relation between *smodels* (Simons *et al.* 2002) and *dlv* (Leone *et al.* 2006) on the one side and our tableau rules on the other side. We first concentrate on characterizing *smodels* and then sketch how our characterization applies to *dlv*.

Given that only literals are explicitly represented in *smodels*' assignments, whereas truth and falsity of bodies are determined implicitly, one might consider rewriting tableau rules to work on literals only, thereby, restricting the domain of assignments to atoms. For instance, tableau rule *FFA* (*(g)* in Figure 1) would then turn into:

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p} \quad \begin{array}{l}(\{r \in \Pi \mid head(r) = p,\\ \qquad body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} = \emptyset)\end{array}$$

Observe that, in such a reformulation, one again refers to bodies by determining their values in the proviso associated with the inference rule. Reformulating tableau rules to work on literals only thus complicates provisos and does not substantially facilitate the description.[5] In (Giunchiglia & Maratea 2005), additional variables for bodies, one for each rule of a program, are even explicitly introduced for comparing *smodels* with DPLL. Given that propagation, even within atom-based solvers, has to consider the truth status of rules' bodies, the only saving in the computation of answer sets is limiting branching to atoms, which is expressed by $Cut[atom(\Pi)]$ in $\mathcal{T}_{smodels}$ (cf. (2)).

Propagation in *smodels* is accomplished by two functions, called $atleast$ and $atmost$ (Simons *et al.* 2002).[6] The former computes deterministic consequences by applying completion-based forward and backward propagation (*(a)-(h)* in Figure 1); the latter falsifies greatest unfounded sets (WFN; *(i)* in Figure 1).

The following result captures propagation via $atleast$ in terms of $\mathcal{T}_{comp}$.

Theorem 4 *Let Π be a logic program and let A be an assignment such that $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} \subseteq atom(\Pi)$. Let $A_S = atleast(\Pi, A)$ and $A_T = D^*_{\mathcal{T}_{comp}}(\Pi, A)$. If $A_S^{\boldsymbol{T}} \cap A_S^{\boldsymbol{F}} \neq \emptyset$, then $A_T^{\boldsymbol{T}} \cap A_T^{\boldsymbol{F}} \neq \emptyset$; otherwise, we have $A_S \subseteq A_T$.*

This result shows that anything derived by $atleast$ can also be derived by $\mathcal{T}_{comp}$ (without Cut). In fact, if $atleast$ detects an inconsistency ($A_S^{\boldsymbol{T}} \cap A_S^{\boldsymbol{F}} \neq \emptyset$), then $\mathcal{T}_{comp}$ can derive it as well ($A_T^{\boldsymbol{T}} \cap A_T^{\boldsymbol{F}} \neq \emptyset$). Otherwise, $\mathcal{T}_{comp}$ can derive at least as much as $atleast$ ($A_S \subseteq A_T$). This subsumption does not only originate from the (different) domains of assignments, that is, only atoms for $atleast$ but also bodies for $\mathcal{T}_{comp}$. Rather, it is the redundant representation of rules' bodies within *smodels* that inhibits possible derivations obtained with $\mathcal{T}_{comp}$. To see this, consider rules $a \leftarrow c, d$ and $b \leftarrow c, d$ and an assignment A that contains $\boldsymbol{F}a$ but leaves atoms c and d undefined. For such an A, $atleast$ can only determine that rule $a \leftarrow c, d$ must not be applied, but it does not recognize that rule $b \leftarrow c, d$, sharing body $\{c, d\}$, is inapplicable as well. If $b \leftarrow c, d$ is the only rule with head atom b in the underlying program, then $\mathcal{T}_{comp}$ can, in contrast to $atleast$, derive $\boldsymbol{F}b$ via FFA (*(g)* in Figure 1). A one-to-one correspondence between $atleast$ and $\mathcal{T}_{comp}$ on

[4] Note that every answer set of Π is a model of $Comp(\Pi)$, but not vice versa (Fages 1994).

[5] Restricting the domain of assignments to atoms would also disable the analysis of different Cut variants done below.

[6] Here, $atleast$ and $atmost$ are taken as defined on signed propositions instead of literals (Simons *et al.* 2002).

derived atoms could be obtained by distinguishing different occurrences of the same body. However, for each derivation of $atleast$, there is a corresponding one in $\mathcal{T}_{comp}$. That is, every propagation done by $atleast$ can be described with $\mathcal{T}_{comp}$.

Function $atmost$ returns the maximal set of potentially true atoms, that is, $atom(\Pi) \setminus (GUS(\Pi, A) \cup A^{\boldsymbol{F}})$ for a program Π and an assignment A. Atoms in the complement of $atmost$, that is, the greatest unfounded set $GUS(\Pi, A)$ augmented with $A^{\boldsymbol{F}}$, must be false. This can be described by tableau rules FFB and WFN (*(e)* and *(i)* in Figure 1).

Theorem 5 *Let Π be a logic program and let A be an assignment such that $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} \subseteq atom(\Pi)$. We have $atom(\Pi) \setminus atmost(\Pi, A) = (D_{\{WFN\}}(\Pi, D_{\{FFB\}}(\Pi, A)))^{\boldsymbol{F}} \cup A^{\boldsymbol{F}}$.*

Note that *smodels* adds literals $\{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus atmost(\Pi, A)\}$ to an assignment A. If this leads to an inconsistency, so does $D_{\{WFN\}}(\Pi, D_{\{FFB\}}(\Pi, A))$.

We have seen that *smodels*' propagation functions, $atleast$ and $atmost$, can be described by tableau rules *(a)-(i)*. By adding $Cut[atom(\Pi)]$, we thus get tableau calculus $\mathcal{T}_{smodels}$ (cf. (2)). Note that *lookahead* (Simons *et al.* 2002) can also be described by means of $Cut[atom(\Pi)]$: If *smodels*' lookahead derives some literal $\boldsymbol{t}l$, a respective branch can be extended by Cut applied to the atom involved in l. The subbranch containing $\boldsymbol{f}l$ becomes contradictory by closing it under $\mathcal{T}_{smodels}$. Also, if *smodels*' propagation detects an inconsistency on $\boldsymbol{t}l$, then both subbranches created by Cut, $\boldsymbol{f}l$ and $\boldsymbol{t}l$, become contradictory by closing them; the subtableau under consideration becomes complete.

After having discussed *smodels*, we briefly turn to *dlv*: In contrast to *smodels*' $atmost$, greatest unfounded set detection is restricted to strongly connected components of programs' atom dependency graphs (Calimeri *et al.* 2001). Hence, tableau rule WFN has to be adjusted to work on such components.[7] In the other aspects, propagation within *dlv* (Faber 2002) is (on normal logic programs) similar to *smodels*' $atleast$. Thus, tableau calculus $\mathcal{T}_{smodels}$ also characterizes *dlv* very closely.

Hybrid solvers. Finally, we discuss similarities and differences between atom-based ASP-solvers, *smodels* and *dlv*, and *hybrid* solvers, working on bodies in addition to atoms. Let us first mention that SAT-based solvers, *assat* and *cmodels*, are in a sense hybrid, since the CNF representation of a program's completion contains variables for bodies. Thus, underlying SAT-solvers can branch on both atoms and bodies (via $Cut[atom(\Pi) \cup body(\Pi)]$ in $\mathcal{T}_{comp}$). The only genuine ASP-solver (we know of) explicitly assigning truth values to bodies, in addition to atoms, is *nomore*$^{++}$ (Anger *et al.* 2005).[8]

In (Anger *et al.* 2005), propagation rules applied by *nomore*$^{++}$ are described in terms of operators: $\mathcal{P}$ for forward propagation, $\mathcal{B}$ for backward propagation, $\mathcal{U}$ for falsifying greatest unfounded sets, and $\mathcal{L}$ for lookahead. Similar to our tableau rules, these operators apply to both atoms and bodies. We can thus show direct correspondences between tableau rules *(a), (c), (e), (g)* and $\mathcal{P}$, *(b), (d), (f), (h)* and $\mathcal{B}$, and *(i)* and $\mathcal{U}$. Similar to *smodels*' lookahead, derivations of $\mathcal{L}$ can be described by means of $Cut[atom(\Pi) \cup body(\Pi)]$. So by replacing $Cut[atom(\Pi)]$ with $Cut[atom(\Pi) \cup body(\Pi)]$, we obtain tableau calculus $\mathcal{T}_{nomore^{++}}$ (cf. (4)) from $\mathcal{T}_{smodels}$. In the next section, we show that this subtle difference, also observed on SAT-based solvers, may have a great impact on proof complexity.

Proof complexity

We have seen that genuine ASP-solvers largely coincide on their propagation rules and differ primarily in the usage of Cut. In this section, we analyze the relative efficiency of tableau calculi with different Cut rules. Thereby, we take $\mathcal{T}_{smodels}$, $\mathcal{T}_{noMoRe}$, and $\mathcal{T}_{nomore^{++}}$ into account, all using tableau rules *(a)-(i)* in Figure 1 but applying the Cut rule either to $atom(\Pi)$, $body(\Pi)$, or both of them (cf. (2–4)). These three calculi are of particular interest: On the one hand, they can be used to describe the strategies of ASP-solvers, as shown in the previous section; on the other hand, they also represent different paradigms, either atom-based, rule-based, or hybrid. So by considering these particular calculi, we obtain results that, on the one hand, are of practical relevance and that, on the other hand, apply to different approaches in general.

For comparing different tableau calculi, we use well-known concepts from *proof complexity* (Beame & Pitassi 1998; Järvisalo *et al.* 2005). Accordingly, we measure the complexity of unsatisfiable logic programs, that is, programs without answer sets, in terms of *minimal* refutations. The size of a tableau is determined in the standard way as the number of nodes in it. A tableau calculus $\mathcal{T}$ is not *polynomially simulated* (Beame & Pitassi 1998; Järvisalo *et al.* 2005) by another tableau calculus $\mathcal{T}'$ if there is an infinite (witnessing) family $\{\Pi^n\}$ of unsatisfiable logic programs such that minimal refutations of $\mathcal{T}'$ for Π are asymptotically exponential in the size of minimal refutations of $\mathcal{T}$ for Π. A tableau calculus $\mathcal{T}$ is *exponentially stronger* than a tableau calculus $\mathcal{T}'$ if $\mathcal{T}$ polynomially simulates $\mathcal{T}'$, but not vice versa. Two tableau calculi are *efficiency-incomparable* if neither one polynomially simulates the other. Note that proof complexity says nothing about how difficult it is to find a minimal refutation. Rather, it provides a lower bound on the run-time of proof-finding algorithms (in our context, ASP-solvers), independent from heuristic influences.

In what follows, we provide families of unsatisfiable logic programs witnessing that neither $\mathcal{T}_{smodels}$ polynomially simulates $\mathcal{T}_{noMoRe}$ nor vice versa. This means that, on certain instances, restricting the Cut rule to either only atoms or bodies leads to exponentially longer minimal run-times of either atom- or rule-based solvers in comparison to their counterparts, no matter which heuristic is applied.

Lemma 6 *There is an infinite family $\{\Pi^n\}$ of logic programs such that*

[7] However, iterated application of such a WFN variant leads to the same result as *(i)* in Figure 1.

[8] Complementing atom-based solvers, the *noMoRe* system (Konczak *et al.* 2006) is rule-based (cf. $\mathcal{T}_{noMoRe}$ in (3)).

1. the size of minimal refutations of $\mathcal{T}_{noMoRe}$ is linear in n and

2. the size of minimal refutations of $\mathcal{T}_{smodels}$ is exponential in n.

Lemma 7 *There is an infinite family $\{\Pi^n\}$ of logic programs such that*

1. the size of minimal refutations of $\mathcal{T}_{smodels}$ is linear in n and

2. the size of minimal refutations of $\mathcal{T}_{noMoRe}$ is exponential in n.

Family $\{\Pi^n_a \cup \Pi^n_c\}$ witnesses Lemma 6 and $\{\Pi^n_b \cup \Pi^n_c\}$ witnesses Lemma 7(see Figure 3).

The next result follows immediately from Lemma 6 and 7.

Theorem 8 *$\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are efficiency-incomparable.*

Given that any refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are as well refutations of $\mathcal{T}_{nomore^{++}}$, we have that $\mathcal{T}_{nomore^{++}}$ polynomially simulates both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$. So the following is an immediate consequence of Theorem 8.

Corollary 9 *$\mathcal{T}_{nomore^{++}}$ is exponentially stronger than both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$.*

The major implication of Corollary 9 is that, on certain logic programs, a priori restricting the Cut rule to either only atoms or bodies necessitates the traversal of an exponentially larger search space than with unrestricted Cut. Note that the phenomenon of exponentially worse proof complexity in comparison to $\mathcal{T}_{nomore^{++}}$ does not, depending on the program family, apply to one of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ alone. Rather, families $\{\Pi^n_a\}$, $\{\Pi^n_b\}$, and $\{\Pi^n_c\}$ can be combined such that both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are exponentially worse than $\mathcal{T}_{nomore^{++}}$. For certain logic programs, the unrestricted Cut rule is thus the only way to have at least the chance of finding a short refutation. Empirical evidence for the exponentially different behavior is given in (Anger *et al.* 2006b).

Finally, note that our proof complexity results are robust. That is, they apply to any possible ASP-solver whose proceeding can be described by corresponding tableaux. For instance, any computation of *smodels* can be associated with a tableau of $\mathcal{T}_{smodels}$. A computation of *smodels* thus requires time proportional to the size of the corresponding tableau; in particular, the magnitude of a minimal tableau constitutes a lower bound on the run-time of *smodels*. This correlation is independent from whether an assignment contains only atoms or also bodies of a program: The size of any branch (not containing duplicate entries) is tightly bound by the size of a logic program. Therefore, exponential growth of minimal refutations is, for polynomially growing program families as the ones in Figure 3, exclusively caused by the increase of necessary Cut applications, introducing an exponential number of branches.

Unfounded sets

We have analyzed propagation techniques and proof complexity of existing approaches to ASP-solving. We have seen that all approaches exploit propagation techniques amounting to inferences from program completion (*(a)-(h)* in Figure 1). In particular, SAT-based and genuine ASP-solvers differ only in the treatment of unfounded sets: While the former apply (loop-detecting) unfounded set checks to total assignments only, the latter incorporate (greatest) unfounded set falsification (WFN; *(i)* in Figure 1) into their propagation. However, tableau rule WFN, as it is currently applied by genuine ASP-solvers, has several peculiarities:

A. WFN is partly redundant, that is, it overlaps with completion-based tableau rule FFA (*(g)* in Figure 1), which falsifies atoms belonging to singleton unfounded sets.

B. WFN deals with greatest unfounded sets, which can be (too) exhaustive.

C. WFN is asymmetrically applied, that is, solvers apply no backward counterpart.

In what follows, we thus propose and discuss alternative approaches to unfounded set handling, motivated by SAT-based solvers and results in (Lin & Zhao 2004). Before we start, let us briefly introduce some vocabulary. Given two sets of tableau rules, $\mathcal{R}_1$ and $\mathcal{R}_2$, we say that $\mathcal{R}_1$ is *at least as effective* as $\mathcal{R}_2$ if, for any branch (Π, A), we have $D^*_{\mathcal{R}_2}(\Pi, A) \subseteq D^*_{\mathcal{R}_1}(\Pi, A)$. We say that $\mathcal{R}_1$ is *more effective* than $\mathcal{R}_2$ if $\mathcal{R}_1$ is at least as effective as $\mathcal{R}_2$, but not vice versa. If $\mathcal{R}_1$ is at least as effective as $\mathcal{R}_2$ and vice versa, then $\mathcal{R}_1$ and $\mathcal{R}_2$ are *equally effective*. Finally, $\mathcal{R}_1$ and $\mathcal{R}_2$ are *orthogonal* if they are not equally effective and neither one is more effective than the other. A correspondence between two rule sets $\mathcal{R}_1 \cup \mathcal{R}$ and $\mathcal{R}_2 \cup \mathcal{R}$ means that the correspondence between $\mathcal{R}_1$ and $\mathcal{R}_2$ holds when D^* takes auxiliary rules $\mathcal{R}$ into account as well.

We start with analyzing the relation between WFN and FFA, both falsifying unfounded atoms in forward direction. The role of FFB (*(e)* in Figure 1) is to falsify bodies that positively rely on falsified atoms. Intuitively, this allows to capture iterated applications of WFN and FFA, respectively, in which FFB behaves neutrally. Taking up item A. above, we have the following result.

Proposition 1 *Set of rules $\{WFN, FFB\}$ is more effective than $\{FFA, FFB\}$.*

This tells us that FFA is actually redundant in the presence of WFN. However, all genuine ASP-solvers apply FFA as a sort of "local" negation (e.g. $atleast$ of *smodels* and operator $\mathcal{P}$ of *nomore*$^{++}$) and separately WFN as "global" negation (e.g. $atmost$ of *smodels* and operator $\mathcal{U}$ of *nomore*$^{++}$). Certainly, applying FFA is reasonable as applicability is easy to determine. (Thus, SAT-based solvers apply FFA, but not WFN.) But with FFA at hand, Proposition 1 also tells us that greatest unfounded sets are too unfocused to describe the sort of unfounded sets that truly require a dedicated treatment: The respective tableau rule, WFN, subsumes a simpler one, FFA.

A characterization of WFN's effect, not built upon greatest unfounded sets, is obtained by putting results in (Lin & Zhao 2004) into the context of partial assignments.

Theorem 10 *Sets of rules $\{WFN, FFB\}$ and $\{FFA, FL, FFB\}$ are equally effective.*

$$\Pi_a^n = \left\{ \begin{array}{lcl} x & \leftarrow & not\ x \\ x & \leftarrow & a_1, b_1 \\ & \vdots & \\ x & \leftarrow & a_n, b_n \end{array} \right\} \qquad \Pi_b^n = \left\{ \begin{array}{lcllcl} x & \leftarrow & c_1, \ldots, c_n, not\ x & & & \\ c_1 & \leftarrow & a_1 & c_1 & \leftarrow & b_1 \\ & \vdots & & & \vdots & \\ c_n & \leftarrow & a_n & c_n & \leftarrow & b_n \end{array} \right\} \qquad \Pi_c^n = \left\{ \begin{array}{lcl} a_1 & \leftarrow & not\ b_1 \\ b_1 & \leftarrow & not\ a_1 \\ & \vdots & \\ a_n & \leftarrow & not\ b_n \\ b_n & \leftarrow & not\ a_n \end{array} \right\}$$

Figure 3: Families of programs $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$.

Hence, one may safely substitute WFN by FFA and FL (*(k)* in Figure 1), without forfeiting atoms that must be false due to the lack of (non-circular) support. Thereby, FFA concentrates on single atoms and FL on unfounded loops. Since both tableau rules have different scopes, they do not overlap but complement each other.

Proposition 2 *Sets of rules* $\{FFA, FFB\}$ *and* $\{FL, FFB\}$ *are orthogonal.*

SAT-based approaches provide an explanation why concentrating on cyclic structures, namely loops, besides single atoms is sufficient: When falsity of unfounded atoms does not follow from a program's completion or FFA, then there is a loop all of whose external bodies are false. Such a loop (called *terminating loop* in (Lin & Zhao 2004)) is a subset of the greatest unfounded set. So in view of item B. above, loop-oriented approaches allow for focusing unfounded set computations on the intrinsically necessary parts. In fact, the more sophisticated unfounded set techniques applied by genuine ASP-solvers aim at circular structures induced by loops. That is, both *smodels*' approach, based on "source pointers" (Simons 2000), as well as *dlv*'s approach, based on strongly connected components of programs' atom dependency graphs (Calimeri *et al.* 2001), can be seen as restrictions of WFN to structures induced by loops. However, neither of them takes loops as such into account.

Having considered forward propagation for unfounded sets, we come to backward propagation, that is, BTA, WFJ, and BL (*(h)*, *(j)*, and *(l)* in Figure 1). Although no genuine ASP-solver currently integrates propagation techniques corresponding to WFJ or BL, as mentioned in item C. above, both rules are answer set preserving.

Proposition 3 *Let* Π *be a logic program and let* A *be an assignment. Let* $B \in body(\Pi)$ *such that* $\boldsymbol{T}B \in D_{\{WFJ\}}(\Pi, A)$ *(or* $\boldsymbol{T}B \in D_{\{BL\}}(\Pi, A)$*, respectively). Then, branch* $(\Pi, A \cup D_{\{WFN\}}(\Pi, A \cup \{\boldsymbol{F}B\}))$ *(or* $(\Pi, A \cup D_{\{FL\}}(\Pi, A \cup \{\boldsymbol{F}B\}))$*, respectively) is contradictory.*

Both WFJ and BL ensure that falsifying some body does not lead to an inconsistency due to applying their forward counterparts. In fact, WFJ and BL are contrapositives of WFN and FL, respectively, in the same way as simpler rule BTA is for FFA.

A particularity of supporting true atoms by backward propagation is that "global" rule WFJ is more effective than "local" ones, BTA and BL. Even adding tableau rule BTB (*(f)* in Figure 1), for enabling iterated application of backward rules setting bodies to true, does not compensate for the global character of WFJ.

Proposition 4 *Set of rules* $\{WFJ, BTB\}$ *is more effective than* $\{BTA, BL, BTB\}$.

We conclude by discussing different approaches to unfounded set handling. Both SAT-based and genuine ASP-solvers apply tableau rules FFA and BTA, both focusing on single atoms. In addition, genuine ASP-solvers apply WFN to falsify more complex unfounded sets. However, WFN gives an overestimation of the parts of unfounded sets that need a dedicated treatment: SAT-based approaches show that concentrating on loops, via FL, is sufficient. However, the latter apply loop-detecting unfounded set checks only to total assignments or use loop formulas recorded in reaction to previously failed unfounded set checks. Such a recorded loop formula is then exploited by propagation within SAT-based solvers in both forward and backward direction, which amounts to applying FL and BL. A similar kind of backward propagation, by either WFJ or BL, is not exploited by genuine ASP-solvers, so unfounded set treatment is asymmetric. We however believe that bridging the gap between SAT-based and genuine ASP-solvers is possible by putting the concept of loops into the context of partial assignments. For instance, a loop-oriented unfounded set algorithm is described in (Anger *et al.* 2006a).

Discussion

In contrast to the area of SAT, where the proof-theoretic foundations of SAT-solvers are well-understood (Mitchell 2005; Beame & Pitassi 1998), the literature on ASP-solvers is generally too specific in terms of algorithms or solvers; existing characterizations are rather heterogeneous and often lack declarativeness. We address this deficiency by proposing a tableau proof system that provides a formal framework for analyzing computations of ASP-solvers. To our knowledge, this approach is the first uniform proof-theoretic account for computational techniques in ASP. Our tableau framework allows to abstract away implementation details and to identify valid inferences; hence, soundness and completeness results are easily obtained. This is accomplished by associating specific tableau calculi with the approaches of ASP-solvers, rather than with their solving algorithms.

The explicit integration of bodies into assignments has several benefits. First, it allows us to capture completion-based and hybrid approaches in a closer fashion. Second, it allows us to reveal exponentially different proof complexities of ASP-solvers. Finally, even inferences in atom-based systems, like *smodels* and *dlv*, are twofold insofar as they must take program rules into account for propagation. This feature is simulated in our framework through the corresponding bodies. Although this simulation is sufficient for

establishing formal results, it is worth noting that dealing with rules bears more redundancy than dealing with their bodies. Related to this, we have seen that rule-wise consideration of bodies, as for instance done in *smodels*' $atleast$, can forfeit derivations that are easily obtained based on non-duplicated bodies (cf. Theorem 4). The tableau rules underlying atom-based and hybrid systems also reveal that the only major difference lies in the selection of program objects to branch upon.

The branching rule, Cut, has a major influence on proof complexity. It is well-known that an uncontrolled application of Cut is prone to inefficiency. The restriction of applying Cut to (sub)formulae occurring in the input showed to be an effective way to "tame" the cut (D'Agostino *et al.* 1999). We followed this by investigating Cut applications to atoms and bodies occurring in a program. Our proof complexity results tell us that the minimal number of required Cut applications may vary exponentially when restricting Cut to either only atoms or bodies. For not a priori degrading an ASP-solving approach, the Cut rule must thus not be restricted to either only atoms or bodies. Note that these results hold for any ASP-solver (or algorithm) whose proceeding can be described by tableaux of a corresponding calculus.

Regarding the relation between SAT-based and genuine ASP-solvers, we have seen that unfounded set handling constitutes the major difference. Though both approaches, as practiced by solvers, appear to be quite different, the aims and effects of underlying tableau rules are very similar. We expect that this observation will lead to convergence of SAT-based and genuine ASP-solvers, in the sense that the next generation of genuine ASP-solvers will directly incorporate the same powerful reasoning strategies that are already exploited in the area of SAT (Mitchell 2005).

Acknowledgments. This work was supported by DFG (SCHA 550/6-4). We are grateful to Christian Anger, Philippe Besnard, Martin Brain, Yulyia Lierler, and the anonymous referees for many helpful suggestions.

References

Anger, C.; Gebser, M.; Linke, T.; Neumann, A.; and Schaub, T. 2005. The `nomore++` approach to answer set solving. In Sutcliffe, G., and Voronkov, A., eds., *LPAR*, 95–109. Springer.

Anger, C.; Gebser, M.; and Schaub, T. 2006a. Approaching the core of unfounded sets. In Dix, J., and Hunter, A., eds., *NMR*.

Anger, C.; Gebser, M.; and Schaub, T. 2006b. What's a head without a body. In Brewka, G., ed., *ECAI*, to appear.

Apt, K.; Blair, H.; and Walker, A. 1987. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann. 89–148.

Babovich, Y., and Lifschitz, V. 2003. Computing answer sets using program completion. Unpublished draft.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press.

Beame, P., and Pitassi, T. 1998. Propositional proof complexity: Past, present, and future. *EATCS*, 65:66–89.

Bonatti, P. 2001. Resolution for skeptical stable model semantics. *J. AR*, 27(4):391–421.

Calimeri, F.; Faber, W.; Leone, N.; and Pfeifer, G. 2001. Pruning operators for answer set programming systems. INFSYS RR-1843-01-07, TU Wien.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*, Plenum. 293–322.

D'Agostino, M.; Gabbay, D.; Hähnle, R.; and Posegga, J., eds. 1999. *Handbook of Tableau Methods*, Kluwer.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *J. ACM*, 7:201–215.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem-proving. *C. ACM*, 5:394–397.

Faber, W. 2002. *Enhancing Efficiency and Expressiveness in Answer Set Programming Systems*. Dissertation, TU Wien.

Fages, F. 1994. Consistency of clark's completion and the existence of stable models. *J. MLCS*, 1:51–60.

Fitting, M. 1994. Tableaux for logic programming. *J. AR*, 13(2):175–188.

Fitting, M. 2002. Fixpoint semantics for logic programming: A survey. *TCS*, 278(1-2):25–51.

Giunchiglia, E., and Maratea, M. 2005. On the relation between answer set and SAT procedures (or, cmodels and smodels). In Gabbrielli, M., and Gupta, G., eds., *ICLP*, 37–51. Springer.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. A SAT-based polynomial space algorithm for answer set programming. In Delgrande, J., and Schaub, T., eds., *NMR*, 189–196.

Hähnle, R. 2001. Tableaux and related methods. In Robinson, J., and Voronkov, A., eds., *Handbook of Automated Reasoning*, Elsevier and MIT Press. 100–178.

Järvisalo, M.; Junttila, T.; and Niemelä, I. 2005. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *AMAI*, 44(4):373–399.

Junttila, T., and Niemelä, I. 2000. Towards an efficient tableau method for boolean circuit satisfiability checking. In Lloyd, J.; et al., eds., *CL*, 553–567. Springer.

Konczak, K.; Linke, T.; and Schaub, T. 2006. Graphs and colorings for answer set programming. *TPLP*, 6(1-2):61–106.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In Kaelbling, L., and Saffiotti, A., eds., *IJCAI*, 503–508. Professional Book Center.

Leone, N.; Faber, W.; Pfeifer, G.; Eiter, T.; Gottlob, G.; Koch, C.; Mateis, C.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM TOCL*, to appear.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM TOCL*, to appear.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *AIJ*, 157(1-2):115–137.

Mitchell, D. 2005. A SAT solver primer. *EATCS*, 85:112–133.

Olivetti, N. 1999. Tableaux for nonmonotonic logics. In D'Agostino et al. (1999), 469–528.

Pearce, D.; de Guzmán, I.; and Valverde, A. 2000. A tableau calculus for equilibrium entailment. In Dyckhoff, R., ed., *TABLEAUX*, 352–367. Springer.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *AIJ*, 138(1-2):181–234.

Simons, P. 2000. Extending and implementing the stable model semantics. Dissertation, Helsinki UT.

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650.

1.6 Approaching the Core of Unfounded Sets

Approaching the Core of Unfounded Sets

Christian Anger and **Martin Gebser** and **Torsten Schaub***
Institut für Informatik
Universität Potsdam
Postfach 90 03 27, D–14439 Potsdam
{christian, gebser, torsten}@cs.uni-potsdam.de

Abstract

We elaborate upon techniques for unfounded set computations by building upon the concept of loops. This is driven by the desire to minimize redundant computations in solvers for Answer Set Programming. We begin by investigating the relationship between unfounded sets and loops in the context of partial assignments. In particular, we show that subset-minimal unfounded sets correspond to active elementary loops. Consequentially, we provide a new loop-oriented approach along with an algorithm for computing unfounded sets. Unlike traditional techniques that compute greatest unfounded sets, we aim at computing small unfounded sets and rather let propagation (and iteration) handle greatest unfounded sets. This approach reflects the computation of unfounded sets employed in the *nomore++* system. Beyond that, we provide an algorithm for identifying active elementary loops within unfounded sets. This can be used by SAT-based solvers, like *assat*, *cmodels*, or *pbmodels*, for optimizing the elimination of invalid candidate models.

Introduction

Search strategies of solvers for *Answer Set Programming* (ASP) naturally decompose into a deterministic and a non-deterministic part. While the non-deterministic part is realized through heuristically driven choice operations, the deterministic one is based on advanced propagation operations, often amounting to the computation of *well-founded semantics* (van Gelder *et al.* 1991). The latter itself can be broken up into techniques realizing *Fitting's operator* (Fitting 2002) and the computation of *unfounded sets* (van Gelder *et al.* 1991). The notion of an unfounded set captures the intuition that its atoms might circularly support themselves but have no support from "outside." Hence, there is no reason to believe in the truth of an unfounded set, and the contained atoms must be false. The opposites of unfounded sets are *externally supported sets* (Lee 2005), their atoms have a non-circular support.

While genuine ASP-solvers, like *dlv* (Leone *et al.* 2006) and *smodels* (Simons *et al.* 2002), aim at determining *greatest* unfounded sets, SAT-based ASP-solvers, like *assat* (Lin & Zhao 2004), *cmodels* (Lierler & Maratea 2004), and *pbmodels* (Liu & Truszczyński 2005), use *loops* and associated *loop formulas* (Lin & Zhao 2004; Lee 2005) for eliminating models containing unfounded sets. Both approaches comprise certain redundancies: For instance, not all elements of a greatest unfounded set need to be determined by special-purpose unfounded set techniques. Alternatively, one may restrict attention to crucial unfounded sets and handle the remaining ones via simpler forms of propagation and iteration. In fact, we show that a subset of a program's loops grants the same propagation strength as obtained with greatest unfounded sets. Further on, the problem with the standard concept of loops is that it tolerates the generation of ineffective loop formulas within SAT-based solvers. That is, unfounded subsets of a loop might recur, causing the need to generate additional loop formulas. Both redundancy issues are addressed by *(active) elementary loops* (Gebser & Schaub 2005), on which the computational approaches presented in this paper build upon.

We consider two diametrical computational tasks dealing with unfounded sets: first, falsification of greatest unfounded sets and, second, identification of subset-minimal unfounded sets. Greatest unfounded sets are worthwhile when the aim is setting unfounded atoms to false, as done within genuine ASP-solvers. Subset-minimal unfounded sets can serve best when one needs to eliminate an undesired model of a program's *completion* (Clark 1978) by a loop formula, which is important for SAT-based solvers.

First, we turn our attention to greatest unfounded sets computed by genuine ASP-solvers. In *dlv*, operator $\mathcal{R}_{\Pi_C,I}$ is applied to so-called *head-cycle-free components* C of a disjunctive program Π, where I is a (partial) interpretation (Calimeri *et al.* 2001).[1] The fixpoint, $\mathcal{R}^{\omega}_{\Pi_C,I}(C)$, of this operator is the greatest unfounded set with respect to I, restricted to atoms inside C.[2] Component-wise unfounded set identification is in *dlv* achieved by computing complements,

* Affiliated with the School of Computing Science at Simon Fraser University, Canada.

[1] Such a component C is a strongly connected component of the atom dependency graph, where positive as well as negative dependencies (through not) contribute edges. Head-cycle-freeness additionally assures tractability of unfounded set checks, which otherwise are intractable for disjunctive programs.

[2] Note that a "global" greatest unfounded set is not guaranteed to exist for a disjunctive program (Leone *et al.* 1997). However, a head-cycle-free component always has a "local" greatest unfounded set, which can be computed in linear time.

that is, $C \setminus \mathcal{R}^{\omega}_{\Pi_C,I}(C)$. This set is externally supported, all other atoms of C form the greatest unfounded set.

In *smodels*, unfounded set computation follows a similar idea. The respective function, called *Atmost*, is based on *source pointers* (Simons 2000). Each non-false atom has a source pointer indicating a rule that provides an external support for that atom. When some source pointers are invalidated (in effect of a choice), *Atmost* proceeds as follows:

Iterate over the strongly connected components of a program's *(positive) atom dependency graph* (see next section). For the current component, do:

1. Remove source pointers that point to rules whose bodies are false.
2. Remove further source pointers that point to rules whose positive bodies contain some atoms currently not having source pointers themselves.
3. Determine new source pointers if possible. That is, re-establish source pointers of atoms that are heads of rules with non-false bodies such that all atoms in the positive parts have source pointers themselves.
4. All atoms without a new source pointer are unfounded. Set them to false (possibly invalidating source pointers of other components' atoms) and proceed.

Essentially, Step 1 and 2 check for atoms that might be unfounded due to rules whose bodies have recently become false. Afterwards, Step 3 determines the atoms that are still externally supported and, hence, not unfounded. Observe that the atoms to falsify as a result of Step 4 are precisely the ones that are not found externally supported in the step before. Thus, both *smodels* and *dlv* compute greatest unfounded sets as complements of externally supported sets. Notably, computations are modularized to strongly connected components of atom dependency graphs.

Having considered the falsification of greatest unfounded sets, we now turn to the diametrical problem: determining subset-minimal unfounded sets. The ability to compute subset-minimal unfounded sets is attractive for SAT-based solvers, which compute (propositional) models of a program's completion. Whenever a computed candidate model does not correspond to an answer set,[3] a loop formula that eliminates the model is added to the completion. For the loop formula eliminating the model, the respective loop must be unfounded. SAT-based solver *assat* determines so-called *terminating* loops (Lin & Zhao 2004), which are subset-maximal unfounded loops. Terminating loops are easy to compute: They are strongly connected components of the (positive) atom dependency graph induced by the greatest unfounded set. Given that terminating loops are not necessarily subset-minimal unfounded sets, their loop formulas condense the reason why a model is invalid less precisely than the ones of subset-minimal unfounded sets.

In this paper, we present a novel approach to achieving the aforementioned computational tasks. In fact, both tasks are settled on the same theoretical fundament. On the one hand, we can explain strategies of genuine ASP-solvers to handle greatest unfounded sets, also we present the strategy recently implemented in *nomore++* (Anger *et al.* 2005). On the other hand, we point out how our approach can be exploited by SAT-based solvers for determining more effective loop formulas. The overall contributions are:

- We relate the notion of elementary loops to unfounded sets in the context of partial assignments. Thereby, we reveal unfounded sets that must intrinsically be considered by both SAT-based and genuine ASP-solvers. The developed theoretical fundament fortifies new approaches to computational tasks dealing with unfounded sets.
- We describe a novel algorithm for computing unfounded sets in a loop-oriented way. The algorithm determines unfounded sets directly, avoiding the complementation of externally supported sets. This approach allows us to immediately propagate falsity of atoms in a detected unfounded set and to postpone unprocessed unfounded set checks. We thereby achieve a tighter coupling of unfounded set checks with simpler forms of propagation and localize the causes and effects of operations. The algorithm has recently been implemented in *nomore++*, but may be integrated into other solvers, e.g., *dlv*, as well.
- We present an algorithm for extracting active elementary loops from unfounded sets. The algorithm, which is the first of its kind, exploits particular properties of active elementary loops, building the "cores" of unfounded sets. Active elementary loops can replace terminating loops in SAT-based solvers. Note that a terminating loop is not guaranteed to be elementary, hence, a respective loop formula might be redundant (Gebser & Schaub 2005). Our algorithm can be integrated in solvers like *assat*, *cmodels*, and *pbmodels*. Such an integration could form the base for an empirical evaluation of the effectiveness of active elementary loops.

Background

Given an alphabet $\mathcal{P}$, a (normal) *logic program* is a finite set of rules of the form

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \qquad (1)$$

where $0 \leq m \leq n$ and each $p_i \in \mathcal{P}$ $(0 \leq i \leq n)$ is an *atom*. A *literal* is an atom p or its negation $not\ p$. For a rule r as in (1), let $head(r) = p_0$ be the *head* of r and $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ be the *body* of r. Given a set X of literals, let $X^+ = \{p \in \mathcal{P} \mid p \in X\}$ and $X^- = \{p \in \mathcal{P} \mid not\ p \in X\}$. For $body(r)$, we then get $body(r)^+ = \{p_1, \ldots, p_m\}$ and $body(r)^- = \{p_{m+1}, \ldots, p_n\}$. The set of atoms occurring in a logic program Π is denoted by $atom(\Pi)$. The set of bodies in Π is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping rule bodies sharing the same head p, define $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$. A program Π is called *positive* if $body(r)^- = \emptyset$ for all $r \in \Pi$. $Cn(\Pi)$ denotes the smallest set of atoms closed under positive program Π. The *reduct*, Π^X, of Π relative to a set X of atoms is defined by $\Pi^X = \{head(r) \leftarrow body(r)^+ \mid r \in \Pi,$

[3] Any answer set of a program is a model of the program's completion, whereas the converse does generally not hold (Fages 1994).

$body(r)^- \cap X = \emptyset\}$. A set X of atoms is an *answer set* of a logic program Π if $Cn(\Pi^X) = X$.

An unfounded set is defined relative to an *assignment*. In *nomore++*, values are assigned to both atoms and bodies, whereas *smodels* and *dlv* explicitly assign values only to atoms (from which the (in)applicability of rules is determined). Note that an assignment to atoms and bodies can reflect any state resulting from an assignment to atoms, whereas the converse does not hold because a body might be false without yet containing a false literal. Also, the restriction of assignments to atoms limits search to branching on atoms, which may lead to exponentially worse proof complexity than obtained when branching on both atoms and bodies (Gebser & Schaub 2006). Given that assignments to both atoms and bodies provide extra value, we define an *assignment* $\boldsymbol{A}$ for a program Π as a (total) function:

$$\boldsymbol{A} : atom(\Pi) \cup body(\Pi) \to \{\ominus, \odot, \otimes, \oplus\}$$

The four values correspond to those used by *dlv* (Faber 2002); that is, $\ominus$ stands for *false*, $\odot$ for *undefined*, $\otimes$ for *must-be-true*, and $\oplus$ for *true*.[4] We also assume that the abstract ASP-solver, invoking the algorithms presented in the following sections, propagates the four values like *dlv* applied to normal programs (which approximates propagation within *nomore++*) and do not provide any details here.[5] We call an assignment $\boldsymbol{A}$ *positive-body-saturated*, abbreviated *pb-saturated*, if for every $B \in body(\Pi)$, $\boldsymbol{A}(B) = \ominus$ if $\boldsymbol{A}(p) = \ominus$ for some $p \in B^+$. An arbitrary assignment is easily turned into a pb-saturated one by propagation.

What is important to note is the difference between $\otimes$ (must-be-true) and $\oplus$ (true). For our unfounded set check to work, the following invariant must hold for any assignment $\boldsymbol{A}$:

$$\{p \in atom(\Pi) \mid \boldsymbol{A}(p) = \oplus\} \cup \left(\bigcup_{B \in body(\Pi), \boldsymbol{A}(B) = \oplus} B^+\right) \subseteq Cn(\{r \in \Pi \mid \boldsymbol{A}(body(r)) = \oplus\}^\emptyset) \quad (2)$$

The invariant stipulates that all atoms and (positive parts of) bodies assigned $\oplus$ are bottom-up derivable within the part of Π assigned $\oplus$. This guarantees that no unfounded set ever contains an atom assigned $\oplus$, and we can safely exclude such atoms as well as bodies assigned $\oplus$ from unfounded set checks. Hence, the invariant helps in avoiding useless work. It also allows for "lazy" unfounded set checks, on which we will come back when discussing the relation of our unfounded set algorithm to *smodels*. Invariant (2) can be maintained by assigning $\oplus$ to an atom, only if some of its bodies is already assigned $\oplus$, and to a body, only if all atoms in the positive part are already assigned $\oplus$. Otherwise, $\otimes$ must be assigned instead of $\oplus$.

[4]Note that the concept of an assignment is to be understood in the sense of a constraint satisfaction problem, rather than an interpretation. This is because answer sets are defined as models that are represented by their entailed atoms. By assigning values to bodies, which can be viewed as conjunctions, we do not construct such a model but deal with problem-relevant variables. For this reason, we use symbolic values instead of ascribed truth values.

[5]When referring to propagation, we mean any technique that deterministically extends assignments except for unfounded set checks, to be detailed in the following sections.

We now come to unfounded sets. For a program Π, we define a set $U \subseteq atom(\Pi)$ as an *unfounded set* with respect to an assignment $\boldsymbol{A}$ if, for every rule $r \in \Pi$, we have either

- $head(r) \notin U$,
- $\boldsymbol{A}(body(r)) = \ominus$, or
- $body(r)^+ \cap U \neq \emptyset$.

Our definition is close to the original one (van Gelder *et al.* 1991), but differs regarding the second condition, which aims at inapplicable rules. With the original definition, such rules are determined from atoms, that is,

- $\{p \in body(r)^+ \mid \boldsymbol{A}(p) = \ominus\} \neq \emptyset$ or
- $\{p \in body(r)^- \mid \boldsymbol{A}(p) = \otimes \text{ or } \boldsymbol{A}(p) = \oplus\} \neq \emptyset$.

The reason for not determining inapplicable rules from atoms is that, with our definition of an assignment, a body assigned $\ominus$ needs not necessarily contain a false literal. Rather, a body might be inapplicable, that is, assigned $\ominus$, due to any reason (such as a choice or an inference by lookahead). Still it holds that normal programs (in contrast to disjunctive ones (Leone *et al.* 1997)) enjoy the property that the union of distinct unfounded sets is itself an unfounded set. Hence, there always is a *greatest unfounded set*, denoted $GUS_\Pi(\boldsymbol{A})$, for any program Π and any assignment $\boldsymbol{A}$.

Finally, we come to loops, which are sets of atoms involved in cyclic program structures. Traditionally, program structure is described by means of atom dependency graphs (Apt *et al.* 1987). When we restrict attention to unfounded sets, it is sufficient to consider *positive* atom dependency graphs. For a program Π, the *(positive) atom dependency graph* is the directed graph $(atom(\Pi), E)$ where $E = \{(p, p') \mid r \in \Pi, p = head(r), p' \in body(r)^+\}$. That is, the head of a rule has an edge to each atom in the positive body. Following (Lee 2005), we define a *loop* L in a program Π as a non-empty subset of $atom(\Pi)$ such that, for any two elements $p \in L$ and $p' \in L$, there is a path from p to p' in the atom dependency graph of Π all of whose vertices belong to L. In other words, the subgraph of the atom dependency graph of Π induced by L is strongly connected. Note that each set consisting of a single atom is a loop, as every atom is connected to itself via a path of length zero.

The significance of loops has first been recognized in (Lin & Zhao 2002), where the concept was also originally defined.[6] In fact, program completion and loop formulas capture answer sets in terms of propositional models. The advantage of loops and their formulas, in comparison to other SAT-reductions (e.g. (Janhunen 2003; Lin & Zhao 2003)), is that the reduction can be done incrementally (SAT-based solvers *assat*, *cmodels*, and *pbmodels* pursue this strategy); the increase in problem size is very small in the best case. The downside is that a program may yield exponentially many loops, leading to exponential worst-case space complexity of loop-based SAT-reductions (Lifschitz & Razborov 2006). Genuine ASP-solvers can, however, exploit loops

[6]Note that in (Lin & Zhao 2002) loops' atoms must be connected via paths of *non-zero* length. By dropping this requirement, we can relate loops and unfounded sets more directly.

without explicitly representing loop formulas. In what follows, we relate loops to unfounded sets paving the way to loop-oriented unfounded set computations. The difference to SAT-based approaches is that we consider loops in the context of partial assignments, and not with respect to total (propositional) models.

Relating Unfounded Sets and Loops

Recall the definition of an unfounded set given in the previous section. It states that any rule whose head belongs to an unfounded set is either inapplicable or contains an unfounded atom in the positive part of the body. Since unfounded sets are finite, the following is a consequence.

Proposition 1 *Let Π be a logic program, $\boldsymbol{A}$ be an assignment, and U be an unfounded set w.r.t. $\boldsymbol{A}$.*

If $U \neq \emptyset$, we have $L \subseteq U$ for some loop L in Π that is unfounded w.r.t. $\boldsymbol{A}$.

This result establishes that any non-empty unfounded set is a superset of some loop that is itself unfounded. Note that Proposition 1 would not hold, if we had defined loops according to (Lin & Zhao 2004), where the contained atoms must be connected via paths of non-zero length. Omitting this, a singleton unfounded set $\{p\}$ such that all bodies in $body(p)$ are assigned $\ominus$ is a loop. Otherwise, some element from an unfounded set U must be contained in B^+, if B is the body of a rule whose head is in U and not assigned $\ominus$. The latter condition gives rise to inherent cyclicity.

When dealing with greatest unfounded sets, one usually concentrates on the part of an assignment not assigned $\ominus$. In fact, for an atom p assigned $\ominus$ and a set U of atoms such that $p \in U$, any body B such that $p \in B^+$ satisfies the condition of containing an element from U as well as the condition of containing a false literal. Since the latter condition is easy to verify, it is reasonable to exclude atoms assigned $\ominus$ when looking for the relevant part of a greatest unfounded set.

However, our definition of an unfounded set does not look "through" bodies for determining inapplicability. As a minimum requirement, we thus need an assignment to be pb-saturated, before a relevant unfounded set is determined. Certainly this requirement is reasonable, while working on "unsynchronized" assignments of atoms and bodies would be rather weird. For a pb-saturated assignment, the non-false part of the greatest unfounded set is an unfounded set.

Lemma 1 *Let Π be a logic program and $\boldsymbol{A}$ be a pb-saturated assignment.*

Then, $\{p \in GUS_\Pi(\boldsymbol{A}) \mid \boldsymbol{A}(p) \neq \ominus\}$ is an unfounded set w.r.t. $\boldsymbol{A}$.

Combining Proposition 1 and Lemma 1 yields the following.

Theorem 1 *Let Π be a logic program, $\boldsymbol{A}$ be a pb-saturated assignment, and $U = \{p \in GUS_\Pi(\boldsymbol{A}) \mid \boldsymbol{A}(p) \neq \ominus\}$.*

If $U \neq \emptyset$, we have $L \subseteq U$ for some loop L in Π that is unfounded w.r.t. $\boldsymbol{A}$.

The above result is the "partial assignment counterpart" of (Lin & Zhao 2004, Theorem 2), where the latter refers to total (propositional) models. Due to Theorem 1, we can concentrate greatest unfounded set computation on loops: By successively falsifying the atoms of unfounded loops and pb-saturating the resulting assignment, we eventually falsify all atoms in a greatest unfounded set. Clearly, more advanced propagation techniques (such as contraposition) can be applied in addition to pb-saturation. Theorem 1 still grants that there always is an unfounded loop whose atoms are not assigned $\ominus$, as long as there are non-false atoms left in the greatest unfounded set. Note that all answer set solvers we know of apply propagation techniques that are at least as strong as Fitting's operator (Fitting 2002). Whenever this operator has reached a fixpoint, all singleton loops $\{p\}$ such that all bodies in $body(p)$ are assigned $\ominus$ are already set to false. More sophisticated unfounded set checks can thus concentrate on loops as defined in (Lin & Zhao 2004).

Up to now, we have considered loops, which are defined by means of atom dependency graphs. Such graphs do not reflect program-specific connection via the bodies of rules. Given that we are interested in intrinsically relevant unfounded sets, loops are not yet fine-grained enough. To see this, consider the following programs:

$$\begin{array}{lllll} \Pi_1 & = & \{\ a \leftarrow & b \leftarrow a, c & c \leftarrow b\ \} \\ \Pi_2 & = & \{\ a \leftarrow & b \leftarrow a \quad b \leftarrow c & c \leftarrow b\ \} \end{array}$$

Though sharing the same atom dependency graph, the single answer set of Π_1 is $\{a\}$, whereas we obtain $\{a, b, c\}$ for Π_2. The reason for this is that the apparently different rules, $b \leftarrow a, c$ in Π_1 as well as $b \leftarrow a$ and $b \leftarrow c$ in Π_2, contribute the same edges to an atom dependency graph. However, rule $b \leftarrow a$ provides an external support for the set $\{b, c\}$, whereas rule $b \leftarrow a, c$ does not.

For distinguishing between putative and virtual external supports, we have to consider elementary loops (Gebser & Schaub 2005). We define a loop L in a program Π as *elementary* if, for each non-empty proper subset K of L, there is a rule $r \in \Pi$ such that

- $head(r) \in K$,
- $body(r)^+ \cap K = \emptyset$, and
- $body(r)^+ \cap L \neq \emptyset$.

In words, a loop is elementary if each of its non-empty proper subsets has a rule whose head is in the subset and whose body positively relies on the loop, but not on the subset itself.[7] A particular property of elementary loops, rather than general ones, is that they potentially provide an external support to any of their non-empty proper subsets, even when they are unfounded. If such a situation arises, we say that an elementary loop is active. Formally, an elementary loop L in a program Π is *active* w.r.t. an assignment $\boldsymbol{A}$ if

- L is an unfounded set w.r.t. $\boldsymbol{A}$ and
- L is elementary in $\{r \in \Pi \mid \boldsymbol{A}(body(r)) \neq \ominus\}$.

Due to the first condition, an active elementary loop always is unfounded. The next result tells us that any non-empty unfounded set contains an active elementary loop.

[7] In analogy to general loops, every singleton is an elementary loop by definition. This is different from (Gebser & Schaub 2005), where loops are defined according to (Lin & Zhao 2004).

Proposition 2 *Let Π be a logic program, $\boldsymbol{A}$ be an assignment, and U be an unfounded set w.r.t. $\boldsymbol{A}$.*

If $U \neq \emptyset$, we have $L \subseteq U$ for some elementary loop L in Π that is active w.r.t. $\boldsymbol{A}$.

This result strengthens Proposition 1. For a pb-saturated assignment, it together with Lemma 1 grants the existence of an active elementary loop none of whose atoms is assigned $\ominus$, whenever the greatest unfounded set contains non-false atoms. It is thus sufficient to concentrate unfounded set computations on active elementary loops. Going beyond is impossible: Any non-empty proper subset of an active elementary loop is externally supported.

Proposition 3 *Let Π be a logic program, $\boldsymbol{A}$ be an assignment, and L be an active elementary loop in Π w.r.t. $\boldsymbol{A}$.*

Then, any non-empty proper subset of L is not unfounded w.r.t. $\boldsymbol{A}$.

The following is a "partial assignment counterpart" of results on total (propositional) interpretations in (Gebser *et al.* 2006).[8] It is a consequence of Proposition 2 and 3.

Theorem 2 *Let Π be a logic program, $\boldsymbol{A}$ be an assignment, and $L \subseteq atom(\Pi)$.*

Then, L is an active elementary loop in Π w.r.t. $\boldsymbol{A}$ iff L is a subset-minimal non-empty unfounded set w.r.t. $\boldsymbol{A}$.

This result shows that active elementary loops form in fact the "cores" of unfounded sets. Any proper superset of an active elementary loop contains atoms that are unnecessary for identifying (parts of) the set as unfounded. In turn, no non-empty proper subset of an active elementary loop can be identified as unfounded. Active elementary loops motivate novel computational approaches in two aspects: First, they can be used to make unfounded set computations less exhaustive by not aiming at greatest unfounded sets; second, they reveal intrinsically relevant unfounded sets and rule out superfluous ones. In the next sections, we provide respective computational approaches.

Greatest Unfounded Sets

We now exploit Theorem 1 and Proposition 2, granting the existence of an active elementary loop as a subset of the non-false part of a greatest unfounded set, and design an algorithm aiming at such loops. In order to restrict computations to necessary parts, we make the following assumptions:

- Invariant (2) on assignments holds. It guarantees that neither an atom assigned $\oplus$ nor an atom from the positive part of a body assigned $\oplus$ is unfounded.
- If, for an atom, the bodies of all rules with the atom as head are assigned $\ominus$, then the atom is assigned $\ominus$. Vice versa, an atom is assigned $\oplus$ if it has a body assigned $\oplus$.
- A body is assigned $\ominus$ if some of its literals is false, that is, an atom from the positive part is assigned $\ominus$ or one from the negative part is assigned either $\otimes$ or $\oplus$. Also, a body is assigned $\oplus$ if all atoms in its positive part are assigned $\oplus$ and all atoms in the negative part $\ominus$.

Due to the first assumption, the external support of atoms and bodies assigned $\oplus$ is granted. Furthermore, atoms and bodies already assigned $\ominus$ need not be considered anyway. We can thus restrict attention to atoms and bodies assigned either $\odot$ or $\otimes$. The second and third assumption grant that anything decidable by Fitting's operator is already assigned. (Note that this implies assignments to be pb-saturated.) Fixpoints of Fitting's operator are computed by *dlv*, *smodels*, and *nomore++* before an unfounded set check is initiated.

The unfounded sets we are aiming at are loops. Loops are bound from above by the strongly connected components of a program's atom dependency graph. For conveniently arranging both atoms and bodies into strongly connected components, we extend dependency graphs to bodies. For a program Π, we define the *(positive) atom-body dependency graph* as the directed graph $(atom(\Pi) \cup body(\Pi), E \cup E_0)$ where $E = \{(head(r), body(r)) \mid r \in \Pi\}$ and $E_0 = \{(body(r), p) \mid r \in \Pi, p \in body(r)^+\}$.[9] The strongly connected components of such graphs are understood in the standard graph-theoretical sense, loops are the atoms contained in strongly connected subgraphs.

We are now ready to describe our algorithm for computing an unfounded set. It accesses the following global variables.

Π**:** The underlying logic program.

$\boldsymbol{A}$**:** The current assignment.

SCC**:** The vertices of a strongly connected component of the atom-body dependency graph of Π.

Set**:** A set of atoms such that $\mathsf{Set} \subseteq \mathsf{SCC} \cap atom(\Pi)$.

Ext**:** The set $\mathsf{Ext} = \bigcup_{p \in \mathsf{Set}} \{B \in body(p) \mid B^+ \cap \mathsf{Set} = \emptyset, \boldsymbol{A}(B) \neq \ominus\}$ of bodies.

Source**:** A subset of $body(\Pi)$.

Sink**:** A subset of $atom(\Pi)$.

Variable Set contains the atoms to be extended to an unfounded set. All atoms in Set belong to the same strongly connected component: SCC. Set Ext of bodies can be thought of as a todo list. It comprises bodies that provide external supports for the atoms in Set, hence, some atoms from their positive parts must be added to Set. Synonymously to *smodels*' source pointers, set Source contains bodies for which it is known that external supports for their positive parts exist. Set Sink contains atoms some of whose non-false bodies are in Source or in a different strongly connected component; such atoms are not contained in any unfounded set. A source pointer in *smodels* can be thought of as a link from an atom in Sink to a body in Source or outside SCC.

Our unfounded set algorithm is shown in Algorithm 1. The designated initial situation is that some atom, assigned either $\odot$ or $\otimes$, has been chosen to start an unfounded set check from. This atom is initially contained in Set, its "external bodies" in Ext. For the computation being reasonable, each external body is supposed to be contained in

[8] Please note that the reformulation of (active) elementary loops provided here is inspired by the notion of an *elementary set* (Gebser *et al.* 2006), for which similar results in the context of total (propositional) interpretations were developed first.

[9] So-called *body-head graphs* are used in (Linke & Sarsakov 2005) for describing isomorphisms between dependency graphs and syntactically restricted program classes.

Algorithm 1: UNFOUNDED SET

```
1  while Ext ≠ ∅ do
2      Ext ← Ext \ {B} for some B ∈ Ext
3      if there is some p ∈ B+ ∩ SCC such that p ∉ Sink and A(p) ≠ ⊕ then
4          J ← {B ∈ body(p) | B ∉ SCC, A(B) ≠ ⊖} ∪
               {B ∈ body(p) | B ∈ Source, A(B) ≠ ⊖}
5          if J = ∅ then
6              Set ← Set ∪ {p}
7              Ext ← Ext \ {B ∈ Ext | p ∈ B+}
8              Ext ← Ext ∪ {B ∈ body(p) | B+ ∩ Set = ∅, A(B) ≠ ⊖}
9          else
10             Sink ← Sink ∪ {p}
11             Ext ← Ext ∪ {B}
12     else
13         Source ← Source ∪ {B}
14         R ← {p ∈ Set | B ∈ body(p)}
15         while R ≠ ∅ do
16             Set ← Set \ R
17             Sink ← Sink ∪ R
18             J ← {B ∈ body(Π) ∩ SCC | B+ ∩ R ≠ ∅, A(B) ≠ ⊖,
                       {p ∈ B+ ∩ SCC | p ∉ Sink, A(p) ≠ ⊕} = ∅}
19             Source ← Source ∪ J
20             R ← {p ∈ Set | body(p) ∩ J ≠ ∅}
21         Ext ← ⋃p∈Set {B ∈ body(p) | B+ ∩ Set = ∅, A(B) ≠ ⊖}
```

SCC \ Source. The outer while-loop from line 1 to 21 is iterated as long as there are external bodies. Note that we have Ext $= \emptyset$ whenever Set $= \emptyset$; in this case, the empty Set indicates that no unfounded set contains any atom that has temporarily been in Set.

If Ext $\neq \emptyset$, we select in line 2 an external body B from whose positive part an atom should be added to Set next. Such an atom p must be contained in SCC, but not in Sink, and not be assigned $\oplus$ (line 3). If there is such an atom p, we determine in line 4 all bodies of atom p that are not assigned $\ominus$ and either not contained in SCC or contained in Source. If such bodies exist, that is, J $\neq \emptyset$, p is externally supported, and we add it to Sink (line 10). Otherwise, we can extend Set with atom p (line 6). All bodies that were formerly external but positively rely on p are then removed from Ext (line 7). Finally, we add bodies of rules with head p to Ext if they do not positively rely on Set and are not assigned $\ominus$ (line 8).

From line 12 to 21, we handle the case that no atom from the positive part of body B can be added to Set. Then, we add B to Source as it is externally supported (line 13). In line 14, we determine the atoms from Set that occur as heads of rules with body B. These atoms are as well externally supported and must be removed from Set. Note that we always have R $\neq \emptyset$ because B occurs as body of at least one atom in Set. From line 15 to line 20, we remove atoms from Set and add them to Sink as long as further bodies and associated head atoms are found externally supported. The crucial line is 18: Here we determine bodies B from SCC, not assigned $\ominus$, such that some atoms in the positive part have recently been removed from Set ($B^+ \cap$ R $\neq \emptyset$) and all other atoms are either not contained in SCC, contained in Sink, or assigned $\oplus$. In a bottom-up fashion, we derive such externally supported bodies and add them to Source (line 19), respective head atoms are successively removed from Set and added to Sink (lines 16, 17, and 20). Finally, we update in line 21 the external bodies of the atoms still in Set.

Like unfounded set detection algorithms of *dlv* and *smodels*, Algorithm 1 can be implemented such that it works in linear time. The distinguishing element to other algorithms is that it extends the set of considered atoms on demand, that is, if there are bodies from whose positive parts no atom is included yet. The algorithm stops and does not explore any more atoms when such bodies do not exist. The aim is to keep a computed unfounded set as small as possible. This is motivated as follows: Propagation of single atoms and bodies can be done very efficiently and does, in contrast to unfounded set checks, not risk "wasted" work yielding no inferences. Simpler forms of propagation, like Fitting's operator, are thus in *nomore++* applied as soon as possible, in the hope that pending unfounded set checks can be avoided in effect. For enabling such "early" propagation, it is important that we compute unfounded sets directly, as it is done by Algorithm 1, and do not complement externally supported sets, as done within *dlv* and *smodels*.

Let us now consider ways of integrating Algorithm 1 into solvers. Any solver using Algorithm 1 has to grant that potential external support for bodies in Source and atoms in Sink really exists, since the elements of these sets are not examined by the algorithm. The same applies to atoms and bodies assigned $\oplus$. Systems *dlv* and *nomore++* assure the latter by assigning *must-be-true* or $\otimes$, when later unfoundedness of a true atom or body cannot be excluded. Detecting unfoundedness of program parts that must be true leads to a conflict, which has to be detected for soundness reasons.

The strategy of *smodels* is different, it does not use an analog for $\otimes$. Unfounded program parts, whether they contain true elements or not, are determined from source pointers. Such source pointers correspond to elements of Source and Sink. They are maintained during the solving process, and invalid ones are removed during the "first stage" of function *Atmost*, before it performs the actual unfounded set check. For a true atom, the removal of its source pointer can be seen as turning the value from $\oplus$ to $\otimes$, in order to make the atom accessible to a pending unfounded set check.

In contrast to *smodels*' *Atmost*, *dlv* and *nomore++* do not have a "first stage" for canceling outdated external support information. They simply start their unfounded set computations from head atoms of rules whose bodies have become false since the last unfounded set check. (Such atoms are also the starting points for *Atmost* to remove source pointers.) Unfounded set checks are done locally for strongly connected components of the respective dependency graphs. After processing a component, no information is kept, and no updates are necessary upon a revisit. Another parallel between *dlv* and *nomore++* is that the former propagates a component's greatest unfounded set before initiating further unfounded set checks (Faber 2006). Though not the same, this is quite similar to *nomore++* immediately propagating an unfounded set determined by Algorithm 1.

The discussion above shows that Algorithm 1 can potentially be put into various contexts, using different strategies to maintain acquired information and to combine unfounded set checks with propagation. Concerning the latter, Algorithm 1 is designed to stop as soon as an unfounded set is detected. In this way, a solver can immediately propagate falsity of the contained atoms. This allows unfounded set checks to always work on an up-to-date assignment, possibly reducing the overall efforts of a computation. Finally, let us mention that Algorithm 1, though aiming at loops, only guarantees that the atoms of a computed unfounded set belong to the same strongly connected component. They do not necessarily form a loop because of the inherent sensitivity to the order in which atoms are assumed to belong to an unfounded set (the order in which they are added to Set).

Subset-Minimal Unfounded Sets

Having considered the falsification of greatest unfounded sets, we now turn to the diametrical problem: determining subset-minimal unfounded sets, which, by Theorem 2, are active elementary loops. The ability to determine active elementary loops is attractive for SAT-based ASP-solvers, computing (propositional) models of a program's completion and adding loop formulas to eliminate invalid candidate models. To this end, the SAT-based solver *assat* determines terminating loops, which are subset-maximal unfounded loops. Clearly, terminating loops are not necessarily active elementary loops. However, the loop formula of an active elementary loop eliminates an invalid candidate model, like the one of a terminating loop. In addition, undesired models that are not eliminated by the loop formula of a terminating loop might be excluded in future invocations of the underlying SAT-solver (cf. Section 5 in (Gebser & Schaub 2005) for an example). In this section, we show how an active elementary loop can be extracted from a given unfounded set, which might be a terminating loop. Within SAT-based solvers, active elementary loops can thus replace terminating loops.

Though elementary loops, as defined before, suggest that all subsets of a loop must be examined, deciding whether a loop is elementary is tractable. Indeed, elementary loops can also be characterized by elementary subgraphs of a program's atom-body dependency graph (Gebser & Schaub 2005). For a program Π and a set $L \subseteq atom(\Pi)$, we define $B(L) = \{body(r) \mid r \in \Pi, head(r) \in L, body(r)^+ \cap L \neq \emptyset\}$ and $E(L) = \{(p, B) \mid p \in L, B \in B(L), p \leftarrow B \in \Pi\}$. The *elementary subgraph* of L in Π is the directed graph $(L \cup B(L), E(L) \cup EC(L))$ where:

$$\begin{aligned} EC^0(L) &= \emptyset \\ EC^{i+1}(L) &= EC^i(L) \cup \{(B,p) \mid B \in B(L), p \in B^+ \cap L, \\ &\qquad \text{each } p' \in B^+ \cap L \text{ has a path to } p \\ &\qquad \text{in } (L \cup B(L), E(L) \cup EC^i(L))\} \\ EC(L) &= \textstyle\bigcup_{i \geq 0} EC^i(L) \end{aligned}$$

By (Gebser & Schaub 2005, Theorem 10), the elementary subgraph allows for deciding elementariness.

Theorem 3 *Let Π be a logic program and $L \subseteq atom(\Pi)$. If $L \neq \emptyset$, L is an elementary loop in Π iff the elementary subgraph of L in Π is strongly connected.*

If a loop is elementary, its elementary subgraph has the following property (Gebser & Schaub 2005, Proposition 12).

Proposition 4 *Let Π be a logic program, L be an elementary loop in Π, and $(L \cup B(L), E(L) \cup EC(L))$ be the elementary subgraph of L in Π.*

Then, every subgraph $(L \cup B(L), E(L) \cup EC'(L))$ such that $EC'(L) \subseteq EC(L)$ and $\{B \mid (B,p) \in EC'(L)\} = B(L)$ is strongly connected.

Due to the above property, considering only a single edge from a body to a contained loop atom is sufficient for deciding elementariness by elementary subgraph construction. This "don't care" character of elementary subgraphs greatly facilities elementary loop computation: Instead of considering all edges in an atom-body dependency graph, we can select one contained atom as a canonical representative to be reached from a body. Considering the definition of elementary subgraphs, this representative should be a body atom that is reached from all other body atoms under consideration. Proceeding in this way, we can compute active elementary loops by implicitly constructing elementary subgraphs, where bodies reach canonical representatives, reflecting the single edges required to obtain a strongly connected graph.

We have now settled the fundament of Algorithm 2 for extracting an active elementary loop from an unfounded set. Algorithm 2 uses the global variable Set, containing the atoms of an unfounded set. Initially, Set might be the result of Algorithm 1 (which is not necessarily a loop) or a terminating loop. In effect of Algorithm 2, Set will contain the atoms of an active elementary loop, obtained through removing superfluous atoms. The variables Act, Q, and N are local to Algorithm 2. Set Act contains the atoms that are temporarily assumed to be elements of the final active elementary loop. Variable Q is a priority queue of atoms that need to be visited. Each atom p has an associated id, accessible via p.id, atoms in Q are then sorted by their ids in increasing order. Via operation Q.rem(), the first element of Q is removed from Q and returned. Operation Q.add(p) inserts an atom p into Q at the appropriate position, the operation has no effect if p is already contained in Q. Variable N is a counter, used to assign an id to an atom when it is visited for the first time. Besides the id, each atom p is associated with two more variables: root and exp. Integer value root stores the id of the first visited atom that positively depends on p in the elementary subgraph of Set. The set exp corresponds to a todo list of atoms that positively depend on p, but have not yet been explored. Similar to p.id, we access root and exp of an atom p via p.root and p.exp.

Before we start describing the algorithm, let us sketch its fundamental idea. The initial value for N will be |Set|, and we decrement N whenever an atom is visited for the first time. That is, an atom with a greater id is visited before the atoms with smaller ids. While exploring atoms, we make sure that an atom with a smaller id reaches all atoms with greater ids in the elementary subgraph of Set. In this way, we can safely select the contained atom with the greatest id to explore a body from. In fact, this atom is a canonical

Algorithm 2: ACTIVE ELEMENTARY LOOP

```
1  Act ← ∅
2  Q ← ∅
3  N ← |Set|
4  while N ≠ 0 do
5      p.id ← 0 for some p ∈ Set
6      Q.add(p)
7      while Q ≠ ∅ do
8          p ← Q.rem()
9          if p.id = 0 then
10             p.id ← N
11             p.root ← N
12             p.exp ← ∅
13             Act ← Act ∪ {p}
14             N ← N − 1
15             foreach B ∈ body(Π) such that p ∈ B+, B+ ∩ Set ⊆ Act,
               and A(B) ≠ ⊖ do
                   let p′ ∈ B+ ∩ Act such that
16                     p′.id = max{p.id | p ∈ B+ ∩ Act}
17                 p′.exp ← p′.exp ∪ {p ∈ Set | B ∈ body(p)}
18                 Q.add(p′)
19         if p.exp ≠ ∅ then
20             Q.add(p)
21             p.exp ← p.exp \ {p′} for some p′ ∈ p.exp
22             if p′ ∈ Act then p.root ← max{p.root, p′.root}
23             else if p′ ∈ Set then
24                 p′.id ← 0
25                 Q.add(p′)
26         else
27             if p.id = p.root then
28                 if Q ≠ ∅ or N ≠ 0 then
29                     Set ← Set \ {p ∈ Act | p.id ≤ p.id}
30                     Act ← Act \ {p ∈ Act | p.id ≤ p.id}
31             else
32                 p′ ← Q.rem()
33                 p′.root ← max{p.root, p′.root}
34                 Q.add(p′)
```

representative, as discussed below Proposition 4. Whenever an atom is not reached from any atom with a greater id in the elementary subgraph of Set or there are unvisited atoms in Set, we can safely remove all atoms with smaller ids than that of the current atom from Set. The residual atoms in Set still form an unfounded set. We are done when N reaches zero, indicating that all atoms in Set have been inspected and form an active elementary loop.

We now describe Algorithm 2. Given Set as global variable, Act and Q are initialized to be empty, and N is set to the cardinality of Set (lines 1 to 3). The outer while-loop from line 4 to 34 is iterated until N reaches zero, indicating that all atoms in Set have been inspected. As long as this is not the case, we pick an arbitrary atom p from Set, assign p.id zero, and add p to the front of Q (lines 5 and 6). The atom p with the smallest id is removed from Q in line 8. In line 9, we detect from p.id being zero that p is visited for the first time. We then initialize p.id and p.root with N, and p.exp with the empty set (lines 10 to 12). Adding p to Act in line 13 indicates that p has been visited. In line 14, we decrement N to the number of still unvisited atoms in Set.

Due to visiting an atom p for the first time, a body B such that $p \in B^+$ and $B^+ \cap Set \subseteq Act$ becomes accessible, as there is an atom in Act that is reached from all atoms of $B^+ \cap$ Act in the elementary subgraph of Set. Of course, $\boldsymbol{A}(B)$ must not be $\ominus$ since we are interested in an active elementary loop w.r.t. $\boldsymbol{A}$. These conditions are checked in line 15. For each body satisfying the conditions, some atom $p' \in B^+ \cap$ Act has the greatest id; this atom p' is determined in line 16. As discussed above, p' is a canonical representative to reach B from. Thus, we add the head atoms of body B that are in Set to p'.exp and re-add p' to Q (lines 17 and 18). Recall that the latter has no effect if p' is already contained in Q.

After having updated atoms to be explored, we process p.exp for the current atom p from line 19 to 34. If p.exp is non-empty, we re-add p to Q, making sure that p is re-visited later on, and remove some element p' to be processed next from p.exp (lines 20 and 21). The atom p' can be already visited, in which case we maximize ids of atoms reaching p among p.root and p'.root (line 22). If p' is unvisited and has not been removed from Set since it was added to p.exp, we set p'.id to zero and add p' to the front of Q (lines 24 and 25). On re-entering the outer while-loop from line 7, p' is the atom visited next. The else-case from line 26 to 34 reflects that no more atom reaches p. If p is not reached from an atom with a greater id (p.id $=$ p.root in line 27) and there are atoms not reaching p ($Q \neq \emptyset$ or $N \neq 0$ in line 28), we remove all atoms in Act whose ids are not greater than p.id from both Set and Act (lines 29 and 30). The residual atoms of Set still form an unfounded set (otherwise some of them would have reached one of the removed atoms), containing an active elementary loop by Theorem 2. Finally, the else-case from lines 31 to 34 applies when p is reached by some atom with a greater id. In this case, we have $Q \neq \emptyset$, since at least the atom picked in line 5 is still contained in Q. For not mistakenly considering an atom unreached, we propagate the greatest id of an atom reaching p to the atom p' that succeeds p in Q (line 33). Atom p', removed from Q in line 32 and re-added in line 34, is then re-visited in the next iteration of the outer while-loop from line 7.

Regarding complexity of Algorithm 2, note that a body is explored only once, when the last of its atoms contained in Set is visited for the first time. Also, atoms are added to Act only once, upon re-visits only path information is exchanged via root. Visits of bodies and accompanying updates of reached atoms are bound by the number of edges in the part of the atom-body dependency graph that contains atoms in Set and their connecting bodies.

Extracting active elementary loops from unfounded sets might not be important for genuine ASP-solvers, like *dlv*, *smodels*, and *nomore++*, only aiming at falsification of unfounded sets. But active elementary loops can play a role in SAT-based ASP-solvers, such as *assat*, *cmodels*, and *pbmodels*, since their loop formulas eliminate undesired completion models more effectively than those of terminating loops (Gebser & Schaub 2005).

Discussion

This paper contributes to computational approaches to unfounded set handling, both theoretically and practically. Unlike already done in the literature (cf. (Lin & Zhao 2004; Lee 2005)), where loops are related to total propositional models, we have put loops into the context of partial assignments. The major result is that active elementary loops form the "cores" of unfounded sets. Hence, they must intrinsically be dealt with by any ASP-solver.

Based on active elementary loops, traditional approaches to unfounded set computation can be explained. Beyond that, new algorithms exploiting active elementary loops are fortified. We have presented an algorithm that allows for computing unfounded sets directly, avoiding the complementation of externally supported sets. This approach is currently implemented in the *nomore++* system. However, it can also be incorporated into other ASP-solvers. In fact, using assignments to both atoms and bodies is not an obligation for our theoretical results and algorithms to apply, it merely allows us to state them in a way that accounts for *nomore++* as well. For brevity, we do not provide experimental results and just report that the usage of Algorithm 1 has greatly improved the performance of the *nomore++* system. This improvement is of course of relative nature and does not indicate any superiority of the approach.

Finally, we have provided an algorithm that exploits the properties of elementary subgraphs to extract active elementary loops from unfounded sets. This algorithm, which is the first of its kind, can be used by SAT-based ASP-solvers to replace terminating loops with active elementary loops.

Acknowledgments. This work was supported by DFG (SCHA 550/6-4). We are grateful to Martin Brain, Wolfgang Faber, Joohyung Lee, Yuliya Lierler, and the anonymous referees for many helpful suggestions.

References

Anger, C.; Gebser, M.; Linke, T.; Neumann, A.; and Schaub, T. 2005. The `nomore++` approach to answer set solving. In Sutcliffe, G., and Voronkov, A., eds., *LPAR*, 95–109. Springer-Verlag.

Apt, K.; Blair, H.; and Walker, A. 1987. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. Chapter 2, 89–148.

Calimeri, F.; Faber, W.; Leone, N.; and Pfeifer, G. 2001. Pruning operators for answer set programming systems. Report INFSYS RR-1843-01-07, TU Wien.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum. 293–322.

Faber, W. 2002. Enhancing efficiency and expressiveness in answer set programming systems. Dissertation, TU Wien.

Faber, W. 2006. Personal communication.

Fages, F. 1994. Consistency of Clark's completion and the existence of stable models. *J. MLCS* 1:51–60.

Fitting, M. 2002. Fixpoint semantics for logic programming: A survey. *TCS* 278(1-2):25–51.

Gebser, M., and Schaub, T. 2005. Loops: Relevant or redundant? In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *LPNMR*, 53–65. Springer-Verlag.

Gebser, M., and Schaub, T. 2006. Tableau calculi for answer set programming. In Dix, J., and Hunter, A., eds., *NMR*. This volume.

Gebser, M.; Lee, J.; and Lierler, Y. 2006. Elementary sets for logic programs. In Dix, J., and Hunter, A., eds., *NMR*. This volume.

Janhunen, T. 2003. Translatability and intranslatability results for certain classes of logic programs. Report A82, Helsinki UT.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In Kaelbling, L., and Saffiotti, A., eds., *IJCAI*, 503–508. Professional Book Center.

Leone, N.; Faber, W.; Pfeifer, G.; Eiter, T.; Gottlob, G.; Koch, C.; Mateis, C.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM TOCL*. To appear.

Leone, N.; Rullo, P.; and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Inf. Comput.* 135(2):69–112.

Lierler, Y., and Maratea, M. 2004. Cmodels-2: SAT-based answer sets solver enhanced to non-tight programs. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, 346–350. Springer-Verlag.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM TOCL*. To appear.

Lin, F., and Zhao, Y. 2002. ASSAT: computing answer sets of a logic program by SAT solvers. In *AAAI*, 112–118. AAAI/MIT Press.

Lin, F., and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 853–858. Morgan Kaufmann.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *AIJ* 157(1-2):115–137.

Linke, T., and Sarsakov, V. 2005. Suitable graphs for answer set programming. In Baader, F., and Voronkov, A., eds., *LPAR*, 154–168. Springer-Verlag.

Liu, L., and Truszczyński, M. 2005. Pbmodels - software to compute stable models by pseudoboolean solvers. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *LPNMR*, 410–415. Springer-Verlag.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *AIJ* 138(1-2):181–234.

Simons, P. 2000. Extending and implementing the stable model semantics. Dissertation, Helsinki UT.

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

1.7 Elementary Sets for Logic Programs

Elementary Sets for Logic Programs

Martin Gebser
Institut für Informatik
Universität Potsdam, Germany

Joohyung Lee
Computer Science and Engineering
Arizona State University, USA

Yuliya Lierler
Department of Computer Science
Universität Erlangen-Nürnberg, Germany

Abstract

By introducing the concepts of a loop and a loop formula, Lin and Zhao showed that the answer sets of a nondisjunctive logic program are exactly the models of its Clark's completion that satisfy the loop formulas of all loops. Recently, Gebser and Schaub showed that the Lin-Zhao theorem remains correct even if we restrict loop formulas to a special class of loops called "elementary loops." In this paper, we simplify and generalize the notion of an elementary loop, and clarify its role. We propose the notion of an elementary set, which is almost equivalent to the notion of an elementary loop for nondisjunctive programs, but is simpler, and, unlike elementary loops, can be extended to disjunctive programs without producing unintuitive results. We show that the maximal unfounded elementary sets for the "relevant" part of a program are exactly the minimal sets among the nonempty unfounded sets. We also present a graph-theoretic characterization of elementary sets for nondisjunctive programs, which is simpler than the one proposed in (Gebser & Schaub 2005). Unlike the case of nondisjunctive programs, we show that the problem of deciding an elementary set is coNP-complete for disjunctive programs.

Introduction

By introducing the concepts of a loop and a loop formula, Lin and Zhao (2004) showed that the answer sets (a.k.a. stable models) of a nondisjunctive logic program are exactly the models of its Clark's completion (Clark 1978) that satisfy the loop formulas $LF(L)$ of all loops L for the program. This important result has shed new light on the relationship between answer sets and completion, and allowed us to compute answer sets using SAT solvers, which led to the design of answer set solvers ASSAT[1] (Lin & Zhao 2004) and CMODELS[2] (Giunchiglia, Lierler, & Maratea 2004).

The concepts of a loop and a loop formula were further clarified in (Lee 2005). By slightly modifying the definition of a loop, Lee observed that adding loop formulas can be viewed as a generalization of completion, which allows us to characterize the stability of a model in terms of loop formulas: A model is stable iff it satisfies the loop formulas of all loops. He also observed that the mapping LF, which turns loops into loop formulas, can be applied to arbitrary sets of atoms, not only to loops: Adding $LF(Y)$ for a non-loop Y does not affect the models of the theory because $LF(Y)$ is always entailed by $LF(L)$ for some loop L. Though this reformulation of the Lin-Zhao theorem, in which LF is not restricted to loops, is less economical, it is interesting to note that it is essentially a theorem on assumption sets (Saccá & Zaniolo 1990), or unfounded sets (Van Gelder, Ross, & Schlipf 1991; Leone, Rullo, & Scarcello 1997) which has been known for many years. In this sense, the most original contribution of (Lin & Zhao 2004) was not the mapping that turns loops into loop formulas, but the definition of a loop, which yields a relatively small class of sets of atoms for the mapping LF.

However, for nondisjunctive programs, even the definition of a loop turned out still "too generous." Gebser and Schaub (2005) showed that restricting the mapping even more to a special class of loops called "elementary loops," yields a valid modification of the Lin-Zhao theorem (or the Saccá-Zaniolo theorem). That is, some loops are identified as redundant, just as all non-loops are redundant. They noted that the notion of a positive dependency graph, which is used for defining a loop, is not expressive enough to distinguish between elementary and non-elementary loops, and instead proposed another graph-theoretic characterization based on the notion of a so-called "body-head dependency graph."

Our work is motivated by the desire to understand the role of an elementary loop further and to extend the results to disjunctive programs. For nondisjunctive programs, we propose a simpler notion corresponding to an elementary loop, which we call an "elementary set," and provide a further enhancement of the Lin-Zhao theorem based on it. Unlike elementary loops, elementary sets can be extended to disjunctive programs without producing unintuitive results. We show that a special class of unfounded elementary sets coincides with the minimal sets among nonempty unfounded sets. Instead of relying on the notion of a body-head dependency graph, we present a simpler graph-theoretic characterization of elementary sets based on a subgraph of a positive dependency graph.

Elementary Sets for Nondisjunctive Programs

Review of Loop Formulas: Nondisjunctive Case

[1] http://assat.cs.ust.hk/
[2] http://www.cs.utexas.edu/users/tag/cmodels/

A *nondisjunctive rule* is an expression of the form

$$a_1 \leftarrow a_2, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n \qquad (1)$$

where $n \geq m \geq 1$ and $a_1, \ldots, a_n$ are propositional atoms. A *nondisjunctive program* is a finite set of nondisjunctive rules.

We will identify a nondisjunctive rule (1) with the propositional formula

$$(a_2 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n) \rightarrow a_1\ ,$$

and will often write (1) as

$$a_1 \leftarrow B, F \qquad (2)$$

where B is $a_2, \ldots, a_m$ and F is $\mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$. We will sometimes identify B with its corresponding set.

Let Π be a nondisjunctive program. The reduct Π^X of Π with respect to a set X of atoms is obtained from Π by

- deleting each rule (2) such that $X \not\models F$, and
- replacing each remaining rule (2) by $a_1 \leftarrow B$.

Set X is an *answer set (stable model)* of Π if it is minimal among the models that satisfy Π^X. [3]

The *(positive) dependency graph* of Π is the directed graph such that

- its vertices are the atoms occurring in Π, and
- its edges go from a_1 to $a_2, \ldots, a_m$ for all rules (1) of Π.

A nonempty set L of atoms is called a *loop* of Π if, for every pair p, q of atoms in L, there exists a path (possibly of length 0) from p to q in the dependency graph of Π such that all vertices in this path belong to L. In other words, L is a loop of Π iff the subgraph of the dependency graph of Π induced by L is strongly connected. Clearly, any set consisting of a single atom is a loop. For instance, Figure 1 shows the dependency graph of the following program Π_1:

$$\begin{array}{l} p \leftarrow \mathit{not}\ s \\ p \leftarrow r \\ q \leftarrow r \\ r \leftarrow p, q\ . \end{array}$$

Program Π_1 has seven loops: $\{p\}$, $\{q\}$, $\{r\}$, $\{s\}$, $\{p, r\}$, $\{q, r\}$, $\{p, q, r\}$.

For any set Y of atoms, the *external support formula* of Y for Π, denoted by $ES_\Pi(Y)$, is the disjunction of conjunctions

$$B \wedge F$$

for all rules (2) of Π such that

- $a_1 \in Y$, and
- $B \cap Y = \emptyset$.

The first condition expresses that the atom "supported" by (2) is an element of Y. The second condition ensures that this support is "external": The atoms in B that it relies on do not belong to Y. Thus Y is called *externally supported* by Π w.r.t. a set X of atoms if X satisfies $ES_\Pi(Y)$.

[3] We identify an interpretation with the set of atoms that are true in it.

Figure 1: The dependency graph of Program Π_1

For any set Y of atoms, by $LF_\Pi(Y)$ we denote the following formula:

$$\bigwedge_{a \in Y} a \rightarrow ES_\Pi(Y)\ . \qquad (3)$$

Formula (3) is called the *(conjunctive) loop formula* of Y for Π. [4] Note that we still call (3) a loop formula even when Y is not a loop.

The following reformulation of the Lin-Zhao theorem, which characterizes the stability of a model in terms of loop formulas, is a part of the main theorem from (Lee 2005) for the nondisjunctive case.

Theorem 1 *(Lee 2005) Let Π be a nondisjunctive program, and X a set of atoms occurring in Π. If X satisfies Π, then the following conditions are equivalent:*

(a) X is stable;

(b) X satisfies $LF_\Pi(Y)$ for all nonempty sets Y of atoms that occur in Π;

(c) X satisfies $LF_\Pi(Y)$ for all loops Y of Π.

According to the equivalence between conditions (a) and (b) in Theorem 1, a model of Π_1 is stable iff it satisfies the loop formulas of all fifteen nonempty sets of atoms occurring in Π_1. On the other hand, condition (c) tells us that it is sufficient to restrict attention to the following seven loop formulas:

$$\begin{array}{rcl} p & \rightarrow & \neg s \vee r \\ q & \rightarrow & r \\ r & \rightarrow & p \wedge q \\ s & \rightarrow & \bot \\ p \wedge r & \rightarrow & \neg s \\ q \wedge r & \rightarrow & \bot \\ p \wedge q \wedge r & \rightarrow & \neg s\ . \end{array} \qquad (4)$$

Program Π_1 has six models: $\{p\}$, $\{s\}$, $\{p, s\}$, $\{q, s\}$, $\{p, q, r\}$, and $\{p, q, r, s\}$. Among them, $\{p\}$ is the only stable model, which is also the only model that satisfies all loop formulas (4). In the next section, we will see that in fact the last loop formula can be disregarded as well, if we take elementary sets into account.

As noted in (Lee 2005), the equivalence between conditions (a) and (c) is a reformulation of the Lin-Zhao theorem; the equivalence between conditions (a) and (b) is a reformulation of Corollary 2 of (Saccá & Zaniolo 1990), and Theorem 4.6 of (Leone, Rullo, & Scarcello 1997) (for the nondisjunctive case), which characterizes the stability of a model in terms of *unfounded sets*. For sets X, Y of atoms, we say that Y is *unfounded* by Π w.r.t. X if Y is not externally supported by Π w.r.t. X. Condition (b) can be stated in terms of unfounded sets as follows:

[4] If the conjunction in the antecedent is replaced with the disjunction, the formula is called *disjunctive loop formula* (Lin & Zhao 2004). Our results stated in terms of conjunctive loop formulas can be stated in terms of disjunctive loop formulas as well.

(b′) X contains no nonempty unfounded subsets for Π w.r.t. X.

Elementary Sets for Nondisjunctive Programs

As mentioned in the introduction, (Gebser & Schaub 2005) showed that *LF* in Theorem 1 can be further restricted to "elementary loops." In this section, we present a simpler reformulation of their results. We will compare our reformulation with the original definition from (Gebser & Schaub 2005) later in this paper.

The following proposition tells us that a loop can be defined even without referring to a dependency graph.

Proposition 1 *For any nondisjunctive program Π and any nonempty set Y of atoms occurring in Π, Y is a loop of Π iff, for every nonempty proper subset Z of Y, there is a rule (2) in Π such that*

- $a_1 \in Z$, *and*
- $B \cap (Y \setminus Z) \neq \emptyset$.

For any set Y of atoms and any subset Z of Y, we say that Z is *outbound* in Y for Π if there is a rule (2) in Π such that

- $a_1 \in Z$,
- $B \cap (Y \setminus Z) \neq \emptyset$, and
- $B \cap Z = \emptyset$.

Let Π be a nondisjunctive program. For any nonempty set Y of atoms that occur in Π, we say that Y is *elementary* for Π if all nonempty proper subsets of Y are outbound in Y for Π. As with loops, it is clear from the definition that every set consisting of a single atom occurring in Π is elementary for Π. It is also clear that every elementary set for Π is a loop of Π, but a loop is not necessarily an elementary set: The conditions for being an elementary set are stronger than the conditions for being a loop as given in Proposition 1. For instance, one can check that for Π_1, $\{p, q, r\}$ is not elementary since $\{p, r\}$ (or $\{q, r\}$) is not outbound in $\{p, q, r\}$. All the other loops of Π_1 are elementary. Note that an elementary set may be a proper subset of another elementary set (both $\{p\}$ and $\{p, r\}$ are elementary sets for Π_1).

The following program replaces the last rule of Π_1 by two rules:

$$\begin{aligned} p &\leftarrow \mathit{not}\ s \\ p &\leftarrow r \\ q &\leftarrow r \\ r &\leftarrow p \\ r &\leftarrow q\ . \end{aligned}$$

This program has the same dependency graph as program Π_1 and thus has the same set of loops. However, its elementary sets are different: All its loops are elementary.

From the definition of an elementary set above, we get an alternative, equivalent definition by requiring that only the loops contained in Y be outbound, instead of requiring that all nonempty proper subsets of Y be outbound.

Proposition 2 *For any nondisjunctive program Π and any nonempty set Y of atoms that occur in Π, Y is an elementary set for Π iff all loops Z of Π such that $Z \subset Y$ are outbound in Y for Π.*[5]

Note that a subset of an elementary set, even if that subset is a loop, is not necessarily elementary. For instance, for program

$$\begin{aligned} p &\leftarrow p, q \\ q &\leftarrow p, q \\ p &\leftarrow r \\ q &\leftarrow r \\ r &\leftarrow p \\ r &\leftarrow q\ , \end{aligned}$$

set $\{p, q, r\}$ is elementary, but $\{p, q\}$ is not.

The following proposition describes a relationship between loop formulas of elementary sets and those of arbitrary sets.

Proposition 3 *Let Π be a nondisjunctive program, X a set of atoms, and Y a nonempty set of atoms that occur in Π. If X satisfies $LF_\Pi(Z)$ for all elementary sets Z of Π such that $Z \subseteq Y$, then X satisfies $LF_\Pi(Y)$.*

Proposition 3 suggests that condition (c) of Theorem 1 can be further enhanced by taking only loop formulas of elementary sets into account. This yields the following theorem, which is a reformulation of Theorem 3 from (Gebser & Schaub 2005) in terms of elementary sets.

Theorem 1(d) *The following condition is equivalent to conditions (a)–(c) of Theorem 1.*

(d) X satisfies $LF_\Pi(Y)$ for all elementary sets Y of Π.

According to Theorem 1(d), a model of Π_1 is stable iff it satisfies the first six formulas in (4); the loop formula of non-elementary set $\{p, q, r\}$ (the last one in (4)) can be disregarded.

Maximal Elementary Sets and Elementarily Unfounded Sets for Nondisjunctive Programs

If we modify condition (c) of Theorem 1 by replacing "loops" in its statement with "maximal loops," the condition becomes weaker, and the modified statement of Theorem 1 does not hold. For instance, program Π_1 has only two maximal loops, $\{p, q, r\}$ and $\{s\}$, and their loop formulas are satisfied by the non-stable model $\{p, q, r\}$. In fact, maximal loop $\{p, q, r\}$ is not even an elementary set for Π_1.

This is also the case with maximal elementary sets: Theorem 1(d) does not hold if "elementary sets" in its statement is replaced with "maximal elementary sets" as the following program shows:

$$\begin{aligned} p &\leftarrow q, \mathit{not}\ p \\ q &\leftarrow p, \mathit{not}\ p \\ p\ . \end{aligned} \tag{5}$$

Program (5) has two models, $\{p\}$ and $\{p, q\}$, but the latter is not stable. Yet, both models satisfy the loop formula of the only maximal elementary set $\{p, q\}$ for (5) ($p \wedge q \rightarrow \top$).

However, in the following we show that if we consider the "relevant" part of the program w.r.t. a given interpretation, it is sufficient to restrict attention to maximal elementary sets.

[5] Note that Proposition 2 remains correct even after replacing "all loops" in its statement with "all elementary sets."

Given a nondisjunctive program Π and a set X of atoms, by Π_X we denote the set of rules (2) of Π such that $X \models B, F$. The following proposition tells us that all nonempty proper subset of an elementary set for Π_X are externally supported w.r.t. X.

Proposition 4 *For any nondisjunctive program Π, any set X of atoms, and any elementary set Y for Π_X, X satisfies $ES_\Pi(Z)$ for all nonempty proper subsets Z of Y.*

From Proposition 4, it follows that every unfounded elementary set Y for Π_X w.r.t. X is maximal among the elementary sets for Π_X. One can show that if Y is a nonempty unfounded set for Π w.r.t. X that does not contain a maximal elementary set for Π_X, then Y consists of atoms that do not occur in Π_X. From this, we obtain the following result.

Theorem 1(e) *The following condition is equivalent to conditions (a)–(c) of Theorem 1.*

(e) X satisfies $LF_\Pi(Y)$ for every set Y of atoms such that

- *Y is a maximal elementary set for Π_X, or*
- *Y is a singleton whose atom occurs in Π.*

According to Theorem 1(e), model $\{p, q, r\}$ of Π_1 is not stable because it does not satisfy the loop formula of $\{q, r\}$, which is one of the maximal elementary sets for $(\Pi_1)_{\{p,q,r\}} = \Pi_1$.

Note that the analogy does not apply to loops: If we replace "maximal elementary sets" in the statement of Theorem 1(e) with "maximal loops," then the modified statement does not hold. The non-stable model $\{p, q, r\}$ still satisfies the loop formula of the maximal loop $\{p, q, r\}$ of $(\Pi_1)_{\{p,q,r\}}$ (the last one in (4)).

We say that a set Y of atoms occurring in Π is *elementarily unfounded* by Π w.r.t. X if

- Y is an elementary set for Π_X that is unfounded by Π w.r.t. X, or
- Y is a singleton that is unfounded by Π w.r.t. X.[6]

From Proposition 4, every non-singleton elementarily unfounded set for Π w.r.t. X is a maximal elementary set for Π_X.

It is clear from the definition that every elementarily unfounded set for Π w.r.t. X is an elementary set for Π and that it is also an unfounded set for Π w.r.t. X. However, a set that is both elementary for Π and unfounded by Π w.r.t. X is not necessarily an elementarily unfounded set for Π w.r.t. X. For example, consider the following program:

$$\begin{array}{l} p \leftarrow q, \mathit{not}\ r \\ q \leftarrow p, \mathit{not}\ r\ . \end{array} \qquad (6)$$

Set $\{p, q\}$ is both elementary for (6), and unfounded by (6) w.r.t. $\{p, q, r\}$, but it is not an elementarily unfounded set w.r.t. $\{p, q, r\}$.

The following corollary, which follows from Proposition 4, tells us that all nonempty proper subsets of an elementarily unfounded set are externally supported. It is essentially a reformulation of Theorem 5 from (Gebser & Schaub 2005).

[6] Elementarily unfounded sets are closely related to "active elementary loops" in (Gebser & Schaub 2005).

Corollary 1 *Let Π be a nondisjunctive program, X a set of atoms, and Y an elementarily unfounded set for Π w.r.t. X. Then*

- *X does not satisfy $ES_\Pi(Y)$, and*
- *X satisfies $ES_\Pi(Z)$ for all nonempty proper subsets Z of Y.*

Corollary 1 tells us that elementarily unfounded sets form an "anti-chain": One of them cannot be a proper subset of another.[7] In combination with Proposition 4, this tells us that elementarily unfounded sets are minimal among nonempty unfounded sets. Interestingly, the converse also holds.

Proposition 5 *For any nondisjunctive program Π and any sets X, Y of atoms, Y is an elementarily unfounded set for Π w.r.t. X iff Y is minimal among the nonempty sets of atoms occurring in Π that are unfounded by Π w.r.t. X.*

Theorem 1(e) can be stated in terms of elementarily unfounded sets, thereby restricting attention to minimal unfounded sets:

(e′) X contains no elementarily unfounded subsets for Π w.r.t. X.

The notion of an elementarily unfounded set may help improve computation performed by SAT-based answer set solvers. Since there are exponentially many loops in the worst case, SAT-based answer set solvers do not add all loop formulas at once. Instead, they check whether a model returned by a SAT solver is an answer set. If not, a loop formula that is not satisfied by the current model is added, and the SAT solver is invoked again.[8] This process is repeated until an answer set is found, or the search space is exhausted. In view of condition (e′), when loop formulas need to be added, it is sufficient to add loop formulas of elementarily unfounded sets only. This guarantees that loop formulas considered are only those of elementary sets. Since every elementary set is a loop, but not vice versa, the process may involve fewer loop formulas overall than the case when arbitrary loops are considered. In view of Proposition 3 and Corollary 1, this would yield reasonably the most economical way to eliminate non-stable models.

Deciding Elementary Sets: Nondisjunctive Case

The above definition of an elementary set involves all its nonempty proper subsets (or at least all loops that are its subsets). This seems to imply that deciding whether a set is elementary is a computationally hard problem. But in fact, Gebser and Schaub (2005) showed that, for nondisjunctive programs, deciding an elementary loop can be done efficiently. They noted that positive dependency graphs are not expressive enough to distinguish between elementary and non-elementary loops, and instead introduced so-called "body-head dependency graphs" to identify elementary loops. In this section, we simplify this result by still

[7] Recall that the anti-chain property does not hold for elementary sets for Π: An elementary set may contain another elementary set as its proper subset.

[8] To be precise, CMODELS adds "conflict clauses."

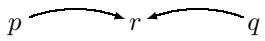


Figure 2: The elementary subgraph of $\{p, q, r\}$ for Π_1

referring to positive dependency graphs. We show that removing some "unnecessary" edges from the dependency graph is just enough to distinguish elementary sets from non-elementary sets.

For any nondisjunctive program Π and any set Y of atoms,

$EC^0_\Pi(Y) = \emptyset$,
$EC^{i+1}_\Pi(Y) = EC^i_\Pi(Y) \cup \{(a_1, b) \mid$ there is a rule (2) in Π such that $b \in B$ and the graph $(Y, EC^i_\Pi(Y))$ has a strongly connected subgraph containing all atoms in $B \cap Y\}$,
$EC_\Pi(Y) = \bigcup_{i \geq 0} EC^i_\Pi(Y)$.

Note that this is a "bottom-up" construction. We call the graph $(Y, EC_\Pi(Y))$ the *elementary subgraph* of Y for Π. It is clear that an elementary subgraph is a subgraph of a dependency graph and that it is not necessarily the same as the subgraph of the dependency graph induced by Y. Figure 2 shows the elementary subgraph of $\{p, q, r\}$ for Π_1, which is not strongly connected.

The following theorem is similar to Theorem 10 from (Gebser & Schaub 2005), but instead of referring to the notion of a body-head dependency graph, it refers to an elementary subgraph as defined above.

Theorem 2 *For any nondisjunctive program Π and any set Y of atoms occurring in Π, Y is an elementary set for Π iff the elementary subgraph of Y for Π is strongly connected.*

Clearly, constructing an elementary subgraph and checking whether it is strongly connected can be done in polynomial time. Therefore, the problem of deciding whether a given set of atoms is elementary is tractable.

Elementary Sets for Disjunctive Programs

Review of Loop Formulas: Disjunctive Case

A *disjunctive rule* is an expression of the form

$$a_1; \dots; a_k \leftarrow a_{k+1}, \dots, a_l, \mathit{not}\ a_{l+1}, \dots, \mathit{not}\ a_m, \mathit{not\ not}\ a_{m+1}, \dots, \mathit{not\ not}\ a_n \quad (7)$$

where $n \geq m \geq l \geq k \geq 0$ and $a_1, \dots, a_n$ are propositional atoms. A *disjunctive program* is a finite set of disjunctive rules.

We will identify a disjunctive rule (7) with the propositional formula

$$(a_{k+1} \wedge \dots \wedge a_l \wedge \neg a_{l+1} \wedge \dots \wedge \neg a_m \wedge \neg\neg a_{m+1} \wedge \dots \wedge \neg\neg a_n) \rightarrow (a_1 \vee \dots \vee a_k) ,$$

and will often write (7) as

$$A \leftarrow B, F \quad (8)$$

where A is $a_1, \dots, a_k$, B is $a_{k+1}, \dots, a_l$, and F is

$$\mathit{not}\ a_{l+1}, \dots, \mathit{not}\ a_m, \mathit{not\ not}\ a_{m+1}, \dots, \mathit{not\ not}\ a_n .$$

We will sometimes identify A and B with their corresponding sets.

Let Π be a disjunctive program. The reduct Π^X of Π with respect to a set X of atoms is obtained from Π by

- deleting each rule (8) such that $X \not\models F$, and
- replacing each remaining rule (8) by $A \leftarrow B$.

Similarly as with a nondisjunctive program, a set X of atoms is an *answer set (stable model)* of Π if X is minimal among the models that satisfy Π^X.

The definition of a dependency graph is extended to a disjunctive program in a straightforward way: The vertices of the graph are the atoms occurring in the program, and its edges go from the elements of A to the elements of B for all rules (8) of the program. The definition of a loop in terms of the dependency graph remains the same as in the case of nondisjunctive programs.

For any set Y of atoms, the *external support formula* of Y for Π, denoted by $ES_\Pi(Y)$, is the disjunction of conjunctions

$$B \wedge F \wedge \bigwedge_{a \in A \setminus Y} \neg a$$

for all rules (8) of Π such that

- $A \cap Y \neq \emptyset$, and
- $B \cap Y = \emptyset$.

When Π is nondisjunctive, this definition reduces to the definition of ES_Π for nondisjunctive programs given earlier.

The notion of LF_Π and the term *(conjunctive) loop formula* similarly apply to formulas (3) when Π is a disjunctive program. As shown in (Lee 2005), Theorem 1 remains correct after replacing "nondisjunctive program" in its statement with "disjunctive program."

Elementary Sets for Disjunctive Programs

In this section, we generalize the definition of an elementary set to disjunctive programs.

Note that a loop of a disjunctive program can be also defined without referring to a dependency graph: Proposition 1 remains correct after replacing "nondisjunctive" in its statement with "disjunctive," "(2)" with "(8)," and "$a_1 \in Z$" with "$A \cap Z \neq \emptyset$."

Let Π be a disjunctive program. For any set Y of atoms, we say that a subset Z of Y is *outbound* in Y for Π if there is a rule (8) in Π such that

- $A \cap Z \neq \emptyset$,
- $B \cap (Y \setminus Z) \neq \emptyset$,
- $A \cap (Y \setminus Z) = \emptyset$, and
- $B \cap Z = \emptyset$.

Note that when Π is nondisjunctive, this definition reduces to the corresponding definition given before.

As with nondisjunctive programs, for any nonempty set Y of atoms that occur in Π, we say that Y is *elementary* for Π if all nonempty proper subsets of Y are outbound in Y for Π. Similarly, every set consisting of a single atom occurring in Π is an elementary set for Π, and every elementary set

for Π is a loop of Π. The definition of an elementary set for a disjunctive program is stronger than the alternative definition of a loop provided in Proposition 1 for the disjunctive case: It requires that the rules satisfy two additional conditions, $A \cap (Y \setminus Z) = \emptyset$ and $B \cap Z = \emptyset$.

With these extended definitions, Propositions 2 and 3 remain correct after replacing "nondisjunctive program" in their statements with "disjunctive program." Theorem 1(d) holds even when Π is disjunctive.

To illustrate the definition, consider the following program:

$$\begin{array}{rcl} p\,;\ q & \leftarrow & p \\ p & \leftarrow & q \\ p & \leftarrow & \mathit{not}\ r\ . \end{array}$$

Among the four loops of the program, $\{p\}$, $\{q\}$, $\{r\}$, and $\{p, q\}$, the last one is not an elementary set because $\{q\}$ is not outbound in $\{p, q\}$: The first rule contains q in the head and p in the body, but it also contains $\{p, q\} \cap (\{p, q\} \setminus \{q\}) = \{p\}$ in the head. According to the extension of Theorem 1(d) to disjunctive programs, the loop formula of $\{p, q\}$ can be disregarded.

Maximal Elementary Sets and Elementarily Unfounded Sets for Disjunctive Programs

Let Π be a disjunctive program. For any sets X, Y of atoms, by $\Pi_{X,Y}$ we denote the set of all rules (8) of Π such that $X \models B, F$ and $X \cap (A \setminus Y) = \emptyset$. Program $\Pi_{X,Y}$ contains all rules of Π that can provide supports for Y w.r.t. X.

The following proposition tells us how $\Pi_{X,Y}$ is related to Π_X when Π is nondisjunctive.

Proposition 6 *Let Π be a nondisjunctive program, and X a set of atoms, and Y a set of atoms such that every element a_1 in Y has a rule (2) in Π such that $X \models B, F$. Then Y is elementary for $\Pi_{X,Y}$ iff it is elementary for Π_X.*

It follows from the proposition that for any non-singleton set Y of atoms, Y is elementary for $\Pi_{X,Y}$ iff it is elementary for Π_X.

We extend the definition of an elementarily unfounded set to disjunctive programs by replacing "Π_X" with "$\Pi_{X,Y}$" and by identifying Π as a disjunctive program. It is clear from the definition that every elementarily unfounded set for Π w.r.t. X is an elementary set for Π and that it is also an unfounded set for Π w.r.t. X.

Propositions 4, 5, Corollary 1, and Theorems 1(e), 1(e$'$) remain correct after replacing "nondisjunctive program" in their statements with "disjunctive program" and "Π_X" with "$\Pi_{X,Y}$." For preserving the intended meaning of Theorem 1(e), "Y is a maximal elementary set for Π_X" can be alternatively replaced with "Y is maximal among all sets Z of atoms that are elementary for $\Pi_{X,Z}$."

Deciding Elementary Sets: Disjunctive Case

Although deciding an elementary set can be done efficiently for nondisjunctive programs, it turns out that the corresponding problem for (arbitrary) disjunctive programs is intractable.

Proposition 7 *For any disjunctive program Π and any set Y of atoms, deciding whether Y is elementary for Π is* coNP*-complete.*

This result can be explained by the close relationship to the problem of deciding whether a set of atoms is *unfounded-free* (Leone, Rullo, & Scarcello 1997), which means that the set contains no nonempty unfounded subsets. In fact, the reduction from deciding unfounded-freeness to deciding elementariness is straightforward.

However, for the class of disjunctive programs called "head-cycle-free" (Ben-Eliyahu & Dechter 1994), deciding an elementary set is tractable. A disjunctive program Π is called *head-cycle-free* if, for every rule (8) in Π, there is no loop L of Π such that $|A \cap L| > 1$. For instance, the program

$$\begin{array}{l} p\,;q \leftarrow \\ p \leftarrow q \end{array}$$

is head-cycle-free, while the program

$$\begin{array}{l} p\,;q \leftarrow \\ p \leftarrow q \\ q \leftarrow p \end{array}$$

is not.

The definition of an elementary subgraph for a nondisjunctive program can be extended to a head-cycle-free program by replacing "(2)" with "(8)" and "$b \in B$" with "$a_1 \in A,\ b \in B$" in the equation for EC^{i+1}_{Π}. With this extended definition of an elementary subgraph, Theorem 2 remains correct after replacing "nondisjunctive program" in its statement with "head-cycle-free program."

Comparison with the Gebser-Schaub Definition

In this section, we compare our reformulation of elementary loops with the original definition given in (Gebser & Schaub 2005) for nondisjunctive programs.

Let Π be a nondisjunctive program. A loop of Π is called *trivial* if it consists of a single atom such that the dependency graph of Π does not contain an edge from the atom to itself. Non-trivial loops were called simply loops in (Lin & Zhao 2004; Gebser & Schaub 2005). For a non-trivial loop L of Π, let

$$R^-_\Pi(L) = \{(2) \in \Pi \mid a_1 \in L,\ B \cap L = \emptyset\},$$

$$R^+_\Pi(L) = \{(2) \in \Pi \mid a_1 \in L,\ B \cap L \neq \emptyset\}.$$

Definition 1 *(Gebser & Schaub 2005, Definition 1) Given a nondisjunctive program Π and a non-trivial loop L of Π, L is called a GS-elementary loop for Π if, for each non-trivial loop L' of Π such that $L' \subset L$, $R^-_\Pi(L') \cap R^+_\Pi(L) \neq \emptyset$.*[9]

Proposition 8 *For any nondisjunctive program Π and any non-trivial loop L of Π, L is a GS-elementary loop for Π iff L is an elementary set for Π.*

[9] A GS-elementary loop was called an "elementary loop" in (Gebser & Schaub 2005). Here we put "GS-" in the name, to distinguish it from a loop that is elementary under our definition.

There are a few differences between Definition 1 and our definition of an elementary set. First, the definition of an elementary set does not assume a priori that the set is a loop. Rather, the fact that an elementary set is a loop is a consequence of our definition. Second, our definition is simpler because it does not refer to a dependency graph. Third, the two definitions do not agree on trivial loops: A trivial loop is an elementary set, but not a GS-elementary loop. This originates from the difference between the definition of a loop given in (Lin & Zhao 2004) and its reformulation given in (Lee 2005). As shown in the main theorem of (Lee 2005), identifying a trivial loop as a loop provides a simpler reformulation of the Lin-Zhao theorem by omitting reference to completion. Furthermore, in the case of elementary sets, this reformulation also enables us to see a close relationship between maximal elementary sets (elementarily unfounded sets) and minimal nonempty unfounded sets. It also allows us to extend the notion of an elementary set to disjunctive programs without producing unintuitive results, unlike with GS-elementary loops. To see this, consider the following program:

$$\begin{array}{l} p\ ;\ q \leftarrow r \\ p\ ;\ r \leftarrow q \\ q\ ;\ r \leftarrow p\ . \end{array} \tag{9}$$

The non-trivial loops of this program are $\{p, q\}$, $\{p, r\}$, $\{q, r\}$, and $\{p, q, r\}$, but not singletons $\{p\}$, $\{q\}$, and $\{r\}$. If we were to extend GS-elementary loops to disjunctive programs, a reasonable extension would say that $\{p, q, r\}$ is a GS-elementary loop for program (9) because all its non-trivial proper subloops are "outbound" in $\{p, q, r\}$. Note that $\{p, q, r\}$ is unfounded w.r.t. $\{p, q, r\}$. Moreover, every singleton is unfounded w.r.t $\{p, q, r\}$ as well. This is in contrast with our Proposition 4, according to which all nonempty proper subsets of an elementary set for program (9) w.r.t. $\{p, q, r\}$ are externally supported w.r.t. $\{p, q, r\}$. This anomaly does not arise with our definition of an elementary set since $\{p, q, r\}$ is not elementary for (9). More generally, an elementary set is potentially elementarily unfounded w.r.t. some model, which is not the case with GS-elementary loops extended to disjunctive programs.

Conclusion

We have proposed the notion of an elementary set and provided a further refinement of the Lin-Zhao theorem based on it, which simplifies the Gebser-Schaub theorem and extends it to disjunctive programs.

We have shown properties of elementary sets that allow us to disregard redundant loop formulas. One property is that, if all elementary subsets of a given set of atoms are externally supported, the set is externally supported as well. Another property is that, for a maximal set that is elementary for the relevant part of the program w.r.t. some interpretation, all its nonempty proper subsets are externally supported w.r.t. the same interpretation. Related to this, we have proposed the concept of elementarily unfounded sets, which turn out to be precisely the minimal sets among nonempty unfounded sets.

Unlike elementary loops proposed in (Gebser & Schaub 2005), elementary sets and the related results are extended to disjunctive programs in a straightforward way. For nondisjunctive and head-cycle-free programs, we have provided a graph-theoretic characterization of elementary sets, which is simpler than the one proposed in (Gebser & Schaub 2005). For disjunctive programs, we have shown that deciding elementariness is coNP-complete, which can be explained by the close relationship to deciding unfounded-freeness of a given interpretation.

Elementary sets allow us to identify relevant unfounded sets more precisely than what loops allow. An apparent application is to consider elementarily unfounded sets in place of arbitrary unfounded loops as considered in the current SAT-based answer set solvers, at least for the tractable cases. For nondisjunctive programs, an efficient algorithm for computing elementarily unfounded sets is described in (Anger, Gebser, & Schaub 2006), which can be extended to head-cycle-free programs as well. Based on the theoretical foundations provided in this paper, we plan to integrate elementarily unfounded set computation into CMODELS for an empirical evaluation.

Acknowledgments

We are grateful to Selim Erdoğan, Vladimir Lifschitz, Torsten Schaub, and anonymous referees for their useful comments. Martin Gebser was supported by DFG under grant SCHA 550/6-4, TP C. Yuliya Lierler was partially supported by the National Science Foundation under Grant IIS-0412907.

References

Anger, C.; Gebser, M.; and Schaub, T. 2006. Approaching the core of unfounded sets. In *Proc. NMR 2006.*

Ben-Eliyahu, R., and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence* 12(1-2):53–87.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum Press. 293–322.

Gebser, M., and Schaub, T. 2005. Loops: Relevant or redundant? In *Proc. LPNMR 2005*, 53–65.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. SAT-based answer set programming. In *Proc. AAAI 2004*, 61–66.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proc. IJCAI 2005*, 503–508.

Leone, N.; Rullo, P.; and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation* 135(2):69–112.

Lin, F., and Zhao, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1–2):115–137.

Saccá, D., and Zaniolo, C. 1990. Stable models and nondeterminism in logic programs with negation. In *Proceed-*

ings of ACM Symposium on Principles of Database Systems (PODS), 205–217.

Van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of ACM* 38(3):620–650.

1.8 Debugging inconsistent answer set programs

Debugging Inconsistent Answer Set Programs

Tommi Syrjänen*
Helsinki University of Technology, Dept. of Computer Science and Eng.,
Laboratory for Theoretical Computer Science,
P.O.Box 5400, FIN-02015 HUT, Finland
Tommi.Syrjanen@tkk.fi

Abstract

In this paper we examine how we can find contradictions from Answer Set Programs (ASP). One of the most important phases of programming is debugging, finding errors that have crept in during program implementation. Current ASP systems are still mostly experimental tools and their support for debugging is limited. This paper addresses one part of ASP debugging, finding the reason why a program does not have any answer sets at all. The basic idea is to compute diagnoses that are minimal sets of constraints whose removal returns consistency. We compute also conflict sets that are sets of mutually incompatible constraints. The final possible source of inconsistency in an ASP program comes from odd negative loops and we show how these may also be detected. We have created a prototype for the ASP debugger that is itself implemented using ASP.

Introduction

One of the most important phases in computer programming is always debugging; no matter how much care is used in program writing, some errors will creep in. For this reason a practical Answer Set Programming (ASP) system should have support for program debugging. It is not possible to detect all errors automatically since a construct may be an error in one case but correct code in another.

The current ASP systems (Niemelä, Simons, & Syrjänen 2000; Dell'Armi *et al.* 2001; East & Truszczyński 2001; Anger, Konczak, & Linke 2001; Babovich 2002; Lin & Zhao 2002) are still on experimental level and their support for debugging is limited. In this paper we examine how we can debug one class of program errors, namely finding the contradictions in a program. We have developed a prototype debugger implementation for the SMODELS input language but the same principles are applicable for most ASP systems.

Program defects can be roughly divided into two classes (Aho, Sethi, & Ullman 1986):

- *syntax errors*: the program does not conform with the formal syntax of the language; and
- *semantic errors*: the program is syntactically correct but does not behave as the programmer intended.

*This research has been funded by the Academy of Finland (project number 211025).

In this discussion we leave out syntactical errors since they are generally easy to remedy: the ASP system notes that the program is not valid and outputs an error message telling where the problem occurred.

The semantical errors are more difficult to handle. In the context of ASP, they too can be roughly divided into two classes:

- *typographical errors* such as misspelling predicate or variable names, using a constant in place of a variable or vice versa; and
- *logical errors* where a rule behaves differently from what was intended.

The intuition of the division is that an error is typographical if it is caused by a simple misspelling of a single syntactical element. For example, using *corect*(X) instead of *correct*(X). On the other hand, a logical error is one where the programmer writes a rule that does not do what he or she expects it to do. For example, a programmer writing an encoding for a planning problem might want to state the constraint that an object may be at one place at a time by using the rule:

$$\leftarrow at(O, L_1, I), at(O, L_2, I).$$

The problem is that the values of L_1 and L_2 are not constrained and may take the same value. Thus, for each object o, location x, and time step i, there will be a ground instance:

$$\leftarrow at(o, x, i), at(o, x, i).$$

which causes a contradiction no matter where the object is. In this case the programmer should have added a test $L_1 \neq L_2$ to the rule body.

Our experience is that finding the reason for a contradiction is one of the most laborious tasks in ASP debugging. Currently the most practical approach is to remove rules from the program until the resulting program has an answer set and then examining the removed rules to see what caused the error.

In this paper we examine how we can automate this process using ASP meta-programming. When we have a contradictory program, we create several new ASP programs based on it such that their answer sets reveal the possible places of error.

We borrow our basic idea from the model-based diagnosis (Reiter 1987) field. There we have a system that does not behave like it should and a diagnosis is a set of components whose failure explains the symptoms. In our approach a *diagnosis* is a set of rules whose removal returns consistency to the program. However, we do not attempt construct a standard diagnostic framework. The reason for this is pragmatic: our aim is to create a practical tool that helps answer set programmers to debug their programs. It is not reasonable to expect that a programmer would have an existing system description that could be analyzed since that would in effect be a correct program. On the other hand, we are not willing to leave the debugger completely without of formal semantics. One of the strengths of ASP is that all programs have declarative semantics so it seems natural that also their diagnoses have one. Thus, we construct our own formal framework that shares some features with model-based diagnosis but is different in other areas.

When we construct diagnoses, we are interested in minimal ones. There are several possible ways to define minimality and we will use *cardinality minimality*: a diagnosis is minimal if there is no diagnosis that contains fewer rules than it. Another possibility would be *subset minimality* where a diagnosis is minimal if it does not contain another diagnosis as its subset. We chose cardinality minimality mainly because it was easier to implement in the prototype and also because it is possible that smaller diagnoses are easier to handle in practical debugging.

Not all minimal diagnoses are equally good for debugging purposes. For example, consider the program:

$$\{a\}\ . \tag{1}$$
$$b \leftarrow a. \tag{2}$$
$$c \leftarrow \text{not } a. \tag{3}$$
$$\leftarrow 1\ \{b, c\}\ . \tag{4}$$

Here (1) says that a may be true or false, (2) tells that b is true if a is true, (3) that c is true if a is not, and finally (4) is a constraint stating that it is an error if either b or c is true.

No matter what truth value we choose for a, either b or c is true, so we have a contradiction. The minimum number of rules that we have to remove to repair consistency is one: removing either (2), (3), or (4) results in a consistent program. Removing (4) gives the most information to the programmer since neither $b \leftarrow a$ nor $c \leftarrow \text{not } a$ can cause the contradiction by themselves. On the other hand, (4) is a constraint telling that its body should not become true so the connection to the contradiction is immediate.

We take the approach that we include only constraints in minimal diagnoses. Examining just them is not enough since a contradiction can arise also from an *odd loop*. An odd loop is a program fragment where an atom depends recursively on itself through an odd number of negations. The simplest example is:

$$a \leftarrow \text{not } a.$$

This rule causes a contradiction since if a is set to false, we have to conclude that a is true. On the other hand, if a is set to true, the body of the rule is not satisfied so we do not have a justification for a and we have to set it false.

Not all odd loops are errors since they may be used to prune out unwanted answer sets. Since it is difficult to determine which odd loops are intentional and which are errors, we take the approach that all odd loops are considered to be errors.

This means that the programmer has to use some other construct to replace the odd loops. In SMODELS the alternative approach is to first generate the possible model candidates using *choice rules* of the form:

$$\{head\} \leftarrow body.$$

Here the intuition is that if *body* is true, then *head* may be true but it may be also false. The pruning is then done using *constraints* of the form:

$$\leftarrow body.$$

A constraint asserts that the *body* must be false. Note that a constraint is actually an odd loop in a disguise: we could replace a constraint by the equivalent rule:

$$f \leftarrow body, \text{not } f.$$

In general, a program may have a number of different minimal diagnoses. In many cases some constraints occurring in them are related to each other. For example, in program:

$$\{a\}\ .$$
$$\leftarrow a. \tag{1}$$
$$\leftarrow \text{not } a. \tag{2}$$
$$\leftarrow \text{not } b. \tag{3}$$

there are two different diagnoses: $\{1, 3\}$ and $\{2, 3\}$. Here the constraints (1) and (2) both depend on the value of a. If a is chosen to be true, then (1) fails, if not, (2) fails. In effect, we can have either (1) or (2) in the program, but not both. The constraint (3) is independent from the other two and it always fails.

A *conflict set* is a way of formalizing the concept of related constraints. The intuition is that a set of constraints is a conflict set if every diagnosis of the program contains exactly one member from the set.[1] We use the conflict sets to give more information to the programmer. In the above program the two conflict sets are $\{2, 3\}$ and $\{4\}$. In general, if two rules belong in the same conflict set, the truth values of the literals that occur in their bodies depend on same truth values of same atoms: choosing one value leads to one contradiction and choosing the other leads to another. Grouping them together may lead the programmer to the place of error faster.

Note that there are programs whose constraints cannot be divided into conflict sets. In those cases we cannot use conflict sets to help debugging and have to use other methods. Fortunately, those cases seem to be quite rare in practice.

[1] Note that conflict sets are different from conflicts. In model-based diagnosis a conflict is a set of components that contains at least one malfunctioning component.

Related Work

Brain et. al. (Brain, Watson, & De Vos 2005) presented an interactive way for computing answer sets. A programmer can use the interactive system as a debugging aid since it can be used to explain why a given atom is either included in an answer set or left out from it. Their approach is very similar to our method of computing explanations for diagnoses.

The NoMoRe system (Anger, Konczak, & Linke 2001) utilizes blocking graphs that can be used to examine why a given rule is applied or blocked and thus they provide a visual method for debugging ASP programs.

The consistency-restoring rules of Balduccini and Gelfond (Balduccini & Gelfond 2003) are another related approach. They define a method that allows a reasoning system to find the best explanation for conflicting observations. The main difference between our approaches is that we do not try to fix the contradictory program but instead try to help the programmer to find the places that are in error.

There has been a lot of previous work on the properties of odd and even cycles in a program (for example, (You & Yuan 1994; Lin & Zhao 2004; Costantini & Provetti 2005; Constantini 2005)) and how they affect the existence and number of answer sets. In this work we propose methodology where even loops are replaced by choice rules and odd loops by constraints, so our viewpoint is slightly different. However, the theoretical results of previous work still hold since our programs could be translated back to normal logic programs. In particular, constraints are equivalent to one-rule odd loops.

The most closely related area of odd loop research is Constatini's work on static program analysis (Constantini 2005). She notes that there are two different ways to escape the inconsistency caused by an odd loop: either there has to be one unsatisfied literal in the body of at least one rule of the loop or there has to be a non-circular justification for some atom in the loop. The literals that are present in rule bodies but are not part of the loop are called AND-handles and the extra rules are OR-handles. In every answer set of the program there has to be an applicable handle for every odd loop in it. Since the handles are purely syntactic properties, we can statically analyze the rules to see what conditions have to be met so that all loops are satisfied. This approach seems promising but there is currently the limitation that the definitions demand that the program is in kernel normal form. This is not an essential limitation from theoretical point of view since every normal logic program can be systematically translated to the normal form, but it will cause an extra step in practical debugger since the results have to be translated back to the original program code.

Language

In this paper we construct a debugger for a subset of SMODELS language.[2] We will consider only finite ground programs that do not have cardinality constraint literals but that may have choice rules.

[2]The actual debugger implementation handles the complete language.

The basic building block of a program is an *atom* that encodes a single proposition that may be either true or false. A *literal* is either an atom a or its negation $\mathrm{not}\ a$.

A *basic rule* is of the form:

$$h \leftarrow l_1, \ldots, l_n$$

where the *head* h is an atom and l_1, ..., l_n in the *body* are literals. The intuition is that if all literals l_1, ..., l_n are true, then h has to be also true. If the body is empty ($n = 0$), then the rule is a *fact*. A *choice rule* has the form:

$$\{h\} \leftarrow l_1, \ldots, l_n$$

where h and l_i are defined as above. The intuition of a choice rule is that if the body is true, then the head may be true but it may also be false. If an atom does not occur in the head of any rule that has a satisfied body, it has to be false.

Basic and choice rules are together called *generating* rules. The other possibility is a *constraint* that is a rule without a head. If the body of a constraint becomes true, then the model candidate is rejected. A *logic program* $P = \langle \mathcal{G}, \mathcal{C} \rangle$ is a pair where $\mathcal{G}$ is a finite set of generating rules and $\mathcal{C}$ a finite set of constraints.

Before we can define the formal ASP semantics for these programs, we need to define notation that allows us to refer to the parts of a rule. Let $r = h \leftarrow a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m$ be a basic rule where a_i and b_i are atoms. Then,

$$\begin{aligned} \mathrm{head}(r) &= h \\ \mathrm{body}^+(r) &= \{a_1, \ldots, a_n\} \\ \mathrm{body}^-(r) &= \{b_1, \ldots, b_m\} \ . \end{aligned}$$

The same notation is used for choice rules. We use $\mathrm{Atoms}(P)$ to denote the set of atoms that occur in a program P.

A set of atoms S *satisfies* an atom a (denoted by $S \vDash a$) iff $a \in S$ and a negative literal $\mathrm{not}\ a$ iff $a \notin S$. A set S satisfies a set of literals L iff $\forall l \in L : S \vDash l$. A set S satisfies constraint $\leftarrow l_1, \ldots, l_n$ iff $S \nvDash l_i$ for some $1 \leq i \leq n$.

The ASP semantics is defined using the concept of a *reduct* (Gelfond & Lifschitz 1988). The reduct P^S of a program $P = \langle \mathcal{G}, \mathcal{C} \rangle$ with respect of a set of atoms S is:

$$\begin{aligned} P^S &= \langle \mathcal{G}^S, \mathcal{C} \rangle, \text{ where} \\ \mathcal{G}^S &= \{\mathrm{head}(r) \leftarrow \mathrm{body}^+(r) \mid r \in \mathcal{G}, S \vDash \mathrm{body}^-(r), \\ &\qquad \text{and } r \text{ is either a basic rule or a} \\ &\qquad \text{choice rule and } \mathrm{head}(r) \in S\} \ . \end{aligned}$$

Note that all rules that belong to the generator part of a reduct P are basic rules and all literals that occur in them are positive. Such rules are monotonic so $\mathcal{G}^S$ has a unique least model (Gelfond & Lifschitz 1988) that we denote with $\mathbf{MM}(\mathcal{G}^S)$. If this least model happens to coincide with S and it also satisfies all constraints, then S is an answer set of P.

Definition 1 *Let $P = \langle \mathcal{G}, \mathcal{C} \rangle$ be a program. A set of ground atoms S is an* answer set *of P if and only if:*

1. $\mathbf{MM}(\mathcal{G}^S) = S$; *and*
2. $\forall r \in \mathcal{C} : S \models r$.

A program P is *consistent* if it has at least one answer set and *inconsistent* if it has none.

Theory for Debugging

Odd Loops

Definition 2 *The* dependency graph $DG_P = \langle V, E^+, E^- \rangle$ *of a program* $P = \langle \mathcal{G}, \mathcal{C} \rangle$ *is a triple where* $V = Atoms(P)$ *and* $E^+, E^- \subseteq V \times V$ *are sets of* positive *and* negative *edges such that:*

$$E^+ = \{\langle h, a \rangle \mid \exists r \in \mathcal{G} : head(r) = h \text{ and } a \in body^+(r)\}$$
$$E^- = \{\langle h, b \rangle \mid \exists r \in \mathcal{G} : head(r) = h \text{ and } b \in body^-(r)\} \ .$$

Definition 3 *Let* $DG_P = \langle V, E^+, E^- \rangle$ *be a dependency graph. Then the two* dependency relations Odd_P *and* Even_P *are the smallest relations on* V *such that:*

1. *for all* $\langle a_1, a_2 \rangle \in E^-$ *it holds that* $\langle a_1, a_2 \rangle \in \text{Odd}_P$;
2. *for all* $\langle a_1, a_2 \rangle \in E^+$ *it holds that* $\langle a_1, a_2 \rangle \in \text{Even}_P$;
3. *if* $\langle a_1, a_2 \rangle \in E^-$ *and* $\langle a_2, a_3 \rangle \in \text{Even}_P$, *then* $\langle a_1, a_3 \rangle \in \text{Odd}_P$;
4. *if* $\langle a_1, a_2 \rangle \in E^-$ *and* $\langle a_2, a_3 \rangle \in \text{Odd}_P$, *then* $\langle a_1, a_3 \rangle \in \text{Even}_P$;
5. *if* $\langle a_1, a_2 \rangle \in E^+$ *and* $\langle a_2, a_3 \rangle \in \text{Even}_P$, *then* $\langle a_1, a_3 \rangle \in \text{Even}_P$; *and*
6. *if* $\langle a_1, a_2 \rangle \in E^+$ *and* $\langle a_2, a_3 \rangle \in \text{Odd}_P$, *then* $\langle a_1, a_3 \rangle \in \text{Odd}_P$.

The reason for the interleaved definition is that the relations Odd and Even are then easy to compute: we start by initializing them with the edges of the dependency graph, and then compute the transitive closure of the graph where every negative edge changes the parity of the dependency: if b depends on c evenly and there is a negative edge from a to b, then a depends oddly on c.

Definition 4 *Let* P *be a program. Then, an* odd loop *is a set* $L = \{a_1, \ldots, a_n\}$ *of atoms such that* $\langle a_i, a_j \rangle \in \text{Odd}_P$ *for all* $1 \leq i, j \leq n$. *An atom* $a \in Atoms(P)$ occurs in an odd loop *iff* $\langle a, a \rangle \in \text{Odd}_P$. *The program* P *is* odd loop free *if* $\forall a \in Atoms(P) : \langle a, a \rangle \notin \text{Odd}_P$.

Diagnoses and Conflict Sets

Definition 5 *Let* $P = \langle \mathcal{G}, \mathcal{C} \rangle$ *be an odd loop free program. Then, a* diagnosis *of* P *is a set* $D \subseteq \mathcal{C}$ *such that the program* $\langle \mathcal{G}, \mathcal{C} \setminus D \rangle$ *is consistent. A diagnosis is* minimal *iff for all diagnoses* D' *of* P *it holds that* $|D'| \geq |D|$. *The set of all minimal diagnoses of* P *is denoted by* $\mathfrak{D}(P)$.

Example 1 *Consider the program:*

$$\{a\}\ .$$
$$\leftarrow a. \quad (1)$$
$$\leftarrow \text{not } a. \quad (2)$$
$$\leftarrow \text{not } b. \quad (3)$$

This program has two minimal diagnoses: $D_1 = \{1, 3\}$ *and* $D_2 = \{2, 3\}$. *To see that* D_1 *is really a diagnosis, note that when its rules are removed, we are left with:*

$$\{a\}.$$
$$\leftarrow \text{not } a.$$

that has the answer set $\{a\}$.

We can observe two properties of diagnoses from Definition 5. First, if P is consistent, then it has a unique minimal diagnosis that is the empty set. The second observation is that every inconsistent program has at least one minimal diagnosis.

Theorem 1 *Let* $P = \langle \mathcal{G}, \mathcal{C} \rangle$ *be an inconsistent odd loop free program. Then there exists at least one minimal diagnosis* D *for it.*

Proof 1 *The rules in* $\mathcal{G}$ *can be systematically translated into an equivalent normal logic program* $\mathcal{G}'$ *where every choice rule is replaced by an even loop (see (Niemelä & Simons 2000) for details). Since* $\mathcal{G}'$ *is odd loop free, it is consistent (You & Yuan 1994). Thus, the set* $D' = \mathcal{C}$ *is a diagnosis. Since* $\mathcal{C}$ *is finite, there has to exist at least one minimal diagnosis* $D \subseteq D'$.

Definition 6 *Let* $P = \langle \mathcal{G}, \mathcal{C} \rangle$ *be a program and* $\mathfrak{D}(P)$ *the set of its minimal diagnoses. Then, a* conflict set $C \subseteq \mathcal{C}$ *is a set of constraints such that:*

1. *for all diagnoses* $D \in \mathfrak{D}(P)$ *it holds that* $|D \cap C| = 1$; *and*
2. *for all constraints* $r \in C$ *there exists a diagnosis* $D \in \mathfrak{D}(P)$ *such that* $r \in D$.

The set of all conflict sets of P *is denoted by* $\mathfrak{C}(P)$.

Intuitively, constraints that belong in a conflict set are mutually exclusive in the sense that it is impossible to have all of them satisfied at the same time. Note that with this definition it is possible that a program does not have any conflict sets at all.

Example 2 *In Example 1 we had two diagnoses* $D_1 = \{1, 3\}$ *and* $D_2 = \{2, 3\}$. *We can partition the constraints that occur in them into two conflict sets:*

$$C_1 = \{1, 2\}$$
$$C_2 = \{3\} \ .$$

Example 3 *The program:*

$$\{a\}. \qquad \leftarrow \text{not } a.\ (1) \qquad \leftarrow a, b.\ (4)$$
$$\{b\}. \qquad \leftarrow \text{not } b.\ (2) \qquad \leftarrow b, c.\ (5)$$
$$\{c\}. \qquad \leftarrow \text{not } c.\ (3) \qquad \leftarrow a, c.\ (6)$$

has six minimal diagnoses: $\{1, 2\}$, $\{1, 3\}$, $\{1, 5\}$, $\{2, 3\}$, $\{2, 6\}$, *and* $\{3, 4\}$. *We see that there is no way to partition the constraints so that every diagnosis contains exactly one rule for each set.*

The ASP Programs

In this section we create three different ASP programs that can be used to debug contradictory programs. We express these programs using the full SMODELS syntax so we need to introduce a few new constructs. We do not give here the full formal semantics but the interested reader may consult (Syrjänen 2004) for details.

A *cardinality constraint literal* is of the form $L\ \{l_1, \ldots, l_n\}\ U$ where L and U are integral *lower* and *upper bounds* and l_i are literals. A cardinality constraint literal is true if the number of satisfied literals l_i is between U and L, inclusive. Next, a *conditional literal* has the form $a(X)\ :\ d(X)$ This construct denotes the set of literals $\{a(t)\ |\ d(t) \text{ is true}\}$. Finally, a fact may have a *numeric range* in it and $a(1..n)$ denotes the set of n facts $\{a(1), \ldots, a(n)\}$.

Odd Loop Detection

When we do the odd loop detection, we will use the standard meta-programming encoding of logic programs (Sterling & Shapiro 1994). A rule:

$$r = h \leftarrow a, \text{not } b$$

is encoded using the facts:

$$\begin{array}{ll} \mathit{rule}(r). & \mathit{pos\text{-}body}(r, a). \\ \mathit{head}(r, h). & \mathit{neg\text{-}body}(r, b). \end{array}$$

We start the odd loop program by extracting the atoms from the program representation:

$$\begin{aligned} \mathit{atom}(H) &\leftarrow \mathit{head}(R, H). \\ \mathit{atom}(A) &\leftarrow \mathit{pos\text{-}body}(R, A). \\ \mathit{atom}(B) &\leftarrow \mathit{neg\text{-}body}(R, B). \end{aligned}$$

Next, we construct the dependency graph for the program:

$$\begin{aligned} \mathit{pos\text{-}edge}(H, A) &\leftarrow \mathit{head}(R, H), \\ &\qquad \mathit{pos\text{-}body}(R, A). \\ \mathit{neg\text{-}edge}(H, B) &\leftarrow \mathit{head}(R, H), \\ &\qquad \mathit{neg\text{-}body}(R, B). \end{aligned}$$

One step positive dependency is even, negative odd:

$$\begin{aligned} \mathit{even}(X, Y) &\leftarrow \mathit{pos\text{-}edge}(X, Y). \\ \mathit{odd}(X, Y) &\leftarrow \mathit{neg\text{-}edge}(X, Y). \end{aligned}$$

Adding a new positive edge preserves parity:

$$\begin{aligned} \mathit{even}(X, Z) &\leftarrow \mathit{pos\text{-}edge}(X, Y), \mathit{even}(Y, Z), \mathit{atom}(Z). \\ \mathit{odd}(X, Z) &\leftarrow \mathit{pos\text{-}edge}(X, Y), \mathit{odd}(Y, Z), \mathit{atom}(Z). \end{aligned}$$

Adding a negative edge flips parity:

$$\begin{aligned} \mathit{odd}(X, Z) &\leftarrow \mathit{neg\text{-}edge}(X, Y), \mathit{even}(Y, Z), \mathit{atom}(Z). \\ \mathit{even}(X, Z) &\leftarrow \mathit{neg\text{-}edge}(X, Y), \mathit{odd}(Y, Z), \mathit{atom}(Z). \end{aligned}$$

There is an odd loop if a predicate depends oddly on itself:

$$\mathit{odd\text{-}loop}(X) \leftarrow \mathit{odd}(X, X).$$

Two atoms X and Y are in same odd loop if X depends oddly on Y and Y depends evenly on X:

$$\mathit{in\text{-}odd\text{-}loop}(X, Y) \leftarrow \mathit{odd}(X, Y), \mathit{even}(Y, X).$$

The above rules correspond directly to the Definitions 1–4. We could stop here, but we can make debugging a bit easier if we also identify which rules belong to which loops. We start by choosing one of the atoms that occur in a loop to act as an identifier for the loop. We take the atom that is lexicographically the first one:

$$\begin{aligned} \mathit{first\text{-}in\text{-}loop}(A) &\leftarrow \mathit{odd\text{-}loop}(A), \text{not } \mathit{has\text{-}predecessor}(A). \\ \mathit{has\text{-}predecessor}(A) &\leftarrow \mathit{in\text{-}odd\text{-}loop}(B, A), B < A. \end{aligned}$$

The final part of the odd loop detection is to compute which rules belong to the loop. The idea is that if X and Y are in the same loop, then a rule that has X in the head and Y in the body participates in the loop. We also have to extract the identifier of the particular loop.

$$\begin{aligned} \mathit{rule\text{-}in\text{-}loop}(R, Z) \leftarrow\ & \mathit{in\text{-}odd\text{-}loop}(X, Y), \\ & \mathit{in\text{-}odd\text{-}loop}(X, Z), \\ & \mathit{first\text{-}in\text{-}loop}(Z), \\ & \mathit{head}(R, X), \\ & \mathit{pos\text{-}body}(R, Y). \\ \mathit{rule\text{-}in\text{-}loop}(R, Z) \leftarrow\ & \mathit{in\text{-}odd\text{-}loop}(X, Y), \\ & \mathit{in\text{-}odd\text{-}loop}(X, Z), \\ & \mathit{first\text{-}in\text{-}loop}(Z), \\ & \mathit{head}(R, X), \\ & \mathit{neg\text{-}body}(R, Y). \end{aligned}$$

Example 4 *Consider the program:*

$$a \leftarrow \text{not } b. \quad (1)$$
$$b \leftarrow a. \quad (2)$$

This program is expressed with facts:

$$\begin{array}{ll} \mathit{head}(1, a). & \mathit{neg\text{-}body}(1, b). \\ \mathit{head}(2, b). & \mathit{pos\text{-}body}(2, a). \end{array}$$

When these facts are given as an input for the odd loop detection program, we have a unique answer set. The relevant atoms from it are:

$$\begin{aligned} S = \{&\mathit{odd\text{-}loop}(a), \mathit{odd\text{-}loop}(b), \mathit{first\text{-}in\text{-}loop}(a), \\ &\mathit{rule\text{-}in\text{-}loop}(2, a), \mathit{rule\text{-}in\text{-}loop}(1, a)\}\ . \end{aligned}$$

This answer set tells that the rules (1) and (2) form an odd loop whose identifier is a.

Finding Diagnoses

We could use the meta-representation of the previous section for also diagnosis computation but it is more efficient in practice to use a more direct translation. The basic idea is that we add a new literal to the bodies of constraint to control whether it is applied or not. For example, a constraint:

$$r = \leftarrow a, \text{not } b.$$

is translated into two rules:

$$\leftarrow \text{not } \textit{removed}(r), a, \text{not } b.$$
$$\textit{constraint}(r).$$

All generating rules are kept as they were. Next, we add the rule:

$$\{\textit{removed}(R) : \textit{constraint}(R)\}\ n.$$

This rule asserts that at most n of the constraints may be removed. Here n is a numeric constant whose value is set from the command line.

The actual diagnoses are then computed by first setting the n to zero and then increasing the value until the translated program has an answer set. The diagnosis can then be extracted by noting the *removed*$/1$ atoms that are true in the answer.

Example 5 *The program from program from Example 1 is translated into:*

$$\{a\}.$$
$$\leftarrow \text{not } \textit{removed}(1), a.$$
$$\leftarrow \text{not } \textit{removed}(2), \text{not } a.$$
$$\leftarrow \text{not } \textit{removed}(3), \text{not } b.$$
$$\textit{constraint}(1..3).$$
$$\{\textit{removed}(R) : \textit{constraint}(R)\}\ n.$$

When we start computing the answer sets for the transformed program we note that there are no answer sets when $n = 0$ and $n = 1$. With $n = 2$ there are two answer sets:

$$S_1 = \{\textit{removed}(1), \textit{removed}(3), a\}$$
$$S_2 = \{\textit{removed}(2), \textit{removed}(3)\}$$

The two diagnoses can then be read directly from S_1 and S_3.

Finding Conflict Sets

Once we have computed all diagnoses, we can check whether the program has conflict sets. We use a fact

$$\textit{in-diagnosis}(d, r).$$

to denote that the constraint r is in the dth diagnosis.

First, we initialize several type predicates:

$$\textit{conflict-set}(1..s).$$
$$\textit{diagnosis}(S) \leftarrow \textit{in-diagnosis}(S, R).$$
$$\textit{rule}(R) \leftarrow \textit{in-diagnosis}(S, R).$$

Here s is again a constant that is set during the instantiation of the program.

We need two rules to compute the sets. The first one states that each rule belongs to exactly one conflict set, and the second states that every diagnosis should have exactly one rule in each conflict set:

$$1\ \{\textit{in-set}(S, R) : \textit{conflict-set}(S)\}\ 1 \leftarrow \textit{rule}(R).$$
$$1\ \{\textit{in-set}(S, R) : \textit{in-diagnosis}(X, R)\}\ 1 \leftarrow \textit{conflict-set}(S), \textit{diagnosis}(X).$$

The conflict sets are computed in a same way as the diagnoses: we start with only one conflict set, and increase their number until we either find a partition or we know that none exists.

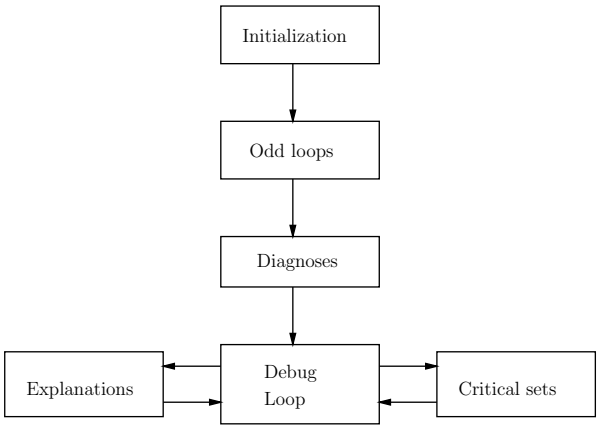


Figure 1: The debugger workflow

Example 6 *From Example 5 we get the facts:*

$$\textit{in-diagnosis}(1, 1). \qquad \textit{in-diagnosis}(1, 3).$$
$$\textit{in-diagnosis}(2, 2). \qquad \textit{in-diagnosis}(2, 3).$$

With these facts we find an answer set[3] *when $s = 2$. This answer set is:*

$$S = \{\textit{in-set}(1, 1), \textit{in-set}(1, 2), \textit{in-set}(2, 3)\}$$

corresponding to $\mathfrak{C}(P) = \{\{1, 2\}, \{3\}\}$.

Debugger Implementation

We have created a prototype implementation for the ASP debugger, `smdebug`, by combining the SMODELS programs with a driver program that is written with Perl. The debugger implementation is included within the *lparse* instantiator that is available for download at `http://www.tcs.tkk.fi/Software/smodels`.

The general system architecture of `smdebug` is shown in Figure 1. The debugger has four main components:

1. Odd loop detection. If the input program has an odd loop, `smdebug` issues an error message and terminates;
2. Diagnosis computation where `smdebug` calls SMODELS to compute all minimal diagnoses of the program;
3. Conflict set computation where `smdebug` tries to find conflict sets of the program; and
4. Explanation computation where `smdebug` computes derivation trees for constraints that occur in diagnoses.

We did not examine the fourth phase in this work but its idea is to give the programmer more detailed knowledge about the reasons of the contradictions. The user selects one diagnosis, and the debugger computes which set of choices leads to this particular contradiction and presents the information in the form of a derivation tree.

[3] More precisely, we have two isomorphic answer sets.

Conclusions and Further Work

In this work we applied the techniques from the symbolic diagnosis (Reiter 1987) field to ASP debugging. The main concepts have a natural mapping into ASP programs where a diagnosis is a set of constraints whose removal returns consistency to the program. We restrict these diagnoses to programs that are created in such a way that they do not have odd loops. We use another ASP program to find the odd loops that occur in a program and to warn about them. Finally, we defined the concept of the conflict set that can be used to check which constraints are mutually exclusive.

We have created a prototype implementation, `smdebug`, that implements the three debugging techniques of this paper for the full SMODELS input language. Additionally, `smdebug` also can compute derivation trees to act as explanations for the contradictions.

The main limitation for the current version of `smdebug` is that it can be used to find only contradictions. However, some of the techniques can be adapted to also explain why a given atom is or is not in an answer set. In particular, the method of computing explanations should generalize to this direction quite easily.

The next step in continuing with the `smdebug` development is to add support for handling non-contradictory programs. This means that we have to add support for computing and analyzing answer sets of the program.

There are several avenues of further research for improving the current system. The algorithm that `smdebug` uses for finding the minimal diagnoses and conflict sets is rather naive: iteratively increasing the size of the parameter until we get a program that has an answer set. It is possible that some other approach could get us equally useful results faster. Also, using some other minimality condition, like subset minimality, might give better results in some cases.

This debugger has not been used to debug large answer set programs, yet. The largest debugged program thus far has been the part of the debugger itself. One of its early versions of the explanation generation program contained a bug that caused it to be contradictory. The debugger not only identified the place of the error immediately, but it also uncovered two bugs in the *lparse* instantiator.

It may be that the current debugging support is not strong enough to handle really large programs. In those cases probably the best way to proceed is to try to manually find the smallest input program where the error happens and to debug that one.

In conclusion, this approach seems promising for ASP development but only time will tell if it will fulfill those promises.

References

Aho, A. V.; Sethi, R.; and Ullman, J. D. 1986. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company.

Anger, C.; Konczak, K.; and Linke, T. 2001. Nomore : A system for non-monotonic reasoning under answer set semantics. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, 406–410.

Babovich, Y. 2002. Cmodels, a system computing answer sets for tight logic programs.

Balduccini, M., and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *AAAI Spring 2003 Symposium*, 9–18.

Brain, M.; Watson, R.; and De Vos, M. 2005. An interactive approach to answer set programming. In *Answer Set Programming: Advances in Theory and Implementation ASP-05*, 190 – 202.

Constantini, S. 2005. Towards static analysis of answer set programs. Computer Science Group Technical Reports CS-2005-03, Dipartimento di Ingegneria, Universita' di Ferrara.

Costantini, S., and Provetti, A. 2005. Normal forms for answer sets programming. *TPLP* 5(6):747–760.

Dell'Armi, T.; Faber, W.; Ielpa, G.; Koch, C.; Leone, N.; Perri, S.; and Pfeifer, G. 2001. System description: Dlv. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*. Vienna, Austria: Springer-Verlag.

East, D., and Truszczyński, M. 2001. Propositional satisfiability in answer-set programming. In *Proceedings of KI 2001: Advances in Artificial Intelligence*, 138–153.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, 1070–1080. The MIT Press.

Lin, F., and Zhao, Y. 2002. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 112–118. Edmonton, Alberta, Canada: The AAAI Press.

Lin, F., and Zhao, Y. 2004. On odd and even cycles in normal logic programs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 80–85. The AAAI Press.

Niemelä, I., and Simons, P. 2000. Extending the smodels system with cardinality and weight constraints. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer. 491–521.

Niemelä, I.; Simons, P.; and Syrjänen, T. 2000. Smodels: A system for answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Sterling, L., and Shapiro, E. 1994. *The Art of Prolog*. MIT press.

Syrjänen, T. 2004. Cardinality constraint logic programs. In *The Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, 187–200. Lisbon, Portugal: Springer-Verlag.

You, J.-H., and Yuan, L. Y. 1994. A three-valued semantics for deductive database and logic programs t. *Journal of Computer and System Science* 49:334–361.

1.9 Forgetting and Conflict Resolving in Disjunctive Logic Programming

Forgetting and Conflict Resolving in Disjunctive Logic Programming*

Thomas Eiter
Technische Universität Wien, Austria
eiter@kr.tuwien.ac.at

Kewen Wang†
Griffith University, Australia
k.wang@griffith.edu.au

Abstract

We establish a declarative theory of forgetting for disjunctive logic programs. The suitability of this theory is justified by a number of desirable properties. In particular, one of our results shows that our notion of forgetting is completely captured by the classical forgetting. A transformation-based algorithm is also developed for computing the result of forgetting. We also provide an analysis of computational complexity. As an application of our approach, a fairly general framework for resolving conflicts in inconsistent knowledge bases represented by disjunctive logic programs is defined. The basic idea of our framework is to weaken the preferences of each agent by forgetting certain knowledge that causes inconsistency. In particular, we show how to use the notion of forgetting to provide an elegant solution for preference elicitation in disjunctive logic programming.

Introduction

Forgetting (Lin & Reiter 1994; Lang, Liberatore, & Marquis 2003) is a key issue for adequately handle a range of classical tasks such as query answering, planning, decision-making, reasoning about actions, or knowledge update and revision. It is, moreover, also important in recently emerging issues such as design and engineering of Web-based ontology languages. Suppose we start to design an ontology of *Pets*, which is a knowledge base of various pets (like cats, dogs but not lions or tigers). Currently, there are numerous ontologies on the Web. We navigated the Web and found an ontology *Animals* which is a large ontology on various animals including cats, dogs, tigers and lions. It is not a good idea to download the whole ontology Animals. The approach in the current Web ontology language standard OWL[1] is to discard those terminologies that are not desired (although this function is still very limited in OWL). For example, we may discard (or forget) tigers and lions from the ontology *Animals*. If our ontology is only a list of relations, we can handle the forgetting (or discarding) easily. However, an ontology is often represented as a logical theory, and the removal of one term may influence other terms in the ontology. Thus, more advanced methods are needed.

Disjunctive logic programming (DLP) under the answer set semantics (Gelfond & Lifschitz 1990) is now widely accepted as a major tool for knowledge representation and commonsense reasoning (Baral 2002). DLP is expressive in that it allows disjunction in rule heads, negation as failure in rule bodies and strong negation in both heads and bodies. Studying forgetting within DLP is thus a natural issue, and we make in this paper the following contributions:

- We establish a declarative, semantically defined notion of forgetting for disjunctive logic programs, which is a generalization of the corresponding notion for nondisjunctive programs proposed in (Wang, Sattar, & Su 2005). The suitability of this theory is justified by a number of desirable properties.
- We present a transformation-based algorithm for computing the result of forgetting. This method allows to obtain the result of forgetting a literal l in a logic program via a series of program transformations and other rewritings. In particular, for any disjunctive program P and any literal l, a syntactic representation $\mathsf{forget}(P, l)$ for forgetting l in P always exists. The transformation is novel and does not extend a previous one in (Wang, Sattar, & Su 2005), which as we show is incomplete.
- Connected with the transformation algorithm, we settle some complexity issues for reasoning under forgetting. They provide useful insight into feasible representations of forgetting.
- As an application of our approach, we present a fairly general framework for resolving conflicts in inconsistent knowledge bases. The basic idea of this framework is to weaken the preferences of each agent by forgetting certain knowledge that causes inconsistency. In particular, we show how to use the notion of forgetting to provide an elegant solution for preference elicitation in DLP.

Preliminaries

We briefly review some basic definitions and notation used throughout this paper.

A *disjunctive program* is a finite set of rules of the form

$$a_1 \vee \cdots \vee a_s \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n, \quad (1)$$

*This work was partially supported by the Austrian Science Funds (FWF) projects P17212 and 18019, the European Commission project REWERSE (IST-2003-506779) and the Australia Research Council (ARC) Discovery Project 0666107.

†Part of the work was done while this author was working at Technische Universität Wien.

[1]http://www.w3.org/2004/OWL/

$s, m, n \geq 0$, where a, b's and c's are classical literals in a propositional language. A *literal* is a *positive literal* p or a *negative literal* $\neg p$ for some atom p. An *NAF*-literal is of the form $not\ l$ where not is for the negation as failure and l is a (ordinary) literal. For an atom p, p and $\neg p$ are called *complementary*. For any literal l, its complementary literal is denoted $\tilde{l}$.

To guarantee the termination of some program transformations, the body of a rule is a set of literals rather than a multiset.

Given a rule r of form (1), $head(r) = a_1 \vee \cdots \vee a_s$ and $body(r) = body^+(r) \cup not\ \ body^-(r)$ where $body^+(r) = \{b_1, \ldots, b_m\}$, $body^-(r) = \{c_1, \ldots, c_n\}$, and $not\ body^-(r) = \{not\ q \mid q \in body^-(r)\}$.

A rule r of the form (1) is *normal* or *non-disjunctive*, if $s \leq 1$; *positive*, if $n = 0$; *negative*, if $m = 0$; *constraint*, if $s = 0$; *fact*, if $m = 0$ and $n = 0$, in particular, a rule with $s = n = m = 0$ is the constant *false*.

A disjunctive program P is called *normal program* (resp. *positive program*, *negative program*), if every rule in P is normal (resp. positive, negative).

Let P be a disjunctive program and let X be a set of literals. A disjunction $a_1 \vee \cdots \vee a_s$ is satisfied by X, denoted $X \models a_1 \vee \cdots \vee a_s$ if $a_i \in X$ for some i with $1 \leq i \leq s$. A rule r in P is satisfied by X, denoted $X \models r$, iff "$body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$ imply $X \models head(r)$". X is a model of P, denoted $X \models P$ if every rule of P is satisfied by X.

An *interpretation* X is a set of literals that contains no pair of complementary literals.

The answer set semantics The *reduct* of P on X is defined as $P^X = \{head(r) \leftarrow body^+(r) \mid r \in P, body^-(r) \cap X = \emptyset\}$. An interpretation X is an *answer set* of P if X is a minimal model of P^X (by treating each literal as a new atom). $\mathcal{AS}(P)$ denotes the collection of all answer sets of P. P is *consistent* if it has at least one answer set.

Two disjunctive programs P and P' are *equivalent*, denoted $P \equiv P'$, if $\mathcal{AS}(P) = \mathcal{AS}(P')$.

As usual, B_P is the *Herbrand base* of logic program P, that is, the set of all (ground) literals in P.

Forgetting in Logic Programming

In this section, we want to define what it means to forget about a literal l in a disjunctive program P. The idea is to obtain a logic program which is equivalent to the original disjunctive program, if we ignore the existence of the literal l. We believe that forgetting should go beyond syntactic removal of rules/literals and be close to classical forgetting and answer set semantics (keeping its spirit) at the same time. Thus, the definition of forgetting in this section is given in semantics terms, i.e., based on answer sets, and naturally generalizes the corresponding one in (Wang, Sattar, & Su 2005).

In propositional logic, the result of forgetting $\mathsf{forget}(T, p)$ about a proposition p in a theory T is conveniently defined as $T(p/true) \vee T(p/false)$. This way cannot be directly generalized to logic programming since there is no notion of the "disjunction" of two logic programs. However, if we examine the classical forgetting in model-theoretic point of view, we can obtain the models of $\mathsf{forget}(T, p)$ in this way: first compute all models of T and remove p from each model if it contains p. The resulting collection of sets $\{M \setminus \{p\} \mid M \models T\}$ is exactly the set of all models of $\mathsf{forget}(T, p)$.

Similarly, given a consistent disjunctive program P and a literal l, we naively could define the result of forgetting about l in P as an extended disjunctive program P' whose answer sets are exactly $\mathcal{AS}(P) \setminus l = \{X \setminus \{l\} \mid X \in \mathcal{AS}(P)\}$. However, this notion of forgetting cannot guarantee the existence of P' for even simple programs. For example, consider $P = \{a \leftarrow .\quad p \vee q \leftarrow\}$, then $\mathcal{AS}(P) = \{\{a, p\}, \{a, q\}\}$ and thus $\mathcal{AS}(P) \setminus p = \{\{a\}, \{a, q\}\}$. Since $\{a\} \subset \{a, q\}$ and, as well-known, answer sets are incomparable under set inclusion, $\mathcal{AS}(P) \setminus p$ cannot be the set of answer sets of any disjunctive program.

A solution to this problem is *a suitable notion of minimal answer set* such that the definition of answer sets, minimality, and forgetting can be fruitfully combined. To this end, we call a set X' an *l-subset of a set X*, denoted $X' \subseteq_l X$, if $X' \setminus \{l\} \subseteq X \setminus \{l\}$. Similarly, a set X' is a strict l-subset of X, denoted $X' \subset_l X$, if $X' \setminus \{l\} \subset X \setminus \{l\}$. Two sets X and X' of literals are l-equivalent, denoted $X \sim_l X'$, if $(X \setminus X') \cup (X' \setminus X) \subseteq \{l\}$.

Definition 1 *Let P be a consistent disjunctive program, let l be a literal in P and let X be a set of literals.*

1. *For a collection $\mathcal{S}$ of sets of literals, $X \in \mathcal{S}$ is* l-minimal *if there is no $X' \in \mathcal{S}$ such that $X' \subset_l X$. $\min_l(\mathcal{S})$ denotes the collection of all l-minimal elements in $\mathcal{S}$.*
2. *An answer set X of disjunctive program P is an* l-answer set *if X is l-minimal in $\mathcal{AS}(P)$. $\mathcal{AS}_l(P)$ consists of all l-answer sets of P.*

To make $\mathcal{AS}(P) \setminus l$ incomparable, we could take either minimal elements or maximal elements from $\mathcal{AS}(P) \setminus l$. However, selecting minimal answer sets is in line with semantic principles to minimize positive information.

For example, $P = \{a \leftarrow .\ p \vee q \leftarrow\}$, has two answer sets $X = \{a, p\}$ and $X' = \{a, q\}$. X is a p-answer set of P, but X' is not. This example shows that, for a disjunctive program P and a literal l, not every answer set is an l-answer set.

In the rest of this paper, we assume that P is a consistent program. The following proposition collects some easy properties of l-answer sets.

Proposition 1 *For any consistent program P and a literal l in P, the following four items are true:*

1. *An l-answer set X of P must be an answer set of P.*
2. *For any answer set X of P, there is an l-answer set X' of P such that $X' \subseteq_l X$.*
3. *Any answer set X of P with $l \in X$ is an l-answer set of P.*
4. *If an answer set X of P is not an l-answer set, then (1) $l \notin X$; (2) there exists an l-answer set Y of P such that $l \in Y \subset_l X$.*

Having the notion of minimality about forgetting a literal, we are now in a position to define the result of forgetting about a literal in a disjunctive program.

Definition 2 *Let P be a consistent disjunctive program and l be a literal. A disjunctive program P' is a result of* forgetting about l in P, *if P' represents l-answer sets of P, i.e., the following conditions are satisfied:*

1. $B_{P'} \subseteq B_P \setminus \{l\}$ *and*
2. *For any set X' of literals with $l \notin X'$, X' is an answer set of P' iff there is an l-answer set X of P such that $X' \sim_l X$.*

Notice that the first condition implies that l does not appear in P'. An important difference of the notion of forgetting here from existing approaches to updating and merging logic programs is that only l and possibly some other literals are removed. In particular, no new symbol is introduced in P'.

For a consistent extended program P and a literal l, some program P' as in the above definition always exists (cf. Algorithm 1 for details). However, different such programs P' might exist. It follows from the above definition that they are all equivalent under the answer set semantics.

Proposition 2 *Let P be a disjunctive program and l a literal in P. If P' and P'' are two results of forgetting about l in P, then P' and P'' are equivalent.*

We use $\mathsf{forget}(P, l)$ to denote a possible result of forgetting about l in P.

Example 1 *1. If $P_1 = \{q \leftarrow not\ p\}$, then $\mathsf{forget}(P_1, q) = \emptyset$ and $\mathsf{forget}(P_1, p) = \{q \leftarrow\}$.*

2. *If $P_2 = \{p \vee q \leftarrow\}$, then $\mathsf{forget}(P_2, p) = \emptyset$.*
3. *$P_3 = \{p \vee q \leftarrow not\ p.\ c \leftarrow q\}$ has the unique answer set $\{q, c\}$ and $\mathsf{forget}(P_3, p) = \{q \leftarrow .\ c \leftarrow\}$.*
4. *$P_4 = \{a \vee p \leftarrow not\ b.\ c \leftarrow not\ p.\ b \leftarrow\}$. Then $\mathsf{forget}(P_4, p) = \{c \leftarrow .\ b \leftarrow\}$.*

We will explain how to obtain $\mathsf{forget}(P, l)$ in the next section. The following proposition generalizes Proposition 2.

Proposition 3 *Let P and P' be two equivalent disjunctive programs and l a literal in P. Then $\mathsf{forget}(P, l)$ and $\mathsf{forget}(P', l)$ are also equivalent.*

However, forgetting here does not preserve some special equivalences of logic programs stronger than ordinary equivalence like strong equivalence (Lifschitz, Tang, & Turner 1999) or uniform equivalence (Eiter & Fink 2003). A notion of forgetting which preserves strong equivalence is interesting for some applications, but beyond the scope of this paper. In addition, our approach may be easily refined to preserve equivalences stronger than ordinary equivalences by a canonical form for the result of forgetting (e.g., the output of Algorithm 1).

Proposition 4 *For any consistent program P and a literal l in P, the following items are true:*

1. $\mathcal{AS}(\mathsf{forget}(P, l)) = \{X \setminus \{l\} \mid X \in \mathcal{AS}_l(X)\}$.
2. *If $X \in \mathcal{AS}_l(X)$ with $l \notin X$, then $X \in \mathcal{AS}(\mathsf{forget}(P, l))$.*
3. *For any $X \in \mathcal{AS}(P)$ such that $l \in X$, $X \setminus \{l\} \in \mathcal{AS}(\mathsf{forget}(P, l))$.*
4. *For any $X' \in \mathcal{AS}(\mathsf{forget}(P, l))$, either X' or $X' \cup \{l\}$ is in $\mathcal{AS}(P)$.*
5. *For any $X \in \mathcal{AS}(P)$, there exists $X' \in \mathcal{AS}(\mathsf{forget}(P, l))$ such that $X' \subseteq X$.*
6. *If l does not appear in P, then $\mathsf{forget}(P, l) = P$.*

Let $\models_s$ and $\models_c$ be the skeptical and credulous reasoning defined by the answer sets of a disjunctive program P, respectively: for any literal l,

$P \models_s l$ iff $l \in S$ for every $S \in \mathcal{AS}(P)$.
$P \models_c l$ iff $l \in S$ for some $S \in \mathcal{AS}(P)$.

Proposition 5 *Let l be a specified literal in disjunctive program P. For any literal $l' \neq l$,*

1. *$P \models_s l'$ iff $\mathsf{forget}(P, l) \models_s l'$.*
2. *$P \models_c l'$ if $\mathsf{forget}(P, l) \models_c l'$.*

This proposition says that, if l is ignored, $\mathsf{forget}(P, l)$ is equivalent to P under skeptical reasoning, but weaker under credulous reasoning (i.e., inferences are lost).

Similar to the case of normal programs, the above definitions of forgetting about a literal l can be extended to forgetting about a set F of literals. Specifically, we can similarly define $X_1 \subseteq_F X_2$, $X_1 \sim_F X_2$ and F-answer sets of a disjunctive program. The properties of forgetting about a single literal can also be generalized to the case of forgetting about *a set*. Moreover, the result of forgetting about a set F can be obtained one by one forgetting each literal in F.

Proposition 6 *Let P be a consistent disjunctive program and $F = \{l_1, \ldots, l_m\}$ be a set of literals. Then*

$$\mathsf{forget}(P, F) \equiv \mathsf{forget}(\mathsf{forget}(\mathsf{forget}(P, l_1), l_2), \ldots), l_m).$$

We remark that for removing a proposition p entirely from a program P, it is suggestive to remove both the literals p and $\neg p$ in P (i.e., all positive and negative information about p). This can be easily accomplished by $\mathsf{forget}(P, \{p, \neg p\})$.

Let $\mathsf{lcomp}(P)$ be Clark's completion plus the loop formulas for an ordinary disjunctive program P (Lee & Lifschitz 2003; Lin & Zhao 2004). Then X is an answer set of P iff X is a model of $\mathsf{lcomp}(P)$.

Now we have two kinds of operators $\mathsf{forget}(,)$ and $\mathsf{lcomp}()$. Thus for a disjunctive program and an atom p, we have two classical logical theories $\mathsf{lcomp}(\mathsf{forget}(P, p))$ and $\mathsf{forget}(\mathsf{lcomp}(P), p)$ on the signature $B_P \setminus \{p\}$. It is natural to ask what the relationship between these two theories is. Intuitively, the models of the first theory are all minimal models while the models of the second theory may not be minimal [2]. Let $P = \{p \leftarrow not\ q.\ q \leftarrow not\ p\}$. Then $\mathsf{lcomp}(\mathsf{forget}(P, p)) = \{\neg q\}$ and $\mathsf{forget}(\mathsf{lcomp}(P), p) = \{\mathsf{T} \leftrightarrow \neg q \vee \mathsf{F} \leftrightarrow \neg q\} \equiv \mathsf{T}$, which has two models $\{q\}$ and $\emptyset$.

However, we have the following result. However, notice that for this program P, the *minimal models* of $\mathsf{forget}(\mathsf{lcomp}(P), p)$ are the same as the *models* of $\mathsf{lcomp}(\mathsf{forget}(P, p))$. In fact, this result is true for ordinary disjunctive programs in general.

Theorem 1 *Let P be a logic program without strong negation and p an atom in P. Then X is an answer set of*

[2] Thanks to Esra Erdem and Paolo Ferraris for pointing this out to us.

$\mathsf{forget}(P,p)$ *if and only if* X *is a minimal model of the result of classical forgetting* $\mathsf{forget}(\mathsf{lcomp}(P),p)$. *That is,*

$$\mathcal{AS}(\mathsf{forget}(P,p)) = \mathsf{MMod}(\mathsf{forget}(\mathsf{lcomp}(P),p))$$

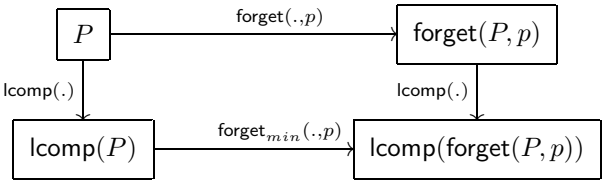
Here $\mathsf{MMod}(T)$ *denotes the set of all minimal models of a theory* T *in classical logic.*

This result means that the answer sets of $\mathsf{forget}(P,p)$ are exactly the minimal models of the result of forgetting about p in the classical theory $\mathsf{lcomp}(P)$. Thus $\mathsf{forget}(P,p)$ can be characterized by forgetting in classical logic. Notice that it would not make much sense if we replace $\mathsf{lcomp}(P)$ with a classical theory which is not equivalent to $\mathsf{lcomp}(P)$ in Theorem 1. In this sense, the notion of forgetting for answer set programming is unique.

We use $\mathsf{forget}_{min}(T,p)$ to denote a set of classical formulas whose models are the minimal models of the classical forgetting $\mathsf{forget}(T,p)$. Then the conclusion of Theorem 1 is reformulated as

$$\mathsf{lcomp}(\mathsf{forget}(P,p)) \equiv \mathsf{forget}_{min}(\mathsf{lcomp}(P),p).$$

This result is graphically represented in the following commutative diagram



The result is a nice property, since it means that one can "bypass" the use of an LP engine entirely, and represent also the answer sets of $\mathsf{forget}(P,p)$ in terms of a circumscription of classical forgetting, applied to $\mathsf{lcomp}(P)$. In fact, we can express combined forgetting and minimal model reasoning by a circumscription of $\mathsf{lcomp}(P)$.

Theorem 2 *Let* P *be a logic program without strong negation and* p *an atom in* P. *Then* S' *is an answer set of* $\mathsf{forget}(P,p)$ *if and only if either* $S = S'$ *or* $S = S' \cup \{p\}$ *is a model of* $\mathsf{Circ}(B_P \setminus \{p\}, \{p\}, \mathsf{lcomp}(P))$.

Computation of Forgetting

As we have noted, $\mathsf{forget}(P,l)$ exists for any consistent disjunctive program P and literal l. In this section, we discuss some issues on computing the result of forgetting.

Naive Algorithm

By Definition 2, we can easily obtain a naive algorithm for computing $\mathsf{forget}(P,l)$ using some ASP solvers for DLP, like DLV (Leone *et al.* 2004) or GnT (Janhunen *et al.* 2000).

Algorithm 1 (Computing a result of forgetting)
Input: disjunctive program P and a literal l in P.
Procedure:

Step 1. Using DLV compute $\mathcal{AS}(P)$;

Step 2. Remove the literal l from every element of $\mathcal{AS}(P)$ and denote the resulting collection as A'

Step 3. Obtain A'' by removing non-minimal elements from A'.

Step 4. Construct P' whose answer sets are exactly A'': Let $A'' = \{A_1, ..., A_m\}$ and for each A_i, $P_i = \{l' \leftarrow not\ \bar{A}_i \mid l' \in A_i\}$. $P' = \cup_{1 \le i \le n} P_i$. Here $\bar{A}_i = B_P \setminus A_i$.

Step 5. Output P' as $\mathsf{forget}(P,l)$.

This algorithm is complete w.r.t. the semantic forgetting defined in Definition 2.

Theorem 3 *For any consistent disjunctive program* P *and a literal* l, *Algorithm 1 always outputs* $\mathsf{forget}(P,l)$.

Basic Program Transformations

The above algorithm is semantic, and does not describe how to syntactically compute the result of forgetting in DLP. In this subsection, we develop an algorithm for computing the result of forgetting in P using program transformations and other modifications. Here we use the set $\mathbf{T}^*_{\mathsf{WFS}}$ of program transformations investigated in (Brass & Dix 1999; Wang & Zhou 2005). In our algorithm, an input program P is first translated into a negative program and the result of forgetting is represented as a nested program (under the minimal answer sets defined by Lifschitz et al. (1999)).

Elimination of Tautologies: P' is obtained from P by the elimination of tautologies if there is a rule r: $head(r) \leftarrow body^+(r), not\ \ body^-(r)$ in P such that $head(r) \cap body^+(r) \neq \emptyset$ and $P' = P \setminus \{r\}$.

Elimination of Head Redundancy P' is obtained from P by the elimination of head redundancy if there is a rule r in P such that an atom a is in both $head(r)$ and $body^-(r)$ and $P' = P \setminus \{r\} \cup \{head(r) - a \leftarrow body(r)\}$.

The above two transformations guarantee that those rules whose head and body have common literals are removed.

Positive Reduction: P' is obtained from P by positive reduction if there is a rule r: $head(r) \leftarrow body^+(r), not\ \ body^-(r)$ in P and $c \in body^-(r)$ such that $c \notin head(P)$ and P' is obtained from P by removing $not\ c$ from r. That is, $P' = P \setminus \{r\} \cup \{head(r) \leftarrow body^+(r), not\ (body^-(r) \setminus \{c\})\}$.

Negative Reduction: P' is obtained from P by negative reduction if there are two rules r: $head(r) \leftarrow body^+(r), not\ body^-(r)$ and r': $head(r') \leftarrow$ in P such that $head(r') \subseteq body^-(r)$ and $P' = P \setminus \{r\}$.

To define our next program transformation, we need the notion of *s-implication* of rules. This is a strengthened version of the notion of *implications* in (Brass & Dix 1999).

Definition 3 *Let* r *and* r' *be two rules. We say that* r' *is an s-implication of* r *if* $r' \neq r$ *and at least one of the following two conditions is satisfied:*

1. r' *is an implication of* r: $head(r) \subseteq head(r')$, $body(r) \subseteq body(r')$ *and at least one inclusion is proper; or*
2. r *can be obtained by changing some negative body literals of* r' *into head atoms and removing some head atoms and body literals from* r' *if necessary.*

Elimination of s-Implications: P_2 is obtained from P_1 by elimination of s-implications if there are two distinct rules r and r' of P_1 such that r' is an s-implication of r and $P_2 = P_1 \setminus \{r'\}$.

Unfolding: P' is obtained from P by unfolding if there is a rule r such that

$$\begin{aligned} P' \quad = \quad & P \setminus \{r\} \cup \{head(r) \vee (head(r') - b) \leftarrow \\ & (body^+(r) \setminus \{b\}), not\ body^-(r), body(r')) \mid \\ & b \in body^+(r), \exists r' \in P \text{ s.t. } b \in head(r')\}. \end{aligned}$$

Here $head(r') - b$ is the disjunction obtained from $head(r')$ by removing b.

Since an implication is always an s-implication, the following result is a direct corollary of Theorem 4.1 in (Brass & Dix 1999).

Lemma 1 *Each disjunctive program P can be equivalently transformed into a negative program N via the program transformations in $\mathbf{T}^*_{\mathsf{WFS}}$, such that on no rule r in N, a literal appears in both the head and the body of r.*

Transformation-Based Algorithm

We are now in a position to present our syntax-based algorithm for computing forgetting in a disjunctive program.

Algorithm 2 (Computing a result of forgetting)
Input: disjunctive program P and a literal l in P.
Procedure:

Step 1. Fully apply the program transformations in $\mathbf{T}^*_{\mathsf{WFS}}$ on program P and then obtain a negative program N_0.

Step 2. Separate l from head disjunction via semi-shifting: For each (negative) rule $r \in N_0$ such that $head(r) = l \vee A$ and A is a non-empty disjunction, it is replaced by two rules: $l \leftarrow not\ A, body(r)$ and $A \leftarrow not\ l, body(r)$. Here $not\ A$ is the conjunction of all $not\ l'$ with l' in A. The resulting disjunctive program is denoted N.

Step 3. Suppose that N has n rules with head l:

$r_j : l \leftarrow not\ l_{j1}, ..., not\ l_{jm_j}$ where $n \geq 0$, $j = 1, \ldots, n$ and $m_j \geq 0$ for all j.

If $n = 0$, then let Q denote the program obtained from N by removing all appearances of $not\ l$.

If $n = 1$ and $m_1 = 0$, then $l \leftarrow$ is the only rule in N having head l. In this case, remove every rule in N whose body contains $not\ l$. Let Q be the resulting program.

For $n \geq 1$ and $m_1 > 0$, let $D_1, \ldots, D_s$ be all possible conjunctions $(not\ not\ l_{1k_1}, \cdots, not\ not\ l_{nk_n})$ where $0 \leq k_1 \leq m_1$, ..., $0 \leq k_n \leq m_n$. Replace in N each occurrence of $not\ l$ in N by all possible D_i. Let Q be the result.

Step 4. Remove all rules with head l from Q and output the resulting program N'.

Some remarks: (1) This is only a general algorithm. Some program transformations could be omitted for some special programs and various heuristics could also be employed to make the algorithm more efficient; (2) In this process, a result of forgetting is represented by a logic program allowing nested negation as failure. This form seems more intuitive than using ordinary logic programs; (3) In the construction of D_i, $not\ not\ l_{ij}$ cannot be replaced with l_{ij} (even for a normal logic program). As one can see, if they are replaced, the resulting program represents only a subset of $\mathcal{AS}_l(P)$ (see Example 2). This also implies that Algorithm 1 in (Wang, Sattar, & Su 2005) is incomplete in general. (4) Algorithm 2 above essentially improves the corresponding algorithm (Algorithm 1) in (Wang, Sattar, & Su 2005) at least in two ways: (i) our algorithm works for a more expressive class of programs (i.e. disjunctive programs) and (ii) the next result shows that our algorithm is complete under the minimal answer set semantics of nested logic programs.

Theorem 4 *Let P be a consistent disjunctive program and l a literal. Then X is an answer set of* forget(P, l) *iff X is a minimal answer set of N'.*

Example 2 *Consider $P_4 = \{c \leftarrow not\ q.\ p \leftarrow not\ q.\ q \leftarrow not\ p\}$. Then, by Algorithm 2,* forget(P_4, p) *is the nested program $\{c \leftarrow not\ q.\ q \leftarrow not\ not\ q\}$, whose minimal answer sets are exactly the same as the answer sets of* forget(P_4, p). *Note that Algorithm 1 in (Wang, Sattar, & Su 2005) outputs a program $N' = \{c \leftarrow not\ q.\ q \leftarrow q\}$ which has a unique answer set $\{c\}$. However,* forget(P_4, p) *has two answer sets $\{c\}$ and $\{q\}$. This implies that the algorithm there is incomplete.*

The above algorithm is worst case exponential, and might also output an exponentially large program. As follows from complexity considerations, there is no program P' that represents the result of forgetting which can be constructed in polynomial time, even if auxiliary literals might be used which are projected from the answer sets of P'. This is a consequence of the complexity results below.

However, the number of rules containing l may not be very large and some conjunctions D_i may be omitted because of redundancy. Moreover, the splitting technique of logic programs (Lifschitz & Turner 1994) can be used to localize the computation of forgetting. That is, an input program P is split into two parts so that the part irrelevant to forgetting is separated from the process of forgetting.

Resolving Conflicts in Multi-Agent Systems

In this section, we present a general framework for resolving conflicts in multi-agents systems, which is inspired from the *preference recovery* problem (Lang & Marquis 2002). Suppose that there are n agents who may have different preferences on the same issue. In many cases, these preferences (or constraints) have conflicts and thus cannot be satisfied at the same time. It is an important issue in constraint reasoning to find an intuitive criteria so that preferences with higher priorities are satisfied. Consider the following example.

Example 3 *(Lang & Marquis 2002) Suppose that a group of four residents in a complex tries to reach an agreement on building a* swimming pool *and/or a* tennis court. *The preferences and constraints are as follows.*

1. *Building a tennis court or a swimming pool costs each one unit of money.*
2. *A swimming pool can be either* red *or* blue.
3. *The first resident would not like to spend more than one money unit, and prefers a red swimming pool.*
4. *The second resident would like to build at least one of tennis court and swimming pool. If a swimming pool is built, he would prefer a blue one.*

5. *The third resident would prefer a swimming pool but either colour is fine with him.*
6. *The fourth resident would like both tennis court and swimming pool to be built. He does not care about the colour of the pool.*

Obviously, the preferences of the group are jointly inconsistent and thus it is impossible to satisfy them at the same time.

In the following, we will show how to resolve this kind of preference conflicts using the theory of forgetting.

An *n-agent system* $\mathcal{S}$ is an n-tuple $(P_1, P_2, \ldots, P_n)$ of disjunctive programs, $n > 0$, where P_i represents agent i's knowledge (including preferences, constraints).

As shown in Example 3, $P_1 \cup P_2 \cup \cdots \cup P_n$ may be inconsistent. The basic idea in our approach is to forget some literals for each agent so that conflicts can be resolved.

Definition 4 *Let $\mathcal{S} = (P_1, P_2, \ldots, P_n)$ be an n-agent system. A* compromise *of $\mathcal{S}$ is a sequence $C = (F_1, F_2, \ldots, F_n)$ where each F_i is a set of literals. An* agreement *of $\mathcal{S}$ on C is an answer set of the disjunctive program* $\mathsf{forget}(\mathcal{S}, C)$ *where* $\mathsf{forget}(\mathcal{S}, C) = \mathsf{forget}(P_1, F_1) \cup \mathsf{forget}(P_2, F_2) \cup \cdots \cup \mathsf{forget}(P_n, F_n)$.

For a specific application, we may need to impose certain conditions on each F_i.

Example 4 *(Example 3 continued) The scenario can be encoded as a collection of five disjunctive programs (P_0 stands for general constraints): $\mathcal{S} = (P_0, P_1, P_2, P_3, P_4)$ where*

P_0 = { $red \vee blue \leftarrow s.$ $\quad \leftarrow red, blue.$
$u_1 \leftarrow not\ s, t.$ $\quad u_1 \leftarrow s, not\ t.$
$u_2 \leftarrow s, t.$ $\quad u_0 \leftarrow not\ s, not\ t$};
P_1 = $\{u_0 \vee u_1 \leftarrow .\ \ red \leftarrow s\}$;
P_2 = $\{s \vee t \leftarrow .\ \ blue \leftarrow s\}$;
P_3 = $\{s \leftarrow\}$; *and* P_4 = $\{s \leftarrow .\ \ t \leftarrow\}$.

Since this knowledge base is jointly inconsistent, each resident may have to weaken some of her preferences so that an agreement is reached. Some possible compromises are:

1. *$C_1 = (\emptyset, F, F, F, F)$ where $F = \{s, blue, red\}$: Every resident would be willing to weaken her preferences on the swimming pool and its colour. Since $\mathsf{forget}(\mathcal{S}, C_1) = P_0 \cup \{u_0 \vee u_1 \leftarrow .\ \ t \leftarrow\}$, $\mathcal{S}$ has a unique agreement $\{t, u_1\}$ on C_1. That is, only a tennis court is built.*
2. *$C_2 = (\emptyset, F, F, F, F)$ where $F = \{u_0, u_1, u_2, blue, red\}$: Every resident can weaken her preferences on the price and the pool colour. Since $\mathsf{forget}(\mathcal{S}, C_2) = P_0 \cup \{s \vee t \leftarrow .\ \ s \leftarrow .\ \ t \leftarrow\}$, $\mathcal{S}$ has two possible agreements $\{s, t, red\}$ and $\{s, t, blue\}$ on C_2. That is, both a tennis court and a swimming pool will be built but the pool colour can be either red or blue.*
3. *$C_3 = (\emptyset, \{blue, red\}, \emptyset, \emptyset, \{t\})$: The first resident can weaken her preference on pool colour and the fourth resident can weaken her preference on tennis court. Since $\mathsf{forget}(\mathcal{S}, C_3) = P_0 \cup P_2 \cup P_3 \cup \{u_0 \vee u_1 \leftarrow .\ \ s \vee t \leftarrow .\ \ s \leftarrow\}$, $\mathcal{S}$ has a unique agreement $\{s, blue, u_1\}$ on C_3. That is, only a swimming pool will be built and its colour is blue.*

As shown in the example, different compromises lead to different results. We do not consider the issue of how to reach compromises here, which is left for future work.

Computational Complexity

In this section we address the computational complexity of forgetting for different classes of logic programs. Our results show that for general disjunctive programs, (1) the model checking of forgetting is Π_2^p-complete; (2) the credulous reasoning of forgetting is Σ_3^p-complete. However, for normal programs or negative disjunctive programs, the complexity levels are lower: (1) the model checking of forgetting is co-NP-complete; (2) the credulous reasoning of forgetting is Σ_2^p-complete. The design of Algorithm 2 in Section is heavily based on the complexity analysis here. Our complexity results for forgetting are summarized in the following table and formally stated after the table.

	disjunctive	negative	normal
model checking	Π_2^p	co-NP	co-NP
$\models_c$	Σ_3^p	Σ_2^p	Σ_2^p

Theorem 5 *Given a disjunctive program P, a literal l, and a set of literals X, deciding whether X is an l-answer set of P is Π_2^p-complete.*

Intuitively, in order to show that X is an l-answer set, we have to witness that X is an answer set (which is coNP-complete to test), and that there is no answer set X' of P such that $X' \subset_l X$. Any X' disproving this can be guessed and checked using an NP-oracle in polynomial time. Thus, l-answer set checking is in Π_2^p, as stated in Theorem 5.

Proof (Sketch) Π_2^p membership holds since checking whether a set of literals X' is an answer set of a disjunctive program P is in co-NP. The hardness result is shown by a reduction from deciding whether a given disjunctive program P (without strong negations) has no answer set, which is Π_2^p-complete (Eiter & Gottlob 1995). Construct a logic program $P' = \{head(r) \leftarrow p, body(r) \mid r \in P\} \cup \{q \leftarrow not\ p.\ \ p \leftarrow not\ q\} \cup \{a \leftarrow \mid a \text{ appears in } P\}$, where p and q are two fresh atoms. This program P' has one answer set X_0 in which p is false and all other atoms are true; all other answer sets are of the form $X \cup \{p\}$, where $X \in \mathcal{AS}(P)$. It holds that $X_0 \in \mathcal{AS}_p(P')$ iff P has no answer set. ■

The construction in the above proof can be extended to show Σ_3^p-hardness of credulous inference.

Theorem 6 *Given a disjunctive program P and literals l and l', deciding whether $\mathsf{forget}(P, l) \models_c l'$ is Σ_3^p-complete.*

In Theorem 6 a suitable l-answer set containing l' can be guessed and checked, by Theorem 5 using Σ_2^p-oracle. Hence, credulous inference $\mathsf{forget}(P, l) \models_c l'$ is in Σ_3^p. The matching lower bounds, Π_2^p- resp. Σ_3^p-hardness can be shown by encodings of suitable quantified Boolean Formulas (QBFs).

In Theorems 5 and 6, the complexity is coNP- and Σ_2^p-complete, respectively, if P is either negative or normal.

Theorem 7 *Given a* negative *program N, a literal l, and a set of literals X, deciding $X \in \mathcal{AS}_l(N)$ is co-NP-complete.*

Proof (Sketch) The co-NP membership holds since checking whether a set of literals X' is an answer set of a negative program P is polynomial. As for co-NP-hardness, let $C = C_1 \wedge \cdots \wedge C_k$ be a CNF over atoms

$y_1, \ldots, y_m$, where each C_j is non-empty. For $1 \leq i \leq m$, let $N_i = \{y_i \leftarrow not\ y_i'.\ y_i' \leftarrow not\ y_i.\ y_i \leftarrow not\ l.\ y_i' \leftarrow not\ l\}$, and for $1 \leq j \leq k$, let $Z_j = \{y_i \mid y_i \in C_j\} \cup \{y_i' \mid \neg y_i \in C_j\}$. Define $N = \cup_{i=1}^{m}(N_i \cup \{\ \leftarrow not\ Z_i\}) \cup \{l \leftarrow not\ y_1.\ l \leftarrow not\ y_1'\}$. Then, $X = \{y_i, y_i' \mid 1 \leq i \leq m\}$ is an answer set of N. Moreover, X is an l-answer set, iff C is unsatisfiable. The satisfiable assignments correspond to the answer sets of N containing l. ■

This construction can be lifted to show that credulous inference $\models_c$ of a literal from the l-answer sets of N is Σ_2^p-hard.

Theorem 8 *Given a negative program N and literals l and l', deciding whether* $\mathsf{forget}(N, l) \models_c l'$ *is Σ_2^p-complete.*

Proof (Sketch). By Theorem 7, Σ_2^p membership is easy. As for Σ_2^p-hardness, take a QBF $\exists X \forall Z E$, where E is a DNF on X and Z and contains some variable from Z in each clause. Construct the same program as above in Theorem 7 for $C = \neg E$ and where $Y = X \cup Z$ and y_1 is from Z, but (1) omit the clauses $x_i \leftarrow not\ l$ and $x_i' \leftarrow not\ l$. (2) add a clause $l' \leftarrow not\ l$. For each subset $S \subseteq X$, the set

$S \cup \{x_i' \mid x_i \in X \setminus S\} \cup Z \cup \{z_j' \mid z_j \in Z\} \cup \{l'\}$

is an answer set of N. These are also all answer sets of N that contain l' (and do not contain l). Furthermore, this set is an l-answer set, iff there is no satisfying assignment for C (=$\neg E$) which corresponds on X to S. Overall, this means that there is some l-answer set of the program in which l' is true, iff the formula $\exists X \forall Z E$ is true. ■

In the proof of Theorem 7, a CNF is actually reduced to a normal program. It is thus easy to see the following result.

Theorem 9 *Given a* normal *program P, a literal l, and a set of literals X, deciding whether X is an l-answer set of P is co-NP-complete.*

Similarly, we can show the credulous reasoning with forgetting for normal program is Σ_2^p-complete.

Theorem 10 *Given a* normal *program P, a literal l, and a literal l', deciding whether* $\mathsf{forget}(P, l) \models_c l'$ *is Σ_2^p-complete.*

By applying techniques that build on non-uniform complexity classes similar as in (Cadoli *et al.* 2000), we conjecture that there is no program $\mathsf{forget}(P, l)$ of polynomial size unless the polynomial hierarchy collapses, even if auxiliary literals might be used (which are projected off). Thus, the exponential blow up of $\mathsf{forget}(P, l)$ is, to some extent, unavoidable in general.

Related Work and Conclusion

We have proposed a theory of forgetting literals in disjunctive programs. Although our approach is purely declarative, we have proved that it is coupled by a syntactic counterpart based on program transformations. The properties of forgetting show that our approach captures the classical notion of forgetting. As we have explained before, the approach in this paper naturally generalizes the forgetting for normal programs investigated in (Wang, Sattar, & Su 2005).

Another approach to forgetting for normal programs is proposed in (Zhang, Foo, & Wang 2005), which is purely procedural since the result of forgetting is obtained by removing some rules and/or literals. A shortcoming of that approach is that there is, intuitively, no semantic justification for the removal.

As an application of forgetting, we have also presented a fairly general framework for resolving conflicts in disjunctive logic programming. In particular, this framework provides an elegant solution to the preference recovery problem. There are some interesting issues to be pursued. First, we are currently improving and implementing the algorithm for computing the result of forgetting. Second, we will explore the application of forgetting in various scenarios of conflict resolving, such as belief merging, update of disjunctive programs, inheritance in disjunctive programs.

References

Baral, C. 2002. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Ben-Eliyahu, R., and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Ann. Math. and AI* 12(1-2):53–87.

Brass, S., and Dix, J. 1999. Semantics of disjunctive logic programs based on partial evaluation. *J. Logic Programming* 38(3):167–312.

Cadoli, M.; Donini, F.; Liberatore, P.; and Schaerf, M. 2000. Space Efficiency of Propositional Knowledge Representation Formalisms. *J. Artif. Intell. Res.* 13:1–31.

Eiter, T., and Fink, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Proc. 19th ICLP*, 224–238.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. and AI* 15(3-4):289–323.

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In *Proc. ICLP*, 579–597.

Janhunen, T.; Niemelä, I.; Simons, P.; and You, J.-H. 2000. Partiality and Disjunctions in Stable Model Semantics. In *Proc. KR 2000*, 411–419.

Lang, J., and Marquis, P. 2002. Resolving inconsistencies by variable forgetting. In *Proc. 8th KR*, 239–250.

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res.* 18:391–443.

Lee, J., and Lifschitz, V. 2003. Loop formulas for disjunctive logic programs. In *Proc. ICLP*, 451–465.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2004. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL* (to appear).

Lifschitz, V.; Tang, L.; and Turner, H. 1999. Nested expressions in logic programs. *Ann. Math. and AI* 25:369–389.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Proc. ICLP*, 23-37.

Lin, F., and Reiter, R. 1994. Forget it. In *Proc. AAAI Symp. on Relevance*, 154–159.

Lin, F., and Zhao, Y. 2004. ASSAT: Computing answer set of a logic program by sat solvers. *Artif. Intell.* 157(1-2): 115–137.

Wang, K., and Zhou, L. 2005. Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM TOCL* 6(2):295–327.

Wang, K.; Sattar, A.; and Su, K. 2005. A theory of forgetting in logic programming. In *Proc. 20th AAAI*, 682–687. AAAI Press.

Zhang, Y.; Foo, N.; and Wang, K. 2005. Solving logic program conflicts through strong and weak forgettings. In *Proc. IJCAI*, 627–632.

1.10 Analysing the Structure of Definitions in ID-logic

Analyzing the Structure of Definitions in ID-logic*

Joost Vennekens and **Marc Denecker**
{joost.vennekens, marc.denecker}@cs.kuleuven.be
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium

Abstract

ID-logic uses ideas from logic programming to extend classical logic with non-monotone inductive definitions. Here, we study the structure of definitions expressed in this formalism. We define the fundamental concept of a dependency relation, both in an abstract, algebraic context and in the concrete setting of ID-logic. We also characterize dependency relations in a more constructive way. Our results are used to study the relation between ID-logic and known classes of inductive definitions and to show the correctness of ID-logic semantics in these cases.

Introduction

Inductive definitions are a distinctive and well-understood kind of knowledge, which occurs often in mathematical practice. The roots of ID-logic lie in the observation that logic programs under the well-founded semantics can be seen as a formal equivalent of this informal mathematical construct (Denecker 1998). This result is particularly useful, because inductive definitions cannot be easily represented in classical logic. ID-logic uses a form of logic programs under the well-founded semantics to extend classical logic with a new "inductive definition" primitive. In the resulting formalism, all kinds of definitions regularly found in mathematical practice can be represented in a uniform way. Moreover, the rule-based representation of a definition in ID-logic neatly corresponds to the form such a definition would take in a mathematical text. ID-logic also has interesting applications in common-sense reasoning. For instance, (Denecker & Ternovska 2004a) gives a natural representation of situation calculus as an iterated inductive definition in ID-logic. The resulting theory correctly handles tricky issues such as recursive ramifications, and is, as far as we know, the most general representation of this calculus to date.

The main ideas behind ID-logic and the relation between the well-founded semantics and inductive definitions have been further generalized in *approximation theory* (Denecker, Marek, & Truszczynski 2003; 2000), an algebraic fixpoint theory for arbitrary operators. Interestingly, this theory not only captures the well-founded semantics for logic programs, but also other logic programming semantics, such as the stable model semantics, as well as several different semantics for other non-monotonic reasoning formalisms, such as auto-epistemic logic and default logic. Approximation theory provides an abstract framework in which general properties of a variety of different semantics for different logics can be succinctly proven.

In this paper, we analyze the structure of definitions in ID-logic. This analysis takes place at three different levels. Firstly, we use approximation theory to analyze definitions in ID-logic by studying the internal structure of certain lattice operators. To this end, we define the algebraic concept of a *dependency relation* for an operator. It turns out that this can be related to a theory of modularity in approximation theory (Vennekens, Gilis, & Denecker 2005), allowing a number of quite general results to easily be derived. Secondly, we also define a similar concept at the more specific level of ID-logic and relate this to its algebraic counterpart. When instantiated to this level, the properties proven in approximation theory immediately provide us with several interesting results, such as a splitting theorem for ID-logic. Finally, we also present a constructive characterization of a specific kind of dependency relation for an ID-logic definition, based on the syntactical structure of its rules.

As an application of these results, we study several classes of inductive definitions known from mathematical literature. The concept of dependency relations can be used to offer a natural definition for each of these classes. The fact that ID-logic correctly formalizes the semantics of definitions belonging to these classes can then be proven in approximation theory. Finally, our constructively characterized dependency relations lead to some semi-syntactical ways of identifying members of each of these classes.

The work in this paper is part of a larger project (Vennekens & Denecker 2005; Denecker & Ternovska 2004b) to establish firm mathematical foundations for ID-logic. While our results are largely theoretical, they are meant to serve as a basis for more practical work. In particular, they should help to more clearly establish the knowledge representation methodology of ID-logic, offer some mathematical tools with which to analyze theories in this logic and prove their correctness, and contribute to the development of efficient reasoning algorithms for (decidable fragments of) ID-logic. The importance of this kind of research can be motivated by looking at its accomplishments in logic programming,

*This work was supported by FWO-Vlaanderen, European Framework 5 Project WASP, and by GOA/2003/08.

where the use of concepts such as dependency graphs dates back at least as far as (Apt, Blair, & Walker 1988). As we will see, the same kind of techniques can be applied in the more complex setting of ID-logic. More recently, Answer Set Programming has seen more results in a similar vein, with work being done to identify interesting subclasses of programs, such as *tight logic programs* (Erdem & Lifschitz 2003). This work is very similar in spirit to the analysis of various subclasses of inductive definitions, that we will present later in this paper.

Preliminaries

Approximation theory

Our presentation of approximation theory is based on (Denecker, Marek, & Truszczynski 2000; 2003). Let $\langle L, \leq \rangle$ be a lattice. We consider the square L^2 of the domain of this lattice. The obvious point-wise extension of $\leq$ to L^2 is called the *product order* $\leq_\otimes$ on L^2, i.e., for all $(x,y),(x',y') \in L^2$, $(x,y) \leq_\otimes (x',y')$ iff $x \leq x'$ and $y \leq y'$. An element (x,y) of L^2 can be seen as denoting an interval $[x,y] = \{z \in L \mid x \leq z \leq y\}$. Using this intuition, we can derive a second order, the *precision order* $\leq_p$, on L^2: for each $(x,y),(x',y') \in L^2, (x,y) \leq_p (x',y')$ iff $x \leq x'$ and $y' \leq y$. Indeed, if $(x,y) \leq_p (x',y')$, then $[x,y] \supseteq [x',y']$. It can easily be seen that $\langle L^2, \leq_p \rangle$ is also a lattice. The structure $\langle L^2, \leq_\otimes, \leq_p \rangle$ is the *bilattice* corresponding to L. If L is complete, then so are $\langle L^2, \leq_\otimes \rangle$ and $\langle L^2, \leq_p \rangle$. Elements (x,x) of L^2 are called *exact*. The set of exact elements forms a natural embedding of L in L^2.

Approximation theory is based on the study of operators which are monotone w.r.t. $\leq_p$. Such operators are called *approximations*. For an approximation A and $x, y \in L$, we denote by $A^1(x,y)$ and $A^2(x,y)$ the unique elements of L, for which $A(x,y) = (A^1(x,y), A^2(x,y))$. An approximation *approximates* an operator O on L if for each $x \in L$, $A(x,x)$ contains $O(x)$, i.e. $A^1(x,x) \leq O(x) \leq A^2(x,x)$. An *exact* approximation maps exact elements to exact elements, i.e. $A^1(x,x) = A^2(x,x)$ for all $x \in L$. Each exact approximation approximates a unique operator O on L, namely that which maps each $x \in L$ to $A^1(x,x) = A^2(x,x)$. An approximation is *symmetric* if for all $(x,y) \in L^2$, if $A(x,y) = (x',y')$ then $A(y,x) = (y',x')$. Each symmetric approximation is exact.

For an approximation A on L^2, we define the operator $A^1(\cdot, y)$ on L that maps an element $x \in L$ to $A^1(x,y)$, i.e. $A^1(\cdot,y) = \lambda x.A^1(x,y)$, and $A^2(x,\cdot)$ that maps an element $y \in L$ to $A^2(x,y)$. These are monotone operators and, therefore, they each have a unique least fixpoint. We define an operator $C_A^\downarrow$ on L, which maps each $y \in L$ to $lfp(A^1(\cdot,y))$ and, similarly, an operator $C_A^\uparrow$, which maps $x \in L$ to $lfp(A^2(x,\cdot))$. $C_A^\downarrow$ is called the *lower stable operator* of A, while $C_A^\uparrow$ is the *upper stable operator* of A. Both these operators are anti-monotone. We define the *partial stable operator* $\mathcal{C}_A$ on L^2 as mapping each (x,y) to $(C_A^\downarrow(y), C_A^\uparrow(x))$. Because the lower and upper partial operators $C_A^\downarrow$ and $C_A^\uparrow$ are anti-monotone, the partial stable operator $\mathcal{C}_A$ is $\leq_p$-monotone. If A is symmetric, then its lower and upper stable operators are equal, i.e., $C_A^\downarrow = C_A^\uparrow$.

An approximation A defines a number of different fixpoints: the least fixpoint of an approximation A is called its *Kripke-Kleene fixpoint*, fixpoints of its partial stable operator $\mathcal{C}_A$ are *stable fixpoints* and the least fixpoint of $\mathcal{C}_A$ is called the *well-founded fixpoint* $wf(A)$ of A. As shown in (Denecker, Marek, & Truszczynski 2000; 2003), these fixpoints correspond to various semantics of logic programming, auto-epistemic logic and default logic. Finally, it should also be noted that the concept of an approximation as defined in these works corresponds to our definition of a *symmetric* approximation.

ID-logic

ID-logic (Denecker & Ternovska 2004b; 2004a) extends classical logic with non-monotone inductive definitions. Actually, the term "ID-logic" refers to a family of logics, depending on which particular version of classical logic serves as a base, i.e., we have the extension $FO(ID)$ of first-order logic with inductive definitions, the extension $SO(ID)$ of second-order logic with inductive definitions, and so on. Because the results of this paper concern the structure of the inductive definitions, it does not really matter which base logic is considered. For generality, we will use $SO(ID)$.

Following (Denecker & Ternovska 2004a), we start by presenting standard second-order logic in a slightly non-standard way. In particular, no distinction is made between constant symbols and variables. We assume an infinite supply of *object symbols* $x, y, \ldots$, *function symbols* f/n, g/n,... of every arity n, and *predicate symbols* $P/n, Q/n, \ldots$ of every arity n. As usual, object symbols are identified with function symbols of arity 0. A *vocabulary* Σ is a set of symbols. We denote by Σ_o the object symbols in Σ, by Σ_f the function symbols, and by Σ_P the predicate symbols. *Terms* and *atoms* of Σ are defined as usual. A *formula of* Σ is inductively defined as:

- a Σ-atom $P(t_1, \ldots, t_n)$ is a Σ-formula;
- if ϕ is a Σ-formula, then so is $\neg\phi$;
- if ϕ_1 and ϕ_2 are Σ-formulas, then so is $(\phi_1 \vee \phi_2)$;
- if ϕ is a $(\Sigma \cup \{\sigma\})$-formula for some symbol σ, then $(\exists\sigma\ \phi)$ is a Σ-formula.

If in all quantifications $\exists\sigma$ of a formula ϕ, σ is an object symbol, ϕ is called *first order*[1]. We also use the usual abbreviations $\forall$ and $\wedge$. Let $\mathcal{V}$ be a set of truth values. Most commonly, $\mathcal{V}$ is $L_2 = \{\mathbf{t}, \mathbf{f}\}$. Given a certain domain D, a symbol σ can be assigned a *value* in D: if $\sigma/n \in \Sigma_f$, a value for σ in D is a function of arity n in D; if $\sigma/n \in \Sigma_P$, a value for σ in D is a function from D^n to $\mathcal{V}$.

A *($\mathcal{V}$-valued) structure S for vocabulary* Σ, or *($\mathcal{V}$-valued) Σ-structure* S, consists of a domain, denoted D_S, and a mapping from each symbol σ in Σ to a value σ^S in D_S for σ. The *restriction* $S'|_\Sigma$ of a Σ'-structure S' to a sub-vocabulary

[1] The first-order version $FO(ID)$ of ID-logic can be defined by requiring that all formulas be first-order. All other definitions would remain the same.

$\Sigma \subseteq \Sigma'$, is the Σ-structure S for which $D_S = D_{S'}$ and, for each symbol σ of Σ, $\sigma^S = \sigma^{S'}$. Under the same conditions, S' is called an *extension* of S to Σ'. For each value a in D_S for a symbol σ, we denote by $S[\sigma/a]$ the extension S' of S to $\Sigma \cup \{\sigma\}$, such that $\sigma^{S'} = a$. We also use this notation for tuples $\vec{\sigma}$ of symbols and $\vec{a}$ of corresponding values.

The *value* of a Σ-term t in a Σ-structure S, also denoted t^S, is inductively defined as: $(f(t_1, \ldots, t_n))^S = f^S(t_1^S, \ldots, t_n^S)$, for a function symbol f and terms $t_1, \ldots, t_n$. We now assume that the set of truth-values $\mathcal{V}$ is partially ordered by some $\leq_\mathcal{V}$, s.t. $\langle \mathcal{V}, \leq_\mathcal{V} \rangle$ is a complete lattice. Moreover, we assume that for each $v \in \mathcal{V}$, a *complement* $v^{-1} \in \mathcal{V}$ exists. For L_2, $\mathbf{f} \leq_{L_2} \mathbf{t}$ and $\mathbf{f}^{-1} = \mathbf{t}$, $\mathbf{t}^{-1} = \mathbf{f}$. We inductively define the truth value of a Σ-formula ϕ in a $\mathcal{V}$-valued Σ-structure S:

- $P(t_1, \ldots, t_n)^S = P^S(t_1^S, \ldots, t_n^S)$;
- $(\neg\phi)^S = (\phi^S)^{-1}$;
- $(\phi \vee \psi)^S = lub_{\leq_\mathcal{V}}(\phi^S, \psi^S)$;
- $(\exists\sigma\ \phi)^S = lub_{\leq_\mathcal{V}}\{\phi^{S[x/a]} \mid a \text{ is a value for } \sigma \text{ in } D_S\}$.

Given a domain D, a *domain atom* is a pair $(P, \vec{a})$, with P/n a predicate of Σ and $\vec{a} \in D^n$. We also write such a pair as $P(\vec{a})$. The set of all domain atoms is denoted as $\mathcal{A}t_D$ or, if D is clear from the context, simply $\mathcal{A}t$. For a structure S, we also write $\mathcal{A}t_S$ for $\mathcal{A}t_{D_S}$. A *pre-interpretation H for Σ* is a structure for the language Σ_f, i.e., one which interprets only the object and function symbols of Σ. A structure S extending H to Σ is called an *H-interpretation*. Clearly, each $\mathcal{V}$-valued Σ-structure can be seen as consisting of a pre-interpretation H and a mapping from $\mathcal{A}t_H$ to $\mathcal{V}$. The set of all H-interpretations is a complete lattice w.r.t. to the order $\leq_t$, defined as: $S \leq_t S'$ iff for all $P(\vec{a}) \in \mathcal{A}t$, $P(\vec{a})^S \leq_\mathcal{V} P(\vec{a})^{S'}$.

We now define the syntax used for inductive definitions in ID-logic. Let Σ be a vocabulary. A *definitional rule* r of Σ is a formula "$\forall\vec{x}\ A \leftarrow \phi$," with A a $(\Sigma \cup \vec{x})$-atom and ϕ a first-order $(\Sigma \cup \vec{x})$-formula, i.e., no second-order quantifications are allowed within a definition. The atom A is called the *head* of r and ϕ is the *body* of r. The symbol "$\leftarrow$" is a new language primitive, the *definitional implication*, which is different from material implication. A *definition* Δ is a set of definitional rules, enclosed in curly brackets $\{\}$. A predicate P for which there is a rule $r \in \Delta$ with P in its head is a *defined predicate* of Δ. Predicates that are not defined in Δ are *open* in Δ. Given a domain D, we denote the set of all domain atoms $P(\vec{a}) \in \mathcal{A}t_D$ for which P is defined in Δ by Def_Δ^D, while its complement $\mathcal{A}t_D \setminus Def_\Delta^D$ is denoted as $Open_\Delta^D$. Once again, D is omitted if it is clear from the context. A *Σ-definition* Δ is a set of definitional rules. Given a pre-interpretation H, a rule r is a *defining rule* of a domain atom $P(\vec{a}) \in Def_\Delta$ under a substitution $[\vec{x}/\vec{c}]$ iff r is $\forall\vec{x}\ P(\vec{t}) \leftarrow \phi$ with $\vec{t}^{H[\vec{x}/\vec{c}]} = a$.

Definition 1. Let Σ be a vocabulary. An *ID-logic formula of* Σ is inductively defined by extending the definition of a formula with the additional base case:

- A definition Δ of Σ is an ID-logic formula of Σ.

Example 1. We consider a game played with a pile of n stones. Two players subsequently remove either one stone or two. The player to make the last move wins. The winning positions of this game can be inductively defined by the following definition Δ_{Game}:

$$\left\{ \begin{array}{r} \forall x\ Win(x) \leftarrow \exists y\ Move(x,y) \wedge \neg Win(y). \\ \forall x,y\ Move(x,y) \leftarrow y \geq 0 \\ \wedge\ ((y = x-1) \vee (y = x-2)). \end{array} \right\}$$

The second rule defines the legal moves of the game. The first rule expresses that a winning position has a move to a position in which the opponent loses. This rule has been around in logic programming since, at least, (Gelder, Ross, & Schlipf 1991) and it therefore illustrates that the connection between inductive definitions and logic programs that underlies ID-logic has been implicitly present in the domain for some time.

To formally state the semantics of such a definition, we can either work in Belnap's lattice $L_4 = \{\mathbf{u}, \mathbf{t}, \mathbf{f}, \mathbf{i}\}$ or in the lattice L_2^2 of pairs of elements of L_2. These settings are known to be equivalent under the mapping h from L_2^2 to L_4, defined by: $h(\mathbf{f}, \mathbf{t}) = \mathbf{u}$, $h(\mathbf{t}, \mathbf{t}) = \mathbf{t}$, $h(\mathbf{f}, \mathbf{f}) = \mathbf{f}$, and $h(\mathbf{t}, \mathbf{f}) = \mathbf{i}$. For the remainder of this section, we assume a certain fixed pre-interpretation H and identify the set of $\mathcal{V}$-valued H-interpretations with $\mathcal{V}^{\mathcal{A}t}$, i.e., with the set of all $\mathcal{V}$-valued functions on $\mathcal{A}t$. We use symbols $\mathcal{R}, \mathcal{S}, \ldots$ for L_4-valued structures and $R, S, \ldots$ for L_2-valued structures. The correspondence h between L_2^2 and L_4 induces an isomorphism between L_4-valued structures and pairs (S_1, S_2) of L_2-valued structures: we denote by $S_1 \oplus S_2$ the structure $\mathcal{S}$ which assigns to each $P(\vec{a}) \in \mathcal{A}t$ the truth-value $h(P(\vec{a})^{S_1}, P(\vec{a})^{S_2})$. The set of all $S \oplus S$ with S a L_2-valued structure forms a natural embedding of L_2 in L_4.

To make it more convenient to relate the semantics of ID-logic to approximation theory, we will define this in the lattice L_2^2. The truth value of a formula can be evaluated in pairs of L_2-valued structures as follows:

Definition 2. Let S_1 and S_2 be L_2-valued Σ-structures and ϕ a Σ-formula. The *value* of ϕ in (S_1, S_2) is inductively defined by:

- $P(\vec{t})^{(S_1,S_2)} = P(\vec{t})^{S_1}$;
- $(\neg\phi)^{(S_1,S_2)} = ((\phi)^{(S_2,S_1)})^{-1}$;
- $(\phi \vee \psi)^{(S_1,S_2)} = lub_{\leq_{L_2}}(\phi^{(S_1,S_2)}, \psi^{(S_1,S_2)})$;
- $(\exists\sigma\ \phi)^{(S_1,S_2)} = lub_{\leq_{L_2}}(\{\phi^{(S_1[\sigma/a],S_2[\sigma/a])} \mid a \text{ is a value for } \sigma \text{ in } H_D\})$.

Observe that in the rule for $\neg\phi$, the roles of S_1 and S_2 are switched. It is worth mentioning that for all pairs (S_1, S_2) of L_2-valued structures, the standard L_4-valued evaluation $\phi^{S_1 \oplus S_2}$ is equal to $h(\phi^{(S_1,S_2)}, \phi^{(S_2,S_1)})$. The evaluation in pairs of L_2-valued structures also has an intuitive appeal of its own: let us consider a structure S approximated by (S_1, S_2), i.e., such that $S_1 \leq_t S \leq_t S_2$. In the evaluation of ϕ in (S_1, S_2), all positively occurring atoms are evaluated with respect to the underestimate S_1 of S, and all negatively occurring atoms are evaluated with respect to the overestimate S_2 of S. Therefore, the truth value of ϕ in (S_1, S_2) is

an underestimate of the value of ϕ in S. Vice versa, in the evaluation of ϕ in (S_2, S_1), all positively occurring atoms are evaluated in the overestimate S_2 while all negatively occurring atoms are evaluated in the underestimate S_1; hence, the truth value of ϕ in (S_2, S_1) is an overestimate of the value of ϕ in S.

Intuitively, an inductive definition describes a process by which, given some fixed interpretation of the open predicates, new truth values for the defined atoms can be derived from some current truth values for these atoms. We will define an immediate consequence operator $\mathcal{T}_\Delta^\mathcal{R}$ that maps an estimate (S_1, S_2) of the defined relations to a more precise estimate $\mathcal{T}_\Delta^\mathcal{R}(S_1, S_2) = (S_1', S_2')$. The new lower bound S_1' is constructed by underestimating the truth of the bodies of the rules in Δ, i.e., by evaluating these in (S_1, S_2). When constructing the new upper bound S_2', on the other hand, the truth of the bodies of these rules is overestimated, i.e., evaluated in (S_2, S_1).

Definition 3. Let Δ be a definition and (R_1, R_2) a pair of L_2-valued structures which interprets at least $Open_\Delta$. We define a function $U_\Delta^{(R_1,R_2)}$ from $(L_2^{Def_\Delta})^2$ to $L_2^{Def_\Delta}$ as $U_\Delta^{(R_1,R_2)}(S_1, S_2) = S$, with for each $P(\vec{a}) \in Def_\Delta$:

$$P(\vec{a})^S = lub_{\leq_{L_2}}(\{\phi^{((R_1 \cup S_1)[\vec{x}/\vec{c}],(R_2 \cup S_2)[\vec{x}/\vec{c}])} \mid$$

"$\forall \vec{x}\, P(\vec{t}) \leftarrow \phi$" is a defining rule of $P(\vec{a})$ in S under $[\vec{x}/\vec{c}]\}$).

We define $\mathcal{T}_\Delta^{(R_1,R_2)}(S_1, S_2)$ as $\big(U_\Delta^{(R_1,R_2)}(S_1, S_2), U_\Delta^{(R_2,R_1)}(S_2, S_1)\big)$.

Each such operator is an approximation. Moreover, for every L_2-valued R, $\mathcal{T}_\Delta^{(R,R)}$ is symmetric. Every $\mathcal{T}_\Delta^{(R_1,R_2)}$ approximates the operator $T_\Delta^{(R_1,R_2)}$ on $L_2^{Def_\Delta}$, defined as $T_\Delta^{(R_1,R_2)}(S) = S'$, with $(S', S') = \mathcal{T}_\Delta^{(R_1,R_2)}(S, S)$. Sometimes, it will be convenient to use an equivalent operator on the lattice L_4, i.e., for every $\mathcal{R} = R_1 \oplus R_2$ and $\mathcal{S} = S_1 \oplus S_2$, we define $\mathcal{T}_\Delta^\mathcal{R}(\mathcal{S}) = U_\Delta^{(R_1,R_2)}(S_1, S_2) \oplus U_\Delta^{(R_2,R_1)}(S_2, S_1)$. We now use the well-founded fixpoint of the approximation $\mathcal{T}_\Delta^{(R_1,R_2)}$ to define the semantics of ID-logic.

Definition 4. A L_2-valued structure S satisfies an ID-logic formula ϕ, denoted $S \models \phi$, if $\phi^S = \mathbf{t}$, where ϕ^S is defined by the standard inductive definition of the L_2-valued truth value, extended by the additional base case:

- for a definition Δ, $\Delta^S = \mathbf{t}$ if $S_1|_{Def_\Delta} = S|_{Def_\Delta} = S_2|_{Def_\Delta}$, with $(S_1, S_2) = wf(\mathcal{T}_\Delta^{(S,S)})$; otherwise $\Delta^S = \mathbf{f}$.

Even though this definition uses the operator $\mathcal{T}_\Delta^{(R_1,R_2)}$ on pairs of L_2-valued structures (or, equivalently, L_4-valued structures), the eventual models of a definition are always single L_2-valued structures. The intuition here is that a definition should completely and consistently define its defined predicates, i.e., no defined domain atoms should be $\mathbf{u}$ or $\mathbf{i}$. For a L_4-valued structure $\mathcal{S}$, we therefore only say that $\mathcal{S} \models \phi$ iff there exists a L_2-valued S, such that $\mathcal{S} = S \oplus S$ and $S \models \phi$.

Algebraic dependency relations

This section studies dependency relations in an algebraic context. We assume the following setting. Let I be some index set and, for each $i \in I$, let $\langle L_i, \leq_i \rangle$ be a lattice. Let L be the product $\bigotimes_{i \in I} L_i$ of the sets $(L_i)_{i \in I}$, i.e., L consists of all functions $f : I \to \bigcup_{i \in I} L_i$, such that $\forall i \in I : f(i) \in L_i$. The product order $\leq_\otimes$ on L is defined as: $\forall x, y \in L$, $x \leq_\otimes y$ iff $\forall i \in I$, $x(i) \leq_i y(i)$. Clearly, $\langle L, \leq_\otimes \rangle$ is also a lattice, which is complete if all lattices $\langle L_i, \leq_i \rangle$ are complete. For a subset $J \subseteq I$, we denote by $L|_J$ the part $\otimes_{j \in J} L_j$ of L, which is equal to the set of all restrictions $x|_J$, with $x \in L$. Now, let O be an operator on L. We are interested in the internal structure of O w.r.t. the component lattices L_i of L. For instance, what information about x is used by O to determine the value $(O(x))(i)$ of some $O(x)$ in a component L_i? Does such an $(O(x))(i)$ depend on the value $x(j)$ of x in each L_j? Or is there some $J \subset I$, such that the restriction $x|_J$ of x to this J already completely determines what $(O(x))(i)$ will be? The following concept captures these basic dependencies expressed by an operator. For a binary relation θ on a set S and $y \in S$, we write (θy) for $\{x \in S \mid x \theta y\}$.

Definition 5. Let O be an operator on a lattice $L = \otimes_{i \in I} L_i$. A binary relation $\rightsquigarrow$ on I is a *dependency relation* of O iff for each $i \in I$ and $x, y \in L$, if $x|_{(\rightsquigarrow i)} = y|_{(\rightsquigarrow i)}$, then $(O(x))(i) = (O(y))(i)$.

An operator can have many dependency relations. In fact, any superset of a dependency relation of an operator O is also a dependency relation of O. Therefore, smaller dependency relations are more informative. An operator does not necessarily have a least dependency relation.

In (Vennekens, Gilis, & Denecker 2005; Vennekens & Denecker 2005), an algebraic theory of modularity was developed. This theory focuses on the study of *stratifiable* operators, i.e., operators on a lattice $\bigotimes_{i \in I} L_i$ whose index set I can be partially ordered by some $\preceq$, such that the value of $(O(x))(i)$ depends only on the value of x in $L|_{(\preceq i)}$.

Definition 6. An operator O on a lattice $L = \otimes_{i \in I} L_i$ is *stratifiable* w.r.t. a partial order $\preceq$ on I iff for all $x, y \in L$ and $i \in I$: if $x|_{(\preceq i)} = y|_{(\preceq i)}$ then $O(x)|_{(\preceq i)} = O(y)|_{(\preceq i)}$.

The main results from (Vennekens & Denecker 2005) concern the relation between a stratifiable operator O and certain smaller operators which can be derived from O. For $J \subseteq I$ and $u \in L|_{I \setminus J}$, we denote by O_J^u the operator on $L|_J$ which maps each $x \in L|_J$ to $O(y)|_J$, with $y|_{I \setminus J} = u$ and $y|_J = x$. Such O_J^u are called *components* of O.

Theorem 1 ((Vennekens & Denecker 2005)). *Let O be an operator on a lattice $L = \otimes_{i \in I} L_i$ which is stratifiable w.r.t. a well-founded[2] partial order $\preceq$ on I. Let $\mathcal{J}$ be a partition of I. For each $x \in L$, x is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of O (assuming that O is monotone or an approximation, where appropriate) iff for each $J \in \mathcal{J}$, $x|_J$ is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of $O_J^{x|_{I \setminus J}}$.*

[2] A binary relation θ on a set S is *well-founded* iff there exists no infinite sequence $x_0, x_1, x_2, \ldots \in S$, s.t. $x_{i+1} \theta x_i$ for all i.

We now show that there exists a uniform way of stratifying an operator O, given one of its dependency relations $\rightsquigarrow$. Let $\leq_{\rightsquigarrow}$ be the reflexive, transitive closure of $\rightsquigarrow$. For each $i \in I$, we denote by $\overline{i}$ the equivalence class $\{j \in I \mid j \leq_{\rightsquigarrow} i$ and $i \leq_{\rightsquigarrow} j\}$ of i. We denote by $E_{\rightsquigarrow}$ the set $\{\overline{i} \mid i \in I\}$ of all equivalence classes of $\leq_{\rightsquigarrow}$ and by $\preceq_{\rightsquigarrow}$ the partial order on $E_{\rightsquigarrow}$ derived from $\leq_{\rightsquigarrow}$, i.e., for all $i, j \in I$, $\overline{i} \preceq_{\rightsquigarrow} \overline{j}$ iff $i \leq_{\rightsquigarrow} j$. Now, O can also be viewed as an operator on $\otimes_{\overline{i} \in E_{\rightsquigarrow}} L|_{\overline{i}}$. This follows from the fact that any product lattice $\otimes_{i \in I} L_i$ is isomorphic to $\otimes_{J \in \mathcal{J}} \otimes_{j \in J} L_j$, for any partition $\mathcal{J}$ of I. This allows us to relate the concept of a dependency relation to that of stratifiability.

Proposition 1. *Let O be an operator on $L = \otimes_{i \in I} L_i$. If a binary relation $\rightsquigarrow$ on I is a dependency relation of O, then O is stratifiable on $\otimes_{\overline{i} \in E_{\rightsquigarrow}} L|_{\overline{i}}$ w.r.t. $\preceq_{\rightsquigarrow}$.*

It can easily be seen that $\preceq_{\rightsquigarrow}$ is well-founded iff $\rightsquigarrow$ is. Theorem 1 now implies the following corollary:

Corollary 1. *Let O be an operator on a lattice $L = \otimes_{i \in I} L_i$, with a well-founded dependency relation $\rightsquigarrow$. Let $\mathcal{J}$ be a partition of I, such that for each equivalence class $\overline{i}$ of $\leq_{\rightsquigarrow}$, there exists a $J \in \mathcal{J}$, such that $\overline{i} \subseteq J$. For each $x \in L$, x is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of O (assuming that O is monotone or an approximation, where appropriate) iff for each $J \in \mathcal{J}$, $x|_J$ is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of $O_J^{x|_{I \setminus J}}$.*

This results shows that if we know a dependency relation for an operator, we will be able to split this operator into components, while still preserving its (various kinds of) fixpoints. Indeed, as long as the stratification is done in such a way that none of the equivalence classes of this dependency relation is split over different levels, we know that this will be the case.

Dependency Relations in ID-logic

In this section, we apply our algebraic results to ID-logic. We fix a vocabulary Σ and a pre-interpretation H for Σ. We restrict our attention to H-interpretations, which can therefore be viewed as assignments of truth values to domain atoms. We study properties of ID-logic in the following product lattice:

$$\bigotimes_{P(\vec{a}) \in \mathcal{A}t} L_4 = L_4^{\mathcal{A}t} = \bigotimes_{P(\vec{a}) \in \mathcal{A}t} L_2^2 = (\bigotimes_{P(\vec{a}) \in \mathcal{A}t} L_2)^2 = (L_2^{\mathcal{A}t})^2.$$

We define the following concept of a dependency relation for a definition. In (Denecker & Ternovska 2004a), the term *reduction relation* was used for such a relation.

Definition 7. Let Δ be a Σ-definition and $\mathcal{R}$ a L_4-valued H-interpretation which interprets some subset $A \subseteq \mathcal{A}t_H$. A binary relation $\rightsquigarrow$ on $\mathcal{A}t_H$ is a *dependency relation* of Δ in $\mathcal{R}$ iff for all L_4-valued H-interpretations $\mathcal{R}'$ of $Open_\Delta$, such that $\mathcal{R}'|_A = \mathcal{R}|_{Open_\Delta}$, for all L_4-valued H-interpretations $\mathcal{S}$ and $\mathcal{S}'$ of Σ such that $\mathcal{S}|_{Open_\Delta} = \mathcal{S}'|_{Open_\Delta} = \mathcal{R}'$, for every rule $(\forall \vec{x}\ P(\vec{t}) \leftarrow \phi)$ in Δ, and every value $\vec{a}$ for $\vec{x}$: if $\mathcal{S}|_{(\rightsquigarrow P(\vec{t}^{H[\vec{x}/\vec{a}]}))} = \mathcal{S}'|_{(\rightsquigarrow P(\vec{t}^{H[\vec{x}/\vec{a}]}))}$, then $\phi^{\mathcal{S}[\vec{x}/\vec{a}]} = \phi^{\mathcal{S}'[\vec{x}/\vec{a}]}$.

Clearly, for a binary relation $\rightsquigarrow$ and an interpretation $\mathcal{R}$ of some $A \subseteq \mathcal{A}t$, $\mathcal{R}$ is a dependency relation of a definition Δ in $\mathcal{R}$ iff for all interpretations $\mathcal{R}'$ of $Open_\Delta$, such that $\mathcal{R}'|_A = \mathcal{R}|_{Open_\Delta}$, $\rightsquigarrow$ is a dependency relation of Δ in $\mathcal{R}'$. This notion of an dependency relation for a definition coincides with the previously defined concept of a dependency relation for an operator.

Proposition 2. *If $\rightsquigarrow$ is a dependency relation of Δ in some interpretation $\mathcal{R}$ of $Open_\Delta$, then $(\rightsquigarrow) \cap Def_\Delta^2$ is a dependency relation of $\mathcal{T}_\Delta^{\mathcal{R}}$.*

(Vennekens & Denecker 2005) proves some results about dependency relations for ID-logic. Perhaps the most important one is that a definition Δ can be split into any partition which does not split up the equivalence classes associated with a dependency relation.

Definition 8. Let Δ be a definition and let $\rightsquigarrow$ be a binary relation on $\mathcal{A}t$. A partition $\{\Delta_1, \ldots, \Delta_n\}$ of Δ is a $(\rightsquigarrow)$-*partition* iff, for each $1 \leq j \leq n$, if Δ_j contains a rule defining a predicate P, then Δ_j also contains all rules defining a predicate Q, for which there exist tuples $\vec{a}, \vec{c}$ of domain elements, s.t. $Q(\vec{c}) \leq_{\rightsquigarrow} P(\vec{a})$ and $P(\vec{a}) \leq_{\rightsquigarrow} Q(\vec{c})$.

The algebraic splitting results then show that:

Theorem 2 ((Vennekens & Denecker 2005)). *Let Δ be a Σ-definition, $\mathcal{R}$ a L_4-valued interpretation of $Open_\Delta$, and $\rightsquigarrow$ a dependency relation of Δ in R. Let $\{\Delta_1, \ldots, \Delta_n\}$ be a $(\rightsquigarrow)$-partition. For each L_4-valued Σ-structure $\mathcal{S}$, such that $\mathcal{S}|_{Open_\Delta} = \mathcal{R}|_{Open_\Delta}$: $\mathcal{S} \models \Delta$ iff $\mathcal{S} \models \Delta_1 \wedge \cdots \wedge \Delta_n$.*

Let us illustrate this by looking at our example Δ_{Game}. We take the natural numbers $\mathbb{N}$ as our domain and interpret the function $-/2$ and the object symbols 0, 1, 2 in the usual way. We will define a rather coarse dependency relation for this definition, which only takes into account the predicate symbols of domain atoms. Concretely, let $\rightsquigarrow$ be the binary relation on $\mathcal{A}t_{\mathbb{N}}$, consisting of $(k \leq l) \rightsquigarrow Move(m,n)$, $(k = l) \rightsquigarrow Move(m,n)$, $Move(k,l) \rightsquigarrow Win(m)$, and $Win(m) \rightsquigarrow Win(n)$, for all $l, k, m, n \in \mathbb{N}$. Because this $\rightsquigarrow$ is dependency relation of Δ_{Game}, the above theorem shows that Δ_{Game} is equivalent to $\Delta_{Move} \wedge \Delta_{Win}$, with:

$$\Delta_{Win} = \{\forall x\ Win(x) \leftarrow \exists y\ Move(x,y) \wedge \neg Win(y).\};$$
$$\Delta_{Move} =$$
$$\left\{ \begin{array}{l} \forall x, y\ Move(x,y) \leftarrow y \geq 0 \\ \qquad\qquad\qquad\qquad \wedge ((y = x-1) \vee (y = x-2)). \end{array} \right\}$$

In the next section, a more fine-grained dependency relation will be used to further analyze Δ_{Win}.

Constructing dependency relations

So far, we have only considered dependencies at a semantical level. In this section, we develop a constructive characterization of certain dependency relations. Recall that a definition can have many dependency relations. In fact, any superset of a dependency relation is also a dependency relation. While large dependency relations, such as the one used to split Δ_{Game}, can be easy to find, they are not very informative. In this section, we present a method of constructing

smaller, more useful dependency relations. We first introduce the concept of a *base* for a formula ϕ. Intuitively, a base for ϕ is a set B of domain atoms, s.t. the truth value of all atoms in B completely determines the truth value of ϕ.

Definition 9. Let ϕ be a Σ-formula and $\mathcal{S}$ a L_4-valued Σ-structure. A set $B \subseteq \mathcal{A}t$ is a *base* for ϕ in $\mathcal{S}$ iff for all Σ-structures $\mathcal{S}'$, s.t. $\mathcal{S}'|_B = \mathcal{S}|_B$, $\phi^{\mathcal{S}'} = \phi^{\mathcal{S}}$.

Clearly, any superset of a base is also a base. The problem of finding a dependency relation for a definition Δ can be reduced to that of finding bases for bodies of rules.

Proposition 3. *Let Δ be a definition, $\mathcal{R}$ a structure interpreting at least $Open_\Delta$, and $\rightsquigarrow$ a binary relation on $\mathcal{A}t$. If for all Σ-structures $\mathcal{S}$, s.t. $\mathcal{S}|_{Open_\Delta} = \mathcal{R}|_{Open_\Delta}$, for every rule "$\forall \vec{x}\ P(\vec{t}) \leftarrow \phi$" in Δ and every tuple $\vec{c}$, the set $(\rightsquigarrow P(\vec{t}^{H[\vec{x}/\vec{c}]}))$ is a base for ϕ in $\mathcal{S}[\vec{x}/\vec{c}]$, then $\rightsquigarrow$ is a dependency relation of Δ in $\mathcal{R}$.*

We now define a method which can be used to extend any set A of domain atoms to a base for a formula ϕ. The following definition introduces both a set $Pos^A_{\mathcal{S}}(\phi)$ of domain atoms which, given some fixed interpretation $\mathcal{S}$ for the atoms in A, influence ϕ in a *positive* way (i.e., greater truth values for all $P(\vec{a}) \in Pos^A_{\mathcal{S}}(\phi)$ lead to a greater truth value for ϕ itself) and a set $Neg^A_{\mathcal{S}}(\phi)$ of domain atoms which, given the interpretation $\mathcal{S}$ for A, influence ϕ is a *negative* way (i.e., greater truth values for the atoms in $Neg^A_{\mathcal{S}}(\phi)$ lead to a lesser truth value for ϕ itself). The union $Dep^A_{\mathcal{S}}(\phi)$ of these two sets will contain all atoms which influence the truth value of ϕ, given $\mathcal{S}$.

Definition 10. Let ϕ be a formula, A a set of domain atoms, and $\mathcal{S}$ a L_4-valued Σ-structure. We define $Pos^A_{\mathcal{S}}(\phi)$ and $Neg^A_{\mathcal{S}}(\phi)$ by simultaneous induction. $Dep^A_{\mathcal{S}}(\phi)$ is used to abbreviate $Pos^A_{\mathcal{S}}(\phi) \cup Neg^A_{\mathcal{S}}(\phi)$.

- For all $P(\vec{t})$, s.t. $P(\vec{t}^{\mathcal{S}}) \in A$, $Pos^A_{\mathcal{S}}(P(\vec{t})) = Neg^A_{\mathcal{S}}(P(\vec{t})) = \{\}$;
- for all other $P(\vec{t})$, $Pos^A_{\mathcal{S}}(P(\vec{t})) = \{P(\vec{t})\}$ and $Neg^A_{\mathcal{S}}(P(\vec{t})) = \{\}$;
- for all $(\phi_1 \vee \phi_2)$, s.t. $Dep^A_{\mathcal{S}}(\phi_1) = \{\}$ and $\phi_1^{\mathcal{S}} = \mathbf{t}$ or $Dep^A_{\mathcal{S}}(\phi_2) = \{\}$ and $\phi_2^{\mathcal{S}} = \mathbf{t}$: $Pos^A_{\mathcal{S}}(\phi_1 \vee \phi_2) = Neg^A_{\mathcal{S}}(\phi_1 \vee \phi_2) = \{\}$;
- for all other $(\phi_1 \vee \phi_2)$: $Pos^A_{\mathcal{S}}(\phi_1 \vee \phi_2) = Pos^A_{\mathcal{S}}(\phi_1) \cup Pos^A_{\mathcal{S}}(\phi_2)$ and $Neg^A_{\mathcal{S}}(\phi_1 \vee \phi_2) = Neg^A_{\mathcal{S}}(\phi_1) \cup Neg^A_{\mathcal{S}}(\phi_2)$;
- for all $(\exists x\ \phi)$, s.t. for some $c \in D$, $Dep^A_{\mathcal{S}[x/c]}(\phi) = \{\}$ and $\phi^{\mathcal{S}[x/c]} = \mathbf{t}$: $Pos^A_{\mathcal{S}}(\exists x\ \phi) = Neg^A_{\mathcal{S}}(\exists x\ \phi) = \{\}$;
- for all other $(\exists x\ \phi)$: $Pos^A_{\mathcal{S}}(\exists x\ \phi) = \bigcup_{d \in D} Pos^A_{\mathcal{S}[x/d]}(\phi)$ and $Neg^A_{\mathcal{S}}(\exists x\ \phi) = \bigcup_{d \in D} Neg^A_{\mathcal{S}[x/d]}(\phi)$;
- for all $(\neg\phi)$: $Pos^A_{\mathcal{S}}(\neg\phi) = Neg^A_{\mathcal{S}}(\phi)$ and $Neg^A_{\mathcal{S}}(\neg\phi) = Pos^A_{\mathcal{S}}(\phi)$.

In a number of places, this definition distinguishes between formulas ϕ for which $Dep^A_{\mathcal{S}}(\phi) = \{\}$ and those for which $Dep^A_{\mathcal{S}}(\phi) \neq \{\}$. The intuition here is that in the first case, the truth of ϕ is already completely determined by the truth values of the atoms in A, i.e., by $\mathcal{S}|_A$.

Lemma 1. *Let ϕ be a formula, A a set of domain atoms, and $\mathcal{S}$ a L_4-valued Σ-structure. If $Dep^A_{\mathcal{S}}(\phi) = \{\}$, then A is a base for ϕ in $\mathcal{S}$.*

It follows from a simple induction over the construction given in Definition 10 that, for all $\mathcal{S}$ and $\mathcal{S}'$ s.t. $\mathcal{S}|_A = \mathcal{S}'|_A$, $Pos^A_{\mathcal{S}}(\phi) = Pos^A_{\mathcal{S}'}(\phi)$ and $Neg^A_{\mathcal{S}}(\phi) = Neg^A_{\mathcal{S}'}(\phi)$. We now show that this definition indeed captures the desired concepts.

Proposition 4. *Let ϕ be a formula, A a set of domain atoms, and $\mathcal{S} = (S_1, S_2)$, $\mathcal{S}' = (S'_1, S'_2)$ L_4-valued structures such that $(S_1, S_2)|_A = (S'_1, S'_2)|_A$. Let $P = Pos^A_{S_1 \oplus S_2}(\phi) = Pos^A_{S'_1 \oplus S'_2}(\phi)$ and $N = Neg^A_{S_1 \oplus S_2}(\phi) = Neg^A_{S'_1 \oplus S'_2}(\phi)$. If $S_1|_P \leq_t S'_1|_P$ and $S_2|_N \geq_t S'_2|_N$, then $\phi^{(S_1,S_2)} \leq_t \phi^{(S'_1,S'_2)}$.*

It follows that, for all A, $A \cup Dep^A_{\mathcal{S}}(\phi)$ is a base of ϕ in $\mathcal{S}$. We can now derive a dependency relation for a definition Δ from the bases of the bodies of its rules. This construction works by extending an *a priori* relation $\hookrightarrow$ to a dependency relation. The point of this *a priori* relation is to express dependencies from defined predicates on open predicates. Often, the simple relation $\hookrightarrow$ consisting of all $P(\vec{a}) \hookrightarrow Q(\vec{c})$ with $P(\vec{a}) \in Open_\Delta$ and $Q(\vec{c}) \in Def_\Delta$ will be used. In the following definition, we write $(\hookrightarrow \cdot)$ to denote the set $\bigcup_{P(\vec{a}) \in \mathcal{A}t}(\hookrightarrow P(\vec{a}))$ of all domain atoms that directly influence some other atom according to $\hookrightarrow$.

Definition 11. Let Δ be a definition, $\hookrightarrow$ a binary relation on $\mathcal{A}t$, and $\mathcal{S}$ a L_4-valued structure interpreting at least $(\hookrightarrow \cdot)$. We define the relation $\hookrightarrow^+_{\mathcal{S}}$ (respectively, $\hookrightarrow^-_{\mathcal{S}}$) on $\mathcal{A}t$ as: for all $P(\vec{a})$, $Q(\vec{b})$, $P(\vec{a}) \hookrightarrow^+_{\mathcal{S}} Q(\vec{b})$ iff $P(\vec{a}) \hookrightarrow Q(\vec{b})$ or there is a rule $(\forall \vec{x}\ P(\vec{t}) \leftarrow \phi) \in \Delta$, such that there exists a $\vec{c} \in H^n_D$, $\vec{t}^{\mathcal{S}[\vec{x}/\vec{c}]} = \vec{a}$ and $Q(\vec{b}) \in Pos^{(\hookrightarrow P(\vec{a}))}_{\mathcal{S}[\vec{x}/\vec{c}]}(\phi)$ (respectively, $Q(\vec{b}) \in Neg^{(\hookrightarrow P(\vec{a}))}_{\mathcal{S}[\vec{x}/\vec{c}]}(\phi)$). Finally, we define $\hookrightarrow^*_{\mathcal{S}}$ as $\hookrightarrow^+_{\mathcal{S}} \cup \hookrightarrow^-_{\mathcal{S}}$.

The following result now follows directly from Propositions 3 and 4.

Proposition 5. *Let Δ be a definition and let $\hookrightarrow$ be a binary relation on $\mathcal{A}t$, such that $(\hookrightarrow \cdot) \subseteq Open_\Delta$. Then for each structure $\mathcal{R}$ interpreting at least $(\hookrightarrow \cdot)$, $\hookrightarrow^*_{\mathcal{R}}$ is a dependency relation of Δ in $\mathcal{R}$.*

We now further analyze the definition $\Delta_{Win} = \{\forall x\ Win(x) \leftarrow \exists y\ Move(x,y) \wedge \neg Win(y)\}$. Intuitively, it is clear that, for $n \in \mathbb{N}$, $Win(n)$ is influenced by all $Win(m)$, s.t. there is a move from n to m, i.e., $Win(0)$ influences $Win(1)$ and, for $n \geq 2$, both $Win(n-1)$ and $Win(n-2)$ influence $Win(n)$. Moreover, all these influences are negative, since n is *winning* if $n-1$ or $n-2$ are *losing*.

We now show how this information can be derived using the concepts defined above. Let $\hookrightarrow$ be the binary relation on $\mathcal{A}t_{\mathbb{N}}$ consisting of $Move(n,m) \hookrightarrow Win(n)$ for all $n, m \in$

$\mathbb{N}$. Let S be a L_2-valued structure interpreting the open predicate $Move/2$. By Definition 11, for every $n \in \mathbb{N}$, the set $\{P(\vec{a}) \in \mathcal{A}t_{\mathbb{N}} \mid P(\vec{a}) \hookrightarrow^*_S Win(n)\}$ of domain atoms influencing $Win(n)$ is precisely equal to $\cup_{m\in\mathbb{N}} Dep^{A_n}_{S_n}(\phi)$, with $A_n = \{Move(n,k) \mid k \in \mathbb{N}\}$, $S_n = S[x/n]$ and $\phi = \exists y\ Move(x,y) \wedge \neg Win(y) \equiv \exists y\ \neg(\neg Move(x,y) \vee Win(y))$. Let $m \in \mathbb{N}$ and let $S^m_n = S_n[y/m]$. Because $(x,y)^{S^m_n} = (n,m)$ and $Move(n,m) \in A_n$, it is clear that $Dep^{A_n}_{S^m_n}(\neg Move(x,y)) = Dep^{A_n}_{S^m_n}(Move(x,y)) = \{\}$. From this, it now follows that if $(\neg Move(x,y))^{S^m_n} = \mathbf{t}$, i.e., $Move^S(n,m) = \mathbf{f}$, then $Dep^{A_n}_{S^m_n}(\phi) = \{\}$. This corresponds to the intuition that if there is no move from n to m (according to the chosen interpretation S of $Move/2$), m does not affect whether n is winning. On the other hand, if $Move^S(n,m) = \mathbf{t}$, i.e., there *is* a move from n to m, we see that $Pos^{A_n}_{S^m_n}(\phi) = Neg^{A_n}_{S^m_n}(Win(y))$ and $Neg^{A_n}_{S^m_n}(\phi) = Pos^{A_n}_{S^m_n}(Win(y))$. Because $Neg^{A_n}_{S^m_n}(Win(y)) = \{\}$ and $Pos^{A_n}_{S^m_n}(Win(y)) = \{Win(m)\}$, we find that in this case $Pos^{A_n}_{S^m_n}(\phi) = \{\}$ and $Neg^{A_n}_{S^m_n}(\phi) = \{Win(m)\}$. Putting all of this together, we see that $(\hookrightarrow^*_S) = (\hookrightarrow^-_S) = \{(Win(m), Win(n)) \mid Move^S(n,m) = \mathbf{t}\} \cup (\hookrightarrow)$. Moreover, if $S \models \Delta_{Move}$, this reduces to $(\hookrightarrow^-_S) = \{(Win(n-1), Win(n)) \mid n \geq 1\} \cup \{(Win(n-2), Win(n)) \mid n \geq 2\} \cup (\hookrightarrow)$.

ID-logic and mathematical induction

ID-logic aims to formalize the principle of inductive definition. As such, the relation between this logic and the kinds of inductive definitions regularly found in mathematical practice is an important research topic. (Denecker & Ternovska 2004a) showed that two known classes of definitions—monotone definitions and definitions over a well-founded order—correspond to certain classes of ID-logic definitions. Informally speaking, a definition is monotone if its associated operator is monotone w.r.t. the L_4-valued truth-order $\leq_t$ or, equivalently, the point-wise extension $\leq_\otimes$ to L_2^2 of the order $\leq_{L_2}$. A definition is a definition by induction over a well-founded order if there exists a well-founded order on its domain atoms, s.t. the truth of every atom depends only on the truth of strictly lower atoms.

Definition 12. Let Δ be a definition. Let $\mathcal{R}$ interpret at least $Open_\Delta$.

- Δ is *monotone* in $\mathcal{R}$ iff $\mathcal{T}^{\mathcal{R}}_\Delta$ is $\leq_\otimes$-monotone.
- Δ is a *definition by induction over a well-founded order* in $\mathcal{R}$ iff Δ has a dependency relation $\rightsquigarrow$ in $\mathcal{R}$, such that the transitive closure of $\rightsquigarrow$ is a well-founded strictorder.

In (Denecker & Ternovska 2004a), it was shown that the ID-logic semantics of such definitions coincides with their usual meaning. Here, we extend this analysis in two ways. We first characterize a third class, namely that of *iterated inductive definitions*, in a similar way. We then show that the results of the previous section can be used to develop syntactic criteria by which members of all three of these classes can be identified. Informally, an iterated inductive definition consists of a well-founded order of definitions, which are structured in such a way that an atom may depend either positively or negatively on an atom defined in a strictly lower level, but may only depend positively on atoms defined in the same level. As such, each of these definitionscan be reduced to a monotone definition, by fixing an interpretation for all lower levels. In our setting, this corresponds to:

Definition 13. Let Δ be a definition and $\mathcal{R}$ a structure interpreting at least $Open_\Delta$. Δ is an *iterated inductive definition* in $\mathcal{R}$ iff there exists a dependency relation $\rightsquigarrow$ of $\mathcal{T}^{\mathcal{R}}_\Delta$ such that the transitive closure of $\rightsquigarrow$ is well-founded and each component $(\mathcal{T}^{\mathcal{R}}_\Delta)^U_{\overline{P(\vec{a})}}$ is $\leq_\otimes$-monotone, with $U = wf(\mathcal{T}^{\mathcal{R}}_\Delta)|_{\prec_\rightsquigarrow \overline{P(\vec{a})}}$.

Because every constant operator is *a fortiori* monotone, the following proposition shows that this class contains all definitions over a well-founded order.

Proposition 6. *Let O be an operator on a lattice $L = \otimes_{i\in I} L_i$. Let $\rightsquigarrow$ be a dependency relation of O, s.t. the transitive closure of $\rightsquigarrow$ is a strict well-founded order. Let $i \in I$, $x \in L$ and $u = x|_{\prec_\rightsquigarrow \bar{\imath}}$. Then the component $(O)^u_{\bar{\imath}}$ is constant.*

Because an iterated inductive definition is nothing more than a sequence of monotone definitions, its models can be constructed by incrementally constructing the least fixpoints of the operators associated with each level, given all lower levels. The fact that this also holds in ID-logic, follows from the following results:

Proposition 7. *Let A be an exact approximation of an operator O, such that A is $\leq_\otimes$-monotone. Then O is a monotone operator and $wf(A) = (lfp(O), lfp(O))$.*

Proposition 8. *Let A be an exact approximation of an operator O, such that A is stratifiable w.r.t. a well-founded order $\preceq$ and each component $A_i^{wf(A)|_{\prec i}}$ of A is $\leq_\otimes$-monotone. Then $(x,y) = wf(A)$ iff for each $i \in I$, $x|_i = y|_i = lfp(O_i^{x|_{\prec i}})$.*

It follows directly that, for an iterated inductive definition Δ and structure S, $S \models \Delta$ iff for all $P(\vec{a}) \in \mathcal{A}t$, $S|_{\overline{P(\vec{a})}} = lfp((T^{(S,S)}_\Delta)^{S|_{\prec_\rightsquigarrow \overline{P(\vec{a})}}}_{\overline{P(\vec{a})}})$, with $\rightsquigarrow$ a dependency relation satisfying the conditions of Definition 13. In other words, models of such definitions can be constructed by iterating a least fixpoint construction, using the L_2-valued immediate consequence operator T^S_Δ.

The constructively defined dependency relations from the previous section can now be used to complement this semantical analysis with a more syntactical way of identifying members of these three classes of definitions.

Proposition 9. *Let Δ be a definition and let $\hookrightarrow$ be a binary relation on $\mathcal{A}t$, such that $(\hookrightarrow\ \cdot) \subseteq Open_\Delta$. Let $\mathcal{R}$ interpret at least $(\hookrightarrow\ \cdot)$. If $(\hookrightarrow^-_{\mathcal{R}}) \subseteq (\hookrightarrow)$, then Δ is a monotone definition. If the transitive closure $TC(\hookrightarrow^*_{\mathcal{R}})$ of $\hookrightarrow^*_{\mathcal{R}}$ is a well-founded strict order, then Δ is a definition by induction over a well-founded order. If $TC(\hookrightarrow^*_{\mathcal{R}})$ is well-founded and $\hookrightarrow^-_{\mathcal{R}}$ is such that for all $(P(\vec{a}), Q(\vec{c})) \in (\hookrightarrow^-_{\mathcal{R}})$, $(Q(\vec{c}), P(\vec{a})) \notin TC(\hookrightarrow^*_{\mathcal{R}})$, then Δ is an iterated inductive definition.*

For Δ_{Win}, we previously defined a relation $\hookrightarrow$, consisting of $Move(m,n) \hookrightarrow Win(m)$ with $m,n \in \mathbb{N}$, and used this to construct a dependency relation $\hookrightarrow^*_S$ for this definition. It was shown that, for any $S \models \Delta_{Move}$, $\hookrightarrow^*_S$ consists of $\{(n-1,n) \mid n \geq 1\} \cup \{(n-2,n) \mid n \geq 2\} \cup (\hookrightarrow)$. Because the transitive closure of such an $(\hookrightarrow^*_S)$ is clearly a strict well-founded order, this shows that Δ_{Win} is a definition over a well-founded order in S. Note that, for a structure S', such that there exists $n \in \mathbb{N}$ with $Move^{S'}(n,n) = \mathbf{t}$ (and therefore $S' \not\models \Delta_{Move}$), Δ_{Win} is of course not a definition over a well-founded order in S'. Indeed, in this case, $Win(n) \hookrightarrow^-_{S'} Win(n)$ and therefore none of the above criteria is satisfied.

Conclusion

We have studied the structure of definitions in ID-logic, using the basic concept of a dependency relation, both at the concrete level of ID-logic and at the algebraic level of approximation theory. These results extend work from (Denecker & Ternovska 2004a) in various ways. Firstly, we have offered a method for constructing dependency relations in ID-logic. Secondly, we extended results concerning the relation between ID-logic and inductive definitions over a well-founded order to the more general class of iterated inductive definitions. Finally, we also showed how members of both these classes can be identified.

This work is part of a larger research effort into mathematical foundations for ID-logic, which aims to lay the groundwork for more practical results. We briefly sketch the importance of our work from this point of view. Firstly, the results presented in the previous section offer additional support for the hypothesis underlying the entire knowledge representation methodology of ID-logic, namely that the "inductive definition"-construct of this logic can be understood as the formal equivalent of inductive definitions as they appear in mathematical texts. Moreover, the concepts we introduced and our taxonomy of inductive definitions will be useful when applying this methodology to a specific domain. Secondly, our results can be used to prove properties of theories in ID-logic, such as, e.g., their correctness w.r.t. a specification. Finally, we suspect they will also have an impact on algorithms for reasoning with ID-logic. While ID-logic is, in general, undecidable, there is ongoing work on identifying decidable fragments. One trivial such fragment is of course the propositional case. For this, a model generator is currently being developed. We are investigating how our work can help to improve its performance. Concretely, we are considering two complementary approaches. The first is that, during the generation of the well-founded model, knowledge about dependency relation can be exploited to avoid a number of superfluous checks and computations. The second is that, if a definition is known to belong to some specific class, then a model generation algorithm can be selected that is tailored specifically to this class. This approach seems especially promising in combination with a "preprocessing step" to transform definitions into a more manageable form.

This process was already illustrated by our treatment of the example Δ_{Game}. Even a coarse dependency relation already shows that this can be split into the conjunction $\Delta_{Move} \wedge \Delta_{Win}$. Now, Δ_{Move} is a non-inductive definition and its well-founded model can therefore be found as a classical model of its completion, which means we can simply use a SAT-solver for this task. Once the model of Δ_{Move} is known, a better dependency relation for Δ_{Win} can be constructed, which allows us to conclude that this is now a definition over a well-founded order. An algorithm specific to this class of definitions can then be applied.

As already mentioned in the introduction, the kind of work presented here has a rich tradition in logic programming. The use of constructs similar to dependency relations to analyze the structure of programs, identify interesting subclasses of programs, and clarify semantical issues dates back at least as far as (Apt, Blair, & Walker 1988). More recently, work such as (Erdem & Lifschitz 2003) on the topic of *tight logic programs* and variants thereof performs an analysis of Answer Set Programs that is very similar to our analysis for ID-logic, studying criteria that suffice to conclude that a program belongs to a certain specific class, for which interesting properties hold.

References

Apt, K.; Blair, H.; and Walker, A. 1988. Towards a theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming*.

Denecker, M., and Ternovska, E. 2004a. Inductive situation calculus. In *Proc. KR '04*, 545–553. AAAI Press.

Denecker, M., and Ternovska, E. 2004b. A logic of nonmonotone inductive definitions and its modularity properties. In *Proc. LPNMR '04*.

Denecker, M.; Marek, V.; and Truszczynski, M. 2000. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*. Kluwer Academic Publishers. 127–144.

Denecker, M.; Marek, V.; and Truszczynski, M. 2003. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence* 143(1):79–122.

Denecker, M. 1998. The well-founded semantics is the principle of inductive definition. In *Proc. JELIA'98*, volume 1489 of *LNAI*, 1–16.

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *Theory and Practice of Logic Programming* 3:499–518.

Gelder, A. V.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38(3):620–650.

Vennekens, J., and Denecker, M. 2005. An algebraic account of modularity in ID-logic. In *Proc. LPNMR'05*.

Vennekens, J.; Gilis, D.; and Denecker, M. 2005. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM TOCL*. To appear.

1.11 Well-Founded semantics for Semi-Normal Extended Logic Programs

Well-Founded semantics for Semi-Normal Extended Logic Programs

Martin Caminada*
Utrecht University

Abstract

In this paper we present a new approach for applying well-founded semantics to extended logic programs. The main idea is not to fundamentally change the definition of well-founded semantics (as others have attempted) but rather to define a few restrictions on the content of the extended logic program, that make it possible to apply "traditional" well-founded semantics in a very straightforward way.

Introduction

Well-founded semantics (van Gelder, Ross, & Schlipf 1991) has originally been stated as an alternative for stable model semantics in normal logic programs. Due to its skeptical nature, it has sometimes been regarded as an easily computable lower bound for the more credulous stable model semantics. At the same time, well-founded semantics avoids some of the problems of stable model semantics, in which relatively small pieces of information (like a rule $\mathtt{a} \leftarrow \mathtt{not\ a}$) can cause the total absence of stable models.

With the emergence of extended logic programming (Gelfond & Lifschitz 1991), several researchers have attempted to apply well-founded semantics to extended logic programs (Sakama 1992; Brewka 1996). The introduction of strong negation, however, introduces additional problems not present in normal (non-extended) logic programming. In this paper, we approach the issue of how to apply well-founded semantics for extended logic programs not by giving another complex and advanced specification of what well-founded semantics for extended logic programs should look like, but instead we state a few restrictions on the *content* of the extended logic programs. We then show that under these restrictions, a relatively simple and straightforward definition of well-founded semantics yields a decent and unproblematic well-founded model.

Basic Definitions

A *program* considered in this paper is en *extended logic program* (ELP) (Gelfond & Lifschitz 1991) containing rules with weak as well as strong negation.

*This work has been supported by the EU ASPIC project.

Definition 1. *An* extended logic program P *is a finite set of clauses of the form:*

$$\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m} (n \geq 0,\ m \geq 0)$$

where each $\mathtt{c}$, $\mathtt{a_i}$ *and* $\mathtt{b_j}$ *is a positive/negative literal and* not *stands for negation as failure. In the above rule,* $\mathtt{b_j}(1 \leq j \leq m)$ *is called a* weakly negated literal. *The literal* $\mathtt{c}$ *is called the* head *of the rule, and the conjunction* $\mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m}$ *is called the* body *of the rule. A rule is called* strict *iff it contains no weakly negated literals (that is, if* $m = 0$*), otherwise, the rule is* defeasible.

Notice that the head of a rule is never empty, although the body can be empty. If $\mathtt{l}$ is a literal, then we identify $\neg\neg\mathtt{l}$ with $\mathtt{l}$. If P is an extended logic program, then $strict(P)$ stands for the set of strict rules in P, and $defeasible(P)$ stands for the set of defeasible rules in P.

The *closure* of a set of strict rules consists of all literals that can be derived with it, as is stated in the following definition.

Definition 2. *Let* S *be a set of strict rules. We define* $Cl(S)$ *as the smallest set of literals such that if* S *contains a rule* $\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$ *and* $\mathtt{a_1}, \ldots, \mathtt{a_n} \in Cl(S)$ *then* $\mathtt{c} \in Cl(S)$.

If S is a set of strict rules and L a set of literals, then we write $Cl(S \cup L)$ as an abbreviation of $Cl(S \cup \{\mathtt{l} \leftarrow \mid\ \mathtt{l} \in L\})$.

Definition 3. *We say that a set of literals* L *is* consistent *iff* L *does not contain a literal* $\mathtt{l}$ *and its negation* $\neg\mathtt{l}$*. We say that a set of strict rules* S *is* consistent *iff* $Cl(S)$ *is consistent.*

The idea of P^L (the Gelfond-Lifschitz reduct of a logic program P under a set of literals L) is to remove each rule from P that is "defeated" by L (that is, to remove each rule containing a weakly negated literal in L) and then from the remaining rules to remove all remaining occurrences of weak negation.

Definition 4. *Let* P *be an extended logic program and let* L *be a set of literals. We define* P^L *as* $\{\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n} \mid \mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{not\ b_1}, \ldots, \mathtt{not\ b_m} \in P(n, m \geq 0)$ *and* $\neg\exists \mathtt{b_j}(1 \leq j \leq m) : \mathtt{b_j} \in L\}$.

Well-founded semantics (van Gelder, Ross, & Schlipf 1991) is a concept originally proposed for non-extended logic programs. As its original description is quite complex, we will use the following definition instead (inspired by (Brewka 1996)).

Definition 5. *Let P be an extended logic program and L be a set of literals. We define $\gamma(L)$ (the standard stable operator) as $Cl(P^L)$. We define $\Gamma(L)$ as $\gamma(\gamma(L))$. The* well-founded model *of P is the smallest fixpoint of Γ.*

The Problem

Well-founded semantics (WFS) has been applied successfully in non-extended logic programs (Dix 1995a; 1995b). Applying WFS for extended logic programs, however, introduces the problem that the well-founded model is not guaranteed to be consistent. Consider the following example, taken from (Caminada & Amgoud 2005).

Example 1.
"John wears something that looks like a wedding ring."
"John parties with his friends until late."
"Someone wearing a wedding ring is usually married."
"A party-animal is usually a bachelor."
"A married person, by its definition, has a spouse."
"A bachelor, by definition, does not have a spouse."
These sentences are represented by the program P:

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{r} & \leftarrow & & \mathtt{p} & \leftarrow & \\ \mathtt{m} & \leftarrow & \mathtt{r}, \mathtt{not}\,\neg\mathtt{m} & \mathtt{b} & \leftarrow & \mathtt{p}, \mathtt{not}\,\neg\mathtt{b} \\ \mathtt{hs} & \leftarrow & \mathtt{m} & \neg\mathtt{hs} & \leftarrow & \mathtt{b}. \end{array}$$

For example 1, applying the unaltered version of WFS yields a well-founded model of $\{\mathtt{r}, \mathtt{p}, \mathtt{m}, \mathtt{b}, \mathtt{hs}, \neg\mathtt{hs}\}$, which is inconsistent.

To cope with this problem, many approaches have been stated. Brewka, for instance, proposes to define the function $\Gamma(L)$ not as $\gamma(\gamma(L))$ but as $\gamma(Cn(\gamma(L)))$, where $Cn(L)$ is L if L is consistent, or Lit if L is not consistent (Brewka 1996). Another approach would be to apply paraconsistent reasoning, as for instance has been done in (Sakama 1992).

An alternative approach would be not to redefine the *semantics* of an ELP, but instead to state some additional conditions on the *content* of the extended logic program. The above example, for instance, would yield a perfectly acceptable outcome if the rules $\neg\mathtt{m} \leftarrow \neg\mathtt{hs}$ and $\neg\mathtt{b} \leftarrow \mathtt{hs}$ were added (which are essentially the contraposed versions of $\mathtt{hs} \leftarrow \mathtt{m}$ and $\neg\mathtt{hs} \leftarrow \mathtt{b}$). In that case, the well-founded model would be $\{\mathtt{r}, \mathtt{p}\}$. This approach would be quite similar to the work that Caminada and Amgoud have done in the field of formal argumentation, where similar difficulties occur (Caminada & Amgoud 2005).

Logic Programming as Argumentation

In this section, we will state some theory that allows us to link logic programming to formal argumentation. Using this theory, we will be able to apply the solution of (Caminada & Amgoud 2005) in the context of extended logic programming.

The first thing to do is to define the set of arguments and the defeat relation, given an (extended) logic program P. We choose a form of arguments that is different from (Dung 1995) and better suited to our purpose.

Definition 6. *Let P be an extended logic program.*

- *An* argument *A based on P is a finite tree of rules from P such that each node (of the form $\mathtt{c} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_n, \mathtt{not}\,\mathtt{b}_1, \ldots, \mathtt{not}\,\mathtt{b}_m$ with $n \geq 0$ and $m \geq 0$) has exactly n children, each having a different head $\mathtt{a}_i \in \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$. The* conclusion *of A ($\mathtt{Conc}(A)$) is the head of its root.*
- *We say that an argument A_1* defeats *an argument A_2 iff A_1 has conclusion $\mathtt{c}$ and A_2 has a rule containing $\mathtt{not}\,\mathtt{c}$.*

We define $Arguments_P$ as the set of arguments that can be constructed using P, and $Defeat_P$ as the defeat relation under P. Let $Args \subseteq Arguments_P$. We define $\mathtt{Concs}(Args)$ as $\{\mathtt{Conc}(A) \mid A \in Args\}$.

We say that argument A is a *subargument* of argument B iff A is a subtree of B. We say that argument A is a *direct subargument* of argument B iff A is a subtree of B and there does not exist an argument C such that $C \neq A$, $C \neq B$, C is a subtree of B, and A is a subtree of C.

Definition 7. *We say that:*

- *a set of arguments $Args$ is* conflict-free *iff $Args$ does not contain two arguments A and B such that A defeats B*
- *a set of arguments $Args$* defends *an argument A iff for each argument B that defeats A, $Args$ contains an argument C that defeats B.*

Definition 8. *Let $Args$ be a set of arguments. We define $f(Args)$ as $\{A \mid Args$ does not contain an argument that defeats $A\}$ and $F(Args)$ as $f(f(Args))$.*

$F(Args)$ can be seen as the set of arguments that are defended by $Args$ (Dung 1995).

Lemma 1. *Let P be an extended logic program and let E be the smallest fixpoint of F under P. E is conflict-free.*

Proof. As E is the smallest fixpoint of F under P, it holds that (Dung 1995) $E = \cup_{i=0}^{\infty} F^i(\emptyset)$. Suppose that E is not conflict-free. As F is a monotonic function and $F^0(\emptyset) = \emptyset$, there must be some smallest i ($i \geq 0$) such that $F^i(\emptyset)$ is conflict-free but $F^{i+1}(\emptyset)$ is not conflict-free. From definition 7 it then follows that $F^{i+1}(\emptyset)$ contains two arguments A and B such that A defeats B. The fact that A defeats B and $B \in F^{i+1}(\emptyset)$ means that there is an argument $C \in F^i(\emptyset)$ that defeats A. The fact that C defeats A and $A \in F^{i+1}(\emptyset)$ means that there is an argument $D \in F^i(\emptyset)$ that defeats C. But then $F^i(\emptyset)$ would not be conflict-free. Contradiction. □

The following property follows from definition 6 and 2.

Property 1. *Let S be a set of strict rules and $\mathtt{l}$ be a literal. It holds that $\mathtt{l} \in Cl(S)$ iff there exists an argument A, based on S, such that $\mathtt{Conc}(A) = \mathtt{l}$.*

The following property follows from definition 4 and 6.

Property 2. *Let P be an extended logic program and L be a set of literals. There exists an argument A, based on P^L, with $\mathtt{Conc}(A) = \mathtt{l}$ iff there exists an argument B, based on P, with $\mathtt{Conc}(B) = \mathtt{l}$, such that B does not contain a weakly negated literal $\mathtt{k} \in L$.*

The function γ is actually quite similar to the function f, as is stated in the following theorem.

Theorem 1. *Let L be a set of literals and $Args$ be a set of arguments. If $L = \texttt{Concs}(Args)$ then $\gamma(L) = \texttt{Concs}(f(Args))$.*

Proof. We need to prove two things:

1. $\gamma(L) \subseteq \texttt{Concs}(f(Args))$
   Let $\texttt{l} \in \gamma(L)$. This, by definition 5, means that $\texttt{l} \in Cl(P^L)$. From property 1 it follows that there exists an argument (A), based on P^L, with $\texttt{Conc}(A) = \texttt{l}$. Then, according to property 2, there exists an argument (B), based on P, with $\texttt{Conc}(B) = \texttt{l}$, such that B does not contain a weakly negated literal $\texttt{k} \in L$. As $L = \texttt{Concs}(Args)$, the argument B is not defeated by $Args$. Therefore, $B \in f(Args)$. As B has conclusion $\texttt{l}$ it holds that $\texttt{l} \in \texttt{Concs}(f(Args))$
2. $\texttt{Concs}(f(Args)) \subseteq \gamma(L)$
   Let $\texttt{l} \in \texttt{Concs}(f(Args)) \subseteq \gamma(L)$. This means that $f(Args)$ contains some argument (say B) with conclusion $\texttt{l}$. That is, there exists an argument (B) with conclusion $\texttt{l}$ that is not defeated by $Args$. From property 2 it then follows that there exists an argument A, based on P^L (as $L = \texttt{Concs}(Args)$), with $\texttt{Conc}(A) = \texttt{l}$. This, by property 1, means that $\texttt{l} \in Cl(P^L)$, which by definition 5 means that $\texttt{l} \in \gamma(L)$.

□

The following theorem states that the well-founded model of a program P coincides with the conclusions of the grounded extension (Dung 1995) of the argument-interpretation of P.

Theorem 2. *Let P be an extended logic program. The grounded extension GE of $\langle Arguments_P, Defeat_P \rangle$ coincides with the smallest fixpoint (WFM) of Γ. That is: $concs(GE) = WFM$.*

Proof. From theorem 1 it follows that, if $L = \texttt{Concs}(Args)$, then $\gamma(\gamma(L)) = \texttt{Concs}(f(f(Args)))$, so $\Gamma(L) = \texttt{Concs}(F(L))$. Therefore, the smallest fixpoint of Γ is equal to the conclusions of the smallest fixpoint of F, which is the grounded extension. □

Semi-Normal Extended Logic Programs

In this section, we define some restrictions on an extended logic program. An extended logic program that satisfies these restrictions is called a *semi-normal* extended logic program (a term inspired by semi-normal default theories). We then show that a semi-normal extended logic program avoids problems like illustrated in example 1 by always having a consistent well-founded model.

Definition 9. *Let s_1 and s_2 be strict rules. We say that s_2 is a* transposition *of s_1 iff:*
$s_1 = \texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_\texttt{n}$ *and*
$s_2 = \neg\texttt{a}_\texttt{i} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_{\texttt{i}-1}, \neg\texttt{c}, \texttt{a}_{\texttt{i}+1}, \ldots, \texttt{a}_\texttt{n}$ *$(1 \leq i \leq n)$.*

The intuition behind transposition can be illustrated by translating a strict rule $\texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_\texttt{n}$ to a material implication $\texttt{c} \subset \texttt{a}_1 \wedge \cdots \wedge \texttt{a}_\texttt{n}$. This implication is equivalent to $\neg\texttt{a}_\texttt{i} \subset \texttt{a}_1 \wedge \cdots \wedge \texttt{a}_{\texttt{i}-1} \wedge \neg\texttt{c} \wedge \texttt{a}_{\texttt{i}+1} \wedge \cdots \wedge \texttt{a}_\texttt{n}$, which is again translated to $\neg\texttt{a}_\texttt{i} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_{\texttt{i}-1}, \neg\texttt{c}, \texttt{a}_{\texttt{i}+1}, \ldots, \texttt{a}_\texttt{n}$. Notice that, when $n = 1$, transposition coincides with classical contraposition.

Definition 10. *A defeasible rule is* semi-normal *iff it is of the form*
$\texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_\texttt{n}, \texttt{not}\ \texttt{b}_1, \ldots, \texttt{not}\ \texttt{b}_\texttt{m}, \texttt{not}\ \neg\texttt{c}$.

Definition 11. *An extended logic program P is called* semi-normal *iff:*

1. *$strict(P)$ is consistent,*
2. *$strict(P)$ is closed under transposition, and*
3. *$defeasible(P)$ consists of semi-normal rules only*

If A is an argument, then the *depth* of A is the number of nodes on the longest root-originated path in A. If A is an argument and r is a rule in A then the *depth of r in A* is the number of nodes on the shortest path from the root to a node labeled with r.

Lemma 2. *Let P be a semi-normal extended logic program, Ass (the assumptions) be a nonempty set of strict rules with empty antecedents $\{\texttt{a}_1 \leftarrow, \ldots, \texttt{a}_\texttt{n} \leftarrow\}$ and A an argument with conclusion $\texttt{c}$ based on $strict(P) \cup Ass$, such that A contains an assumption $\texttt{a}_\texttt{i} \leftarrow$ $(1 \leq i \leq n)$ that does not occur in P. There exists an argument B, based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$ such that B has a conclusion $\neg\texttt{a}_\texttt{i}$.*

Proof. We prove this by induction on the depth of A.

basis Let's assume that the depth of A is 1. In that case, A consists of a single rule, which must then have an empty antecedent. Therefore, the root of A must be $\texttt{c} \leftarrow$. It then follows that $\texttt{c} = \texttt{a}_\texttt{i}$. Therefore, there exists an argument (A itself) based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$ that has conclusion $\neg\texttt{a}_\texttt{i}$.

step Suppose the above lemma holds for all strict arguments of depth $\leq j$. We now prove that it also holds for all strict arguments of depth $j + 1$. Let A be an argument of depth $j + 1$, based on $strict(P) \cup Ass$, with conclusion $\texttt{c}$. Let $\texttt{c} \leftarrow \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_m)$ be the root of A. Let A_i be a direct subargument of A that contains the assumption $a_i \leftarrow$. Because the set of strict rules in P is closed under transposition, there exists a rule $\neg\texttt{Conc}(A_i) \leftarrow \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_{i-1}), \neg c, \texttt{Conc}(A_{i+1}), \ldots, \texttt{Conc}(A_m)$. The fact that A_i has a depth $\leq j$ means that we can apply the induction hypothesis. That is, there exists an argument (say B'), based on $strict(P) \cup Ass \cup \{\neg\texttt{Conc}(A_i) \leftarrow\}$, with conclusion $\neg\texttt{a}_\texttt{i}$. Now, in B', substitute $\neg\texttt{Conc}(A_i) \leftarrow$ by the subargument $\neg\texttt{Conc}(A_i) \leftarrow A_1, \ldots, A_{i-1}, \neg c, A_{i+1}, \ldots A_m$. The resulting argument (call it B) is a strict argument, based on $strict(P) \cup Ass \cup \{\neg\texttt{c} \leftarrow\}$, with conclusion $\neg\texttt{a}_\texttt{i}$.

□

Theorem 3. *Let $\langle Arguments_P, Defeat_P \rangle$ be an argumentation framework built from a semi-normal extended logic program P, and let E be the smallest fixpoint of F. It holds that $\texttt{Concs}(E)$ is consistent.*

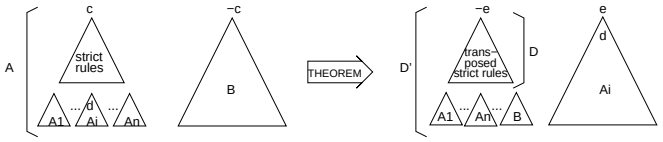


Figure 1: The working of theorem 3

Proof. Let E be the grounded extension of $\langle Arguments_P, Defeat_P \rangle$. Suppose the conclusions of E are not consistent. Then E contains an argument (say A) with conclusion $\mathtt{c}$ and an argument (say B) with conclusion $\neg\mathtt{c}$. As $strict(P)$ is consistent, at least one of these two arguments must contain a defeasible rule. Let us, without loss of generality, assume that A contains at least one defeasible rule. Let d be a defeasible rule in A that has minimal depth. Notice that the depth of d must be at least 1, for if d were the top-rule of A, then B would defeat A and E would not be conflict-free (which conflicts with lemma 1). It now holds that every rule in A with a smaller depth than d is a strict rule (see also figure 1). Let A_i be a subargument of A that has d as its top-rule. We will now prove that there exists an argument (D') in E that defeats A_i. Let $A_1, \ldots, A_n$ be the subarguments of A that are at the same level as A_i in A. Lemma 2 tells us that with the conclusions of $A_1, \ldots, A_n, B$ it is possible to construct an argument with a conclusion that is the opposite of the conclusion of A_i. Call this argument D. Now, let D' be equal to D, but with the assumptions $\mathtt{Conc}(A_1) \leftarrow, \ldots, \mathtt{Conc}(A_n) \leftarrow, \mathtt{Conc}(B) \leftarrow$ substituted by the underlying arguments $A_1, \ldots, A_n, B$. It holds that $D' \in E$ (this is because each defeater of D' is also a defeater of $A_1, \ldots, A_n, B \in E$, and the fact that E is a fixpoint of F means it defends itself against this defeater, which means that $D' \in E$). D', however, defeats A_i on d, so the fact that $D', A_i \in E$ means that E is not conflict-free, and (lemma 1) also no fixpoint of F. Contradiction. □

Theorem 4. *Let P be a semi-normal extended logic program. The smallest fixpoint WFM (the well-founded model) of Γ is consistent.*

Proof. This follows directly from theorem 2 and theorem 3. □

Discussion

Many scholars in the field of defeasible reasoning distinguish two types of abstract rules: *strict rules* and *defeasible rules* (Pollock 1992; Nute 1994; Prakken & Sartor 1997; García & Simari 2004). A strict rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \rightarrow \mathtt{b}$ basically means that if $\mathtt{a_1}, \ldots, \mathtt{a_n}$ hold, then it is *without any possible exception* also the case that $\mathtt{b}$ holds. A defeasible rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \Rightarrow \mathtt{b}$ basically means that if $\mathtt{a_1}, \ldots, \mathtt{a_n}$ hold, then it is *usually* (or *normally*) the case that $\mathtt{b}$ holds.

One possible application of strict rules is to describe things that hold by definition (like ontologies). For instance, a cow is by definition a mammal and someone who is married by definition has a spouse. For this kind of rules, it appears that transposition is quite naturally applicable. If from $\mathtt{a_1}, \ldots, \mathtt{a_n}$ it follows without any possible exception that $\mathtt{b}$, then it also holds that from $\mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{b}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$ it follows without any possible exception that $\neg\mathtt{a_i}$.

In essence, one could say that the problems of example 1 are caused by the fact that two conclusions ($\mathtt{m}$ and $\mathtt{b}$) are conflicting (as $\mathtt{m}$ implies $\mathtt{hs}$, and $\mathtt{b}$ implies $\neg\mathtt{hs}$) but the standard entailment of ELP is too weak to discover this conflict. Transposition (for strict rules) can thus be seen as a way of strengthening the entailment, so that this kinds of hidden conflicts become explicit, and therefore manageable.

Some formalisms for defeasible reasoning, like (Pollock 1992; 1995), have strict rules that coincide with classical (propositional or first order) reasoning. That is, there exists a strict rule $\mathtt{a_1}, \ldots, \mathtt{a_n} \rightarrow \mathtt{b}$ iff $\mathtt{a_1}, \ldots, \mathtt{a_n} \vdash \mathtt{b}$. In such a formalism, example 1 could be represented by the defeasible rules $\mathtt{r} \Rightarrow \mathtt{m}$ and $\mathtt{p} \Rightarrow \mathtt{b}$ and by the propositions $\mathtt{r}$, $\mathtt{p}$, $\mathtt{m} \supset \mathtt{hs}$ and $\mathtt{b} \supset \neg\mathtt{hs}$. Using these propositions one can then construct the strict rules $\mathtt{m}, (\mathtt{m} \supset \mathtt{hs}) \rightarrow \mathtt{hs}$ and $\mathtt{b}, (\mathtt{b} \supset \neg\mathtt{hs}) \rightarrow \neg\mathtt{hs}$, as well as the strict rules $\neg\mathtt{hs}, (\mathtt{m} \supset \mathtt{hs}) \rightarrow \neg\mathtt{m}$ and $\mathtt{hs}, (\mathtt{b} \supset \neg\mathtt{hs}) \rightarrow \neg\mathtt{b}$. These rules can be used not only to construct arguments for $\mathtt{m}$ and $\mathtt{b}$ but also to construct the much needed counterarguments deriving $\neg\mathtt{m}$ and $\neg\mathtt{b}$. By basing strict rules on classical entailment, Pollock is able to specify a formalism that avoids many of the difficult issues that have been plaguing the field of extended logic programming.

It is not difficult to see that transposition is a valid principle in classical logic (from $\mathtt{a_1}, \ldots, \mathtt{a_n} \vdash \mathtt{b}$ it follows that $\mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{b}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n} \vdash \neg\mathtt{a_i}$). In general, the set of strict rules generated by classical entailment satisfies many interesting properties. With transposition we have isolated the specific property of classical logic that is actually needed to avoid problems like illustrated by example 1. We simply apply the part of classical logic that we actually need, without having to go through the complexities of having to implement a full-blown classical logic theorem prover to generate the set of strict rules, as is for instance done in (Pollock 1995). The main cost of our approach is in generating the transpositions of the strict rules. For each strict rule, n transpositions are generated, where n is the number of literals in the body of the rule.

As for the defeasible rules, Pollock distinguishes two ways in which these can be argued against: rebutting and undercutting (Pollock 1992; 1995). Rebutting essentially means deriving the opposite consequent (head) of the rule, whereas undercutting basically means that there is some additional information under which the antecedent (body) of the rule is no longer a reason for the consequent (head) of the rule. For instance, suppose that we have the defeasible rule that an object that looks red usually is red. A rebutter would be that the object is not red, because it is known to be blue. An undercutter would be that the object is illuminated by a red light. This is not a reason for it *not* being red, but merely means that the fact that it looks red can no longer be regarded as a valid reason for it actually being red. Thus, rebutting attacks the consequent (head) of a rule, whereas undercutting attacks merely the connection between the antecedent (body) and the consequent (head) of a rule. Pollock claims, based on his philosophical work regarding episte-

mology, all forms of defeat can be reduced to rebutting and undercutting (Pollock 1992). This observation is important, as both of these forms of defeat can be modeled using semi-normal defeasible rules in extended logic programs.

Many problems in logic programming are caused by specific logic programs containing anomalous information (a rule like $\texttt{a} \leftarrow \texttt{not a}$ could for instance cause the absence of stable models). If one wants to apply standard and relatively straightforward semantics then one needs to make sure that a logic program does not contain such anomalies. If one provides anomalous input (like stating that a married person always has a spouse, without stating that someone who does not have a spouse is not married, using a formalism (ELP) that is not powerful enough to make this inference itself) then one should not be surprised that the outcome (the well-founded model) is anomalous as well. For reasons described above, we think that that the concept of semi-normal extended logic programs can serve as a quite natural and reasonable restriction of which programs can be regarded to be free of anomalies.

Quality Postulates

One way to evaluate the different approaches for providing a suitable semantics for ELP is by providing quality postulates (Caminada & Amgoud 2005). The idea is to state a number of general properties that should be satisfied by any formalism for defeasible reasoning, including ELP. In (Caminada & Amgoud 2005; ASPIC-consortium 2005) the following quality postulates have been stated:

- direct consistency. Let P be an extended logic program such that $strict(P)$ is consistent, and let M be a model of P (under some specified semantics). It must hold that M is consistent.
- closedness. Let P be an extended logic program and let M be a model under P (under some specified semantics). It must hold that $Cl(strict(P) \cup M) = M$.
- indirect consistency. Let P be an extended logic program such that $strict(P)$ is consistent, and let M be a model of P (under some specified semantics). It must hold that $Cl(strict(P) \cup M)$ is consistent.

The quality postulate of direct consistency is quite straightforward and is satisfied by most formalisms that we know of. The quality postulate of closedness basically states that, as far as the strict rules are concerned, the model is "complete". The quality postulate of indirect consistency does by itself not require that the model is closed under the strict rules, but instead requires the more modest property that if one would compute the closure of the model under the strict rules, the result would at least not contain any inconsistencies.

The above three quality postulates are not completely independent. Indirect consistency, for instance, implies direct consistency. Similarly, closedness and direct consistency imply indirect consistency.

To illustrate the value of the above three quality postulates, consider a person who knows a set of strict and defeasible rules, encodes these as a semi-normal extended logic program and then examines a model generated by an ELP inference engine. If the ELP inference engine would (in example 1) provide a model containing $\texttt{m}$ but not containing $\texttt{hs}$ (thus violating closedness) then the user may conclude that the ELP inference engine apparently "forgot" something. Worse yet, if the ELP inference engine provides a model containing $\texttt{m}$ and $\texttt{b}$ (thus violating indirect consistency) then the user may reason like: "My inference engine says that $\texttt{m}$, and I know that from $\texttt{m}$ it always follows that $\texttt{hs}$, therefore $\texttt{hs}$. My inference engine also says that $\texttt{b}$ and I know that from $\texttt{b}$ it always follows that $\neg\texttt{hs}$, therefore $\neg\texttt{hs}$." It is our view that, from an agent perspective, a formalism that does not satisfy indirect consistency cannot be used to generate the beliefs of an agent, as we think that an agent should never run into inconsistencies once it starts to do additional reasoning on its own beliefs.

Although ELP-models should ideally be closed under the strict rules of P, they should not necessarily be closed under the defeasible rules of P. If $\texttt{a}$ is given and there exists a rule "if $\texttt{a}$ then normally $\texttt{b}$", then one cannot simply derive $\texttt{b}$ since the situation may not be normal. The quality postulate of closedness is thus only relevant with respect to strict rules.

A fourth quality postulate that has, as far as we know, not been published earlier is that of crash-resistancy:

- crash-resistancy. There should not exist an extended logic program P, with $strict(P)$ consistent, such that for *any* extended logic program P', with $strict(P')$ consistent, that does not share any atoms with P, it holds that P has the same models (under some specific semantics) as $P \cup P'$.

Crash-resistancy basically states that it should not be possible for an extended logic program to contain some pieces of information (P) that makes other totally unrelated pieces of information (P') totally irrelevant when added.

The above four quality postulates are violated by various approaches that aim to provide extended logic programs with a suitable semantics. Indirect consistency, for instance, is problematic in approaches that are based on paraconsistent reasoning. When the approach of, for instance, (Sakama 1992) is applied to example 1, it produces a well-founded model $\langle\{\texttt{r}, \texttt{p}, \texttt{m}, \texttt{b}, \texttt{hs}, \neg\texttt{hs}\}, \{\neg\texttt{r}, \neg\texttt{p}, \neg\texttt{m}, \neg\texttt{b}\}\rangle$. Using Ginsberg's 7-valued default bilattice, this means that only $\texttt{r}$, $\texttt{p}$, $\texttt{m}$ and $\texttt{b}$ (but not $\texttt{hs}$ or $\neg\texttt{hs}$) are considered true, thus violating closedness and indirect consistency.

Brewka's approach to well-founded semantics (Brewka 1996), on the other hand, violates direct consistency as well as crash-resistancy. In example 1, $strict(P)$ is consistent, but Brewka's approach nevertheless yields the inconsistent set Lit, which violates direct consistency. As the outcome of Lit is obtained even when one adds syntactically totally unrelated rules to P, crash-resistancy is violated as well.

The quality postulate of crash-resistancy is violated by the stable model semantics of answer set programming, where a simple rule like $\texttt{a} \leftarrow \texttt{not a}$ yields no stable models at all, regardless of what additional (unrelated) information is contained in the logic program. A common opinion in the ELP-research community is that programs that have no stable models are by definition anomalous and unnatural. We

hereby would like to argue against this view. Consider a situation in where persons are usually believed in what they say, unless information of the contrary is available (rebut) or the person is known to be unreliable (undercut). Now consider the following three persons, who give the following statements:

- Bert: "Ernie is unreliable."
- Ernie: "Elmo is unreliable."
- Elmo: "Bert is unreliable."

This would correspond with the following extended logic program:

- $\texttt{bert\_says\_u\_ernie} \leftarrow$
- $\texttt{u\_ernie} \leftarrow \texttt{bert\_says\_u\_ernie}, \texttt{not}\ \neg\texttt{u\_ernie}, \texttt{not}\ \texttt{u\_bert}$
- $\texttt{ernie\_says\_u\_elmo} \leftarrow$
- $\texttt{u\_elmo} \leftarrow \texttt{ernie\_says\_u\_elmo}, \texttt{not}\ \neg\texttt{u\_elmo}, \texttt{not}\ \texttt{u\_ernie}$
- $\texttt{elmo\_says\_u\_bert} \leftarrow$
- $\texttt{u\_bert} \leftarrow \texttt{elmo\_says\_u\_bert}, \texttt{not}\ \neg\texttt{u\_bert}, \texttt{not}\ \texttt{u\_elmo}$

It is perfectly possible for a situation to occur in which three persons, sitting in a circle, claim their direct neighbour is unreliable. How this conflict should be dealt with is an issue open for discussion, but it should at least not cause the hearer to enter a state of total ignorence in which also all other entailment is completely blocked. It is our opinion, also for reasons described in (Dung 1995) that the problems of stable model semantics are very often caused by the nature of the semantics itself, and not by an "anomalous" extended logic program.

Summary and Conclusions

One of the advantages of the approach as sketched in the current paper is that it satisfies each of the quality postulates direct consistency, indirect consistency, closedness and crash-resistancy. Furthermore, it does so without the need of an advanced semantics that is complex and potentially difficult to understand. Although the approach only works for the somewhat restricted notion of semi-normal extended logic programs, we believe that these restrictions are in essence quite natural and can be given a decent philosophical justification.

References

ASPIC-consortium. 2005. Deliverable D2.5: Draft formal semantics for ASPIC system.

Brewka, G. 1996. Well-founded semantics for extended logic programs with dynamic preferences. *J. Artif. Intell. Res. (JAIR)* 4:19–36.

Caminada, M., and Amgoud, L. 2005. An axiomatic account of formal argumentation. In *Proceedings of the AAAI-2005*, 608–613.

Dix, J. 1995a. A classification theory of semantics of normal logic programs: I. strong properties. *Fundam. Inform.* 22(3):227–255.

Dix, J. 1995b. A classification theory of semantics of normal logic programs: Ii. weak properties. *Fundam. Inform.* 22(3):257–288.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77:321–357.

García, A., and Simari, G. 2004. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4):365–385.

Nute, D. 1994. Defeasible logic. In Gabbay, D.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford: Clarendon Press. 253–395.

Pollock, J. L. 1992. How to reason defeasibly. *Artificial Intelligence* 57:1–42.

Pollock, J. L. 1995. *Cognitive Carpentry. A Blueprint for How to Build a Person*. Cambridge, MA: MIT Press.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7:25–75.

Sakama, C. 1992. Extended well-founded semantics for paraconsistent logic programs. In *FGCS*, 592–599.

van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

2 Theory of NMR and Uncertainty

Nonmonotonic and uncertain reasoning are both aiming at making optimal use of available information even if it is neither complete nor certain. Whereas the former is influenced mainly by symbolic or qualitative logics, the latter often uses numbers such as probabilities or possibilities to specify degrees of uncertainty. For intelligent agents living in a complex environment, both frameworks provide interesting and powerful approaches to help them realizing their intentions and goals in a particular effective and flexible way. Many approaches have been developed in Artificial Intelligence in order to formalize reasoning under uncertainty, as well as reasoning under incomplete information with rules having potential exceptions. Some of them are symbolic and based on a logical framework or on logic programming. Others are more numerically oriented and make use of probabilities, or possibilistic logic.

This is the special session on *Theory of NMR and Uncertainty*, held in Lake District, England, on June $1st$, 2006, in the framework of the 11th International Workshop on Nonmonotonic Reasoning (NMR'2006).

It gathers 14 contributions that covers various facets of recent researches which are at the junction between nonmonotonic reasoning and the symbolic and numerical handling of uncertainty.

The five first papers deal with fusion and revision of (possibily inconsistent) beliefs and preferences. Didier Dubois' paper is on the issue of iterated belief revision, discussing *Three views on the revision of epistemic states* emerging from three different paradigms. He elaborates relationships to prioritized merging and to conditional belief revision, and reveals clashes between some approaches to iterated belief revision and the famous claim by Gärdenfors and Makinson, that belief revision and nonmonotonic reasoning are two sides of the same coin.

Guilin Qi, Weiru Liu, and David A. Bell deal in *A revision-based approach for handling inconsistency in description logics* with revision operators for description logics. They first investigate their logical properties, and then make use of them to cope with inconsistency in stratified description logic bases.

In *Merging stratified knowledge bases under constraints*, Guilin Qi, Weiru Liu, and David A. Bell propose a family of merging operators for combining stratified knowledge bases under integrity constraints. These knowledge bases need not be self-consistent, nor do they have to share a common scale.

In their paper *Merging Optimistic and Pessimistic Preferences*, Souhila Kaci and Leon van der Torre distinguish between controllable and uncontrollable variables for decision making, where the first ones are considered under an optimistic perspective while the second ones are seen more pessimistic, taking the worst case into account.

Similarity between worlds is a crucial notion for many nonmonotonic consequence relations, and distance measures are a proper means to make this notion more precise. Ofer Arieli pursues this idea in his paper *Distance-Based Semantics for Multiple-Valued Logics* in the context of paraconsistent logics.

The next two papers concern default rules having exceptions, and logic programming with default negations.

On Compatibility and Forward Chaining Normality is on extensions of the class of normal default theories. Mingyi Zhang, Ying Zhang, and Yisong Wang study weakly auto-compatible default theories and their relationships to auto-compatible default theories and Forward Chaining normal default theories. The latter ones generalize normal default theories but share most desirable properties with these.

In *Incomplete knowledge in hybrid probabilistic logic programs*, Emad Saad presents a probabilistic answer set semantics for annotated extended logic programs, allowing both classical and default negation in their syntax.

The two next papers address the formalisation of causality in probabilistic and possibilistic framework. Joost Vennekens, Marc Denecker, and Maurice Bruynooghe present in *Extending the role of causality in probabilistic modeling* a logic that uses conditional probabilistic events as atomic constructs and that is based on two fundamental causal principles. They show interesting relationships between their work and the theories of Bayesian networks and probabilistic logic programming, respectively.

The paper *Model and experimental study of causality ascriptions* by Jean-Francois Bonnefon, Rui Da Silva Neves, Didier Dubois, and Henri Prade discusses an agent's capability of recognizing causal relationships (and related notions of facilitation and justification) from a psychological point of view. Background knowledge in an uncertain world is here represented by means of nonmonotonic consequence relations.

Miodrag Raskovic, Zoran Markovic, and Zoran Ognjanovic prove in *Decidability of a Conditional-probability Logic with Non-standard Valued Probabilities* the decidability of their probabilistic logic that allows the representation of vague or imprecise probabilistic statements. Their framework covers also the case when conditioning is done on events of zero probability, and can be used for default reasoning as well.

Nonmonotonic reasoning basically centers around the question how consequences may change when knowledge is enlarged or shrunken. Technically, this often comes down to inserting or forgetting chunks of information, represented e.g. by literals. *About the computation of forgetting symbols and literals* by Yves Moinard considers this issue from a computational point of view.

The last three papers deal with argumentation and possibilistic reasoning. In *Handling (un)awareness and related issues in possibilistic logic: A preliminary discussion*, Henri Prade sheds some light on the investigation of unawareness in the possibilistic framework. He points out how different graded modalities here can prove to be useful for capturing forms of (un)awareness.

Possibilistic Defeasible Logic Programming is already quite a rich framework for knowledge representation, combining features both from logic programming and argumentation theory, and also allowing possibilistic uncertainty. In *On the Computation of Warranted Arguments within a Possibilistic Logic Framework with Fuzzy Unification*, Teresa Alsinet, Carlos Chesnevar, Lluis Godo, Sandra Sandri, and Guillermo Simari extend this approach once again by incorporating elements of fuzzy logic.

Finally, in their second paper, *Preference reasoning for argumentation: Non-monotonicity and algorithms*, Souhila Kaci and Leon van der Torre apply preference reasoning to argumentation theory, making it possible to compare the acceptability of arguments via ordered values.

Session chairs

Salem Benferhat
(benferhat@cril.univ-artois.fr)

Gabriele Kern-Isberner
(gabriele.kern-isberner@cs.uni-dortmund.de)

Program committee

Gerd Brewka
(brewka@informatik.uni-leipzig.de)

Alexander Bochman
(bochmana@hit.ac.il)

Jim Delgrande
(jim@cs.sfu.ca)

Marc Denecker
(Marc.Denecker@cs.kuleuven.be)

Angelo Gilio
(gilio@dmmm.uniroma1.it)

Lluis Godo
(godo@iiia.csic.es)

Rolf Haenni
(haenni@iam.unibe.ch)

Weiru Liu
(W.Liu@qub.ac.uk)

Thomas Lukasiewicz
(Thomas.Lukasiewicz@dis.uniroma1.it)

David Makinson
(david.makinson@kcl.ac.uk)

Robert Mercer
(mercer@csd.uwo.ca)

Henri Prade
(prade@irit.fr)

Bernd Reusch
(Bernd.Reusch@udo.edu)

Karl Schlechta
(ks1ab@web.de)

Guillermo Simari
(grs@cs.uns.edu.ar)

Paul Snow
(paulsnow@verizon.net)

Choh Man Teng
(cmteng@ai.uwf.edu)

Leon Van der Torre
(leon.vandertorre@uni.lu)

Emil Weydert
(emil.weydert@uni.lu)

Nic Wilson
(n.wilson@4c.ucc.ie)

Schedule Thursday 1 June 2006 (Thirlmere-Wastwater Room)
Session Chairs: S Benferhat and G Kern-Isberner

- 10.30MRaskovic, Z Markovic, and Z Ognjanovic, Decidability of a conditional-probability logic with non-standard valued probabilities
- 10.55 Y Moinard, About the computation of forgetting symbols and literals
- 11.20 H Prade, Handling (un)awareness and related issues in possibilistic logic: A preliminary discussion

- 11.45 T Alsinet, C Chesnevar, L Godo, S Sandri, and G Simari, On the computation of warranted arguments within a possibilistic logic framework with fuzzy unificiation
- 12.10 S Kaci and L van der Torre, Preference reasoning for argumentation: Non-monotonicity and algorithms
- 12.35 Lunch
- 13.50 O Arieli, Distance-based semantics for multiple-valued logics
- 14.15 E Saad, Incomplete knowledge in hybrid probabilistic logic programs
- 14.40 J Vennekens, M Denecker, and M Bruynooghe, Extending the role of causality in probabilistic modeling
- 15.05 J Bonnefon, R Da Silva Neves, D Dubois, and H Prade, Causality ascriptions: Model and experimental study of
- 15.30 Coffee
- 16.00 D Dubois, Three views on the revision of epistemic states
- 16.25 G Qi, W Liu, and D Bell, A revision-based approach for handling inconsistency in description logics
- 16.50 G Qi, W Liu, and D Bell, Merging stratified knowledge bases under constraints
- 17.15 S Kaci and L van der Torre, Merging optimistic and pessimistic preferences
- 17.40 M Zhang, Y Zhang andWYisong, On compatibility and forward chaining normality

2.1 Three views on the revision of epistemic states

Three scenarios for the revision of epistemic states *

Didier Dubois
IRIT-CNRS
Université Paul Sabatier
Toulouse, France
dubois@irit.fr

Abstract

This position paper discusses the difficulty of interpreting iterated belief revision in the scope of the existing literature. Axioms of iterated belief revision are often presented as extensions of the AGM axioms, upon receiving a sequence of inputs. More recent inputs are assumed to have priority over less recent ones. We argue that this view of iterated revision is at odds with the claim, made by Gärdenfors and Makinson, that belief revision and non-monotonic reasoning are two sides of the same coin. We lay bare three different paradigms of revision based on specific interpretations of the epistemic entrenchment defining an epistemic state and of the input information. If the epistemic entrenchment stems from default rules, then AGM revision is a matter of changing plausible conclusions when receiving specific information on the problem at hand. In such a paradigm, iterated belief revision makes no sense. If the epistemic entrenchment encodes prior uncertain evidence and the input information is at the same level as the prior information and possibly uncertain, then iterated revision reduces to prioritized merging. A third problem is one of the revision of an epistemic entrenchment by means of another one. In this case, iteration makes sense, and it corresponds to the revision of a conditional knowledge base describing background information by the addition of new default rules.

Introduction

The interest in belief revision as a topic of investigation in artificial intelligence was triggered by Gärdenfors (1988) book and the axiomatic approach introduced by C. Alchourrón, P. Gärdenfors and D. Makinson (1985) in the setting of propositional logic. This approach assumes that the set of accepted beliefs held by an agent is a deductively closed set of propositions. On this basis, axioms of belief change (revision, but also contraction) formulate constraints that govern the "flux" of information, i.e. that relate one belief set to the next one upon receiving a new piece of information. An important assumption is that belief revision takes place in a static world, so that the input information is supposed to bring insight to a case that the agent deals with, but is never meant to indicate that the world considered by the agent receiving it has evolved.

The crucial point of the AGM theory is that the axiomatic framework enforces the existence of a so-called epistemic entrenchment relation between propositions of the language. This relation acts like a priority assignment instrumental to determine the resulting belief set after revision. It is also similar (even if purely ordinal) to a probability measure. More specifically, an epistemic entrenchment is a complete preordering between propositions which looks like a comparative probability relation (Fishburn 1986), even if it has different properties. Properties of an epistemic entrenchments make it expressible in terms of a complete plausibility ordering of possible worlds, such that the resulting belief set after receiving input A is viewed as the set of propositions that are true in the most plausible worlds where A holds.

The AGM theory leaves the issue of iterated revision as an open problem. Since then, iterated revision has been the topic of quite a number of works (Nayak 1994), (Williams 1995), (Darwiche & Pearl 1997), (Lehmann 1995), (Jin & Thielscher 2005). However it also seems to have created quite a number of misunderstandings, due to the lack of insight into the nature of the problem to be solved.

A typical question that results from studying the AGM theory is: What becomes of the epistemic entrenchment after the belief set has been revised by some input information? Some researchers claimed it was simply lost, and that the AGM theory precludes the possibility of any iteration. Others claimed that it changes along with K, and tried to state axioms governing the change of the plausibility ordering of the worlds, viewing them as an extension of the AGM axioms. This trend led to envisage iterated belief revision as a form of prioritized merging where the priority assignment to pieces of input information reflected their recency.

However, this notion of iterated belief revision seems to be at odds with Gärdenfors and Makinson (1994) view of belief revision as the other side of non-monotonic reasoning, where the epistemic entrenchment relation is present from the start and describes the agent's expectations in the face of the available evidence. Such an epistemic entrenchment may also derive from the analysis of a set of conditionals, in

*This position paper was triggered by discussions with Jerome Lang and Jim Delgrande at a Belief Revision seminar in Dagstuhl, in August 2005

the style of (Lehmann & Magidor 1992), yielding a ranking of worlds via the so-called rational closure.

The revised belief set is then the result of a simple inference step of conditionals from conditionals, whereby propositional conclusions tentatively drawn are altered by the arrival of new pieces of evidence. In this framework, the conditional information, hence the plausibility ordering, is never revised and iteration comes down to inference of new conclusions and dismissal of former ones, in the spirit of nonmonotonic reasoning.

Solving the clash of intuitions between iterated revision and non-monotonic reasoning leads us to considering that the AGM view of belief revision (related to non-monotonic reasoning) has more to do with inference under incomplete information than with iterated revision as studied by many subsequent researchers (see a critical discussion of Darwiche and Pearl(1997) axioms along this line in (Dubois, Moral, & Prade 1998)). Two settings for revision, namely revision as defeasible inference, and revision as prioritized merging emerge, that deal with distinct problems.

This note is also in the spirit of a former position paper by Friedman and Halpern (1996a). In that note, they complain that iterated belief revision research relies too much on the finding of new axioms justified by toy-examples, and representation results, while more stress should be put on laying bare an appropriate "ontology", that is, describing a concrete problem or scenario that iterated revision is supposed to address. Friedman and Halpern suggest two such ontologies, that basically differ by the meaning of the input information. According to the first one, the agent possesses knowledge and beliefs about the state of the world, knowledge being more entrenched than beliefs, and receives inputs considered as true observations. This view is similar to a form of conditioning in the sense of uncertainty theories. In the other scenario, the input information is no longer systematically held for true and competes with prior beliefs, thus corresponding to a kind of merging bearing much similarity to the combination of uncertainty in the theory of evidence (Shafer 1976).

In this paper, we somewhat pursue this discussion by pointing out that the status of the epistemic entrenchment itself may also be understood differently: in some scenarios, it represents background information about the world, telling what is normal from what it is not, in a refined way. In that case, the plausibility ordering underlying the epistemic entrenchment is similar to a statistical probability distribution, except that the underlying population is ill-specified, and statistical data is not directly accessible. In other scenarios, the plausibility ordering expresses beliefs about unreliable observations about the solution to a problem at hand, the pieces of evidence gathered so far from witnesses on a whodunit case, for instance. In the latter situation, the resulting epistemic entrenchment is fully dependent on the case at hand and has no generic value.

It leads to propose three change problems that have little to do with each other even if they may share some technical tools. If we take it for granted that belief revision and nonmonotonic reasoning are two sides of the same coin and if we rely on technical equivalence results between Lehmann and Magidor(1992) conditional logic under rational closure, and the AGM theory, then we come up with a qualitative counterpart of statistical reasoning, with inputs taken as incomplete but sure information about a case at hand. We call it Belief Revision as Defeasible Inference (BRDI). On the other hand, if we take it for granted that the epistemic entrenchment gathers uncertain evidence about a case, likely to evolve when new uncertain pieces of evidence are collected, we speak of Belief Revision as Prioritized Merging (BRPM). Finally, we consider the situation where our background knowledge is modified by new pieces of knowledge, whereby states of fact that we used to think as normal turn out not to be so, or conversely. We then speak of Revision of Background Knowledge by Generic Information (RBKGI). In the latter case, inputs take the form of conditionals.

It may be that other scenarios for belief change could be pointed out. However, we claim that iterated revision in each of the above scenarios corresponds to very different problems. A companion paper (Delgrande, Dubois, & Lang 2006) proposes a formal framework for the BRPM situation in full details. Here, we propose an informal comparative discussion of the three scenarios.

Belief Revision as Defeasible Inference (BRDI)

In the first setting, the AGM theory and non-monotonic reasoning are really regarded as two sides of the same coin. However, while in the AGM approach, only a flat belief set denoted K, composed of logical formulas, is explicitly available (since the epistemic entrenchment is implicit in the axioms of the theory), the nonmonotonic logic approach lays bare all the pieces of information that allows an agent to reason from incomplete reliable evidence and background knowledge. While in the AGM paradigm, the primitive object is the belief set, in the following, everything derives from conditional information, synthetized in the form of a partial ordering of propositions, and the available evidence. This view is fully developed by Dubois Fargier and Prade (2004) (2005) as a theory of accepted beliefs.

In the following, we consider a classical propositional language, and we do not distinguish between logically equivalent propositions. Hence, we consider propositions as subsets of possible worlds, in other words, events (to borrow from the probabilistic literature). The influence of syntax on revision is out of the scope of this paper. Under such a proviso, it is assumed that the agent's epistemic state is made of three components:

1. A confidence relation, in the form of a partial ordering $\succ$ on propositions $A, B, \ldots$ expressed in a given language. This relation, which should be in agreement with logical deduction, expresses that some propositions are more normally expected (or less suprizing) than others. It encodes the background information of the agent, which de-

scribes how (s)he believes the world behaves in general. It reflects the past experience of the agent. Such a confidence relation may directly stem from a set of conditionals Δ. Δ contains pieces of conditional knowledge of the form $A \rightarrow B$ where $\rightarrow$ is a nonclassical implication, stating that in the context where all that is known is A, B is generally true. Each such conditional is then encoded as the constraint $A \wedge B \succ A \wedge \neg B$, understood as the statement that $A \wedge B$ is generally more plausible (that is, less surprizing) than $A \wedge \neg B$ (Friedman & Halpern 1996b). A plausibility ordering of worlds $\geq_\pi$ can be derived from such constraints via some information minimization principle (like rational closure of Lehmann and Magidor (1992), or equivalently, the most compact ranking compatible with the constraints(Pearl 1990), or yet the principle of minimal specificity of possibilistic logic (see (Benferhat, Dubois, & Prade 1997) for instance).

2. A set of contingent observations concerning a case of interest for the agent, under the form of a propositional formula A. The observations are sure evidence about this case, not general considerations about similar cases. Such pieces of evidence are sure facts (or at least accepted as such), hence consistent with each other. It means that a preliminary process is capable of handling conflicting observations and come up with a consistent report.
3. The belief set $K * A$ of the agent. It is made of propositions tentatively accepted as true by the agent about the case, in the face of the current observations. Propositions in $K * A$ are inferred from the observations and the background knowledge (so it is not an independent part of the epistemic state). K is the belief set of the agent before hearing about A. That input information is safe explains why the success postulate ($A \in K * A$) makes sense.

For instance consider a medical doctor about to diagnose a patient. It is assumed that the aim is to determine what the patient suffers from within a time-period where the disease does not evolve. The plausibility ordering reflects the medical knowledge of the medical doctor in general. Before seeing the patient, (s)he may have some idea of which diseases are more plausible than others. Observations consist of reports from medical tests and information provided by the patient on his state of health. The resulting belief set contains the diagnosis of the patient that will be formulated by the doctor on the basis of the available observations. This belief set concerns the patient, not people's health in general.

Formally, under this view, the original belief set K is inferred from Δ, or from $\succ$, or from $\geq_\pi$, (according to the choice of a representation) and from the tautology as input ($A = \top$, assuming no observations). $K * A$ is derived likewise from input A. In terms of conditionals, the change from K to $K * A$ stems from the fact that the conditionals $\top \rightarrow K$ and $A \rightarrow K * A$, respectively, can be inferred from Δ under some inferential system. In terms of a confidence relation $\succ$ between propositions $K * A = \{B, A \wedge B \succ A \wedge \neg B\}$. Dubois et al. (2005) show that requiring the deductive closure of $K * A$ is enough to recover system P of Kraus et al.(1990). Moreover if $\succ$ is the strict part of a complete preordering, one recovers the setting of possibility theory (Dubois, Fargier, & Prade 2004) and all the AGM axioms of belief revision (restricted to consistent inputs). In other words, $\succ$ is a comparative possibility relation in the sense of Lewis(1973), that derives from a plausibility ordering $\geq_\pi$ of possible worlds. Under a plausibility ordering $\geq_\pi$, it is well-known after Grove(1988) that K (resp. $K * A$) are the set of propositions true in the most plausible worlds (resp. where A is true).

This approach is very similar to probabilistic reasoning as emphasized by Pearl (1988), Dubois and Prade (1994). A set of conditionals Δ is the qualitative counterpart of a set of conditional probabilities of the form $P(B \mid A) = \alpha$ defining a family of probability measures. There is no need to resort to infinitesimals for bridging the gap between nonmonotonic reasoning and probabilistic reasoning. Recent works by Gilio and colleagues (2002) indicate that probabilistic reasoning with conditionals of the form $P(B \mid A) = 1$, precisely behaves like system P of Kraus et al. Benferhat *et al.* (1999), show that if we restrict to so-called big-stepped probabilities, conditionals can be interpreted by constraints $P(A \wedge B) > P(A \wedge \neg B)$.

Along the same lines, extracting a minimally informative plausibility ordering of worlds $\geq_\pi$ from a set of conditionals is very similar to the application of the maximal entropy principle from a set of conditional probabilities, an approach advocated by Paris (1994). This similarity has been studied by Maung(1995). So reasoning according to a plausibility ordering is also similar to probabilistic reasoning with Bayes nets (Pearl 1988). In this approach, the background knowledge is encoded by means of a (large) joint probability distribution on the state space defined by a set of (often Boolean) attributes. This probability distribution embodies statistical data pertaining to a population (e.g. of previously diagnosed patients, for instance) in the form of a directed acyclic graph and conditional probability tables. The advantage of the Bayes net format is to lay bare conditional independence assumptions and simplify the computation of inference steps accordingly. The network is triggered by the acquisition of observations on a case. Inferring a conclusion C based on observing A requires the computation of a conditional probability $P(C \mid A)$, and interpreting it as the degree of belief that C is true for the current situation for which all that is known is A. Apart from computing degrees of belief, one is interested in determining the most probable states upon learning A.

It is clear that the plausibility ordering in the above view of the AGM framework plays the same role as a Bayes net. Especially, $\geq_\pi$ might compile a population of cases, even if this population is ill-defined in the non-monotonic setting (the agent knows that "Birds fly" but it is not entirely clear which population of birds is referred to). It means that the input observations, since pertaining only to the case at hand, are not of the same nature as the plausibility ordering, and are not supposed to alter it, just like a Bayes net is not changed by querying it. In this framework, iterating belief change just means accumulating consistent observations and

reasoning from them using the background knowledge. Interestingly, plausibility orderings, encoded as possibility distributions can be represented using the same graphical structures as joint probability distributions (see (Benferhat *et al.* 2002a)), and local methods for reasoning in such graphs can be devised (BenAmor, Benferhat, & Mellouli 2003). These graphical representations are equivalent to the use of possibilistic logic, but not necessarily more computationally efficient. In the purely ordinal case, CP-nets are also the counterparts of Bayes nets, and it is strange they are only proposed for preference modeling, while they could also implement a form of plausible reasoning compatible with the above "ontology" of qualitative reasoning under incomplete observations using background knowledge.

Belief Revision as Prioritized Merging

A radically different view is to consider that an epistemic state is made of uncertain evidence about a particular world of interest (a static world, again). It gathers the past uncertain observations obtained so far about a single case. So the belief set K is actually a completely ordered set (ordered by the epistemic entrenchment), and the underlying plausibility ordering on worlds describes what is the most plausible solution to the problem at hand. The epistemic entrenchment describes what should be more or less believed about the current case. In the BRPM view, the plausibility ordering is no longer like a statistical distribution.

The new observations A have the same status as the plausibility ordering, and are likely to modify it. They are testimonies or sensor measurements. They could be unreliable, uncertain.

So this kind of belief change is particularly adapted to the robotics environment for the fusion of unreliable measurements. It also accounts for the problem of collecting evidence, where the main issue is to validate facts relevant to a case on the basis of unreliable testimonies and incomplete observations. As an example, consider a criminal case where the guilty person is to be found on the basis of (more or less unreliable) testimonies and clues. The investigator's beliefs reflect all evidence gathered so far about the case. The input information consists of an additional clue or testimony.

Under this view, belief revision means changing the pair $(K, \geq_\pi)$ into another pair $(K * A, \geq_{\pi_A})$. Again the belief set K is induced by the plausibility ordering, but here there is no background knowledge at work. A new input should be merged with the existing information, with its own reliability level. If this level is too weak, it may be contradicted by the original belief set. Note that K cannot be viewed as knowledge (as opposed to belief). It is just what the agent thinks is more likely. Here, iterating the revision process makes sense, and comes down to a merging process because the a priori information and the input information are of the same nature. The success postulate just expresses the fact that the newest information is the most reliable. Not questioning this postulate has led to a view of iterated belief revision where the newest piece of information is always more reliable than the previous ones.

One may argue that iterated belief revision can be more convincingly considered as a form of prioritized merging. Indeed, it seems that assigning priorities on the sole basis of the recency of observations in a static problem about which information accumulates is not always a reasonable assumption. Sherlock Holmes would not dismiss previously established facts on the basis of new evidence just because such evidence is new.

At the computational level, an epistemic state $(K, \geq_\pi)$ is best encoded as an ordered belief base using possibilistic logic (Dubois, Lang, & Prade 1994) or kappa rankings (Williams 1995). However the meaning of a prioritized belief base differs according to whether it is viewed as a partial epistemic entrenchment (what Williams calls an "ensconcement") or as a set of constraints on a family of possible epistemic entrenchments (possibilistic logic). Practical methods for merging ordered belief bases were devised in (Benferhat *et al.* 1999), (Benferhat *et al.* 2000) and in the special case when the success postulate is acknowledged see (Benferhat *et al.* 2002c).

The numerical counterpart to this view of iterated revision here is to be found in Shafer(1976)'s mathematical theory of evidence. In this theory, an unreliable testimony takes the form of a proposition E and a weight $m(E)$ reflecting the probability that the source providing E is reliable. It means that with probability $1 - m(E)$, the input information is equivalent to receiving no information at all. More generally, a body of evidence is made of a set of propositions E_i along with positive masses $m(E_i)$ summing to 1. $m(E_i)$ is the probability that proposition E_i correctly reflects the agent's evidence about the case at hand. The degree of belief $Bel(C)$ of a proposition C is the probability that C can be logically inferred from the agent's body of evidence (summing the masses of propositions E_i that imply C). Revising the agent belief upon arrival of a sure piece of information A ($m'(A) = 1$) comes down to a conditioning process ruling out all states or worlds that falsify A. If the input information is not fully reliable, Dempster's rule of combination, an associative and commutative operation, carries out the merging process. Note that the symmetry of the operation is due to the fact that the new pair $(A, m'(A))$ is merged with the body of evidence. The smaller $m'(A)$, the less effective is the input information A in the revision process.

When the input information is legitimately considered as more reliable than what has been acquired so far, merging the plausibility ordering and the new observation in a non-commutative way is a possible option. A similar view was advocated by (Dubois & Prade 1992) where the plausibility ordering was encoded by means of a possibility distribution. The AGM axioms were extended to plausibility orderings $\geq_\pi$ and are thus discussed in terms of their relevance for characterizing the revision of possibility distributions by input information. The success postulate led us to consider belief revision as a form of conditioning, in the tradition of

probability kinematics (Domotor 1980).

Darwiche and Pearl (1997) axioms of iterated belief change embody the principle of minimal change of the ordering that is expected when the priority is always given to the new information. Among revision operations satisfying these postulates (applied to plausibility orderings) Boutilier's natural revision (Boutilier 1993) can be viewed as iterated revision of a plausibility ordering $\geq_\pi$, with priority to the new input A. In this scheme, the resulting most plausible worlds are the $\geq_\pi$-best A-worlds, all other things remaining equal, while possibilistic conditioning flatly eliminates worlds not in agreement with the input information (thus not obeying the Darwiche-Pearl postulates). Papini and colleagues (Benferhat *et al.* 2002b) adopt the view that in the resulting plausibility ordering all A-worlds are more plausible than any $\neg A$-world all things being equal. This method also satisfies the Darwiche-Pearl postulates.

The case of uncertain inputs is discussed in (Dubois & Prade 1992). It is pointed out that two situations may occur: one whereby the degree of certainty of the new piece of information is considered as a constraint. Then, this piece of information is to be entered into the a priori ordered belief set with precisely this degree of certainty. If this degree of certainty is low it may result in a form of contraction (if the source reliably claims that a piece of information cannot be known, for instance). In probability theory this is at work when using Jeffrey's revision rule (Jeffrey 1965). Darwiche and Pearl (1997) propose one such revision operation in terms of kappa-functions. The other view is that the degree of uncertainty attached to the input is an estimation of the reliability of the source, and then the piece of information is absorbed or not into the belief set. The latter view is more in line with the prioritized merging setting.

The companion paper (Delgrande, Dubois, & Lang 2006) reconsiders postulates for iterated revision without making any recency assumption: there is a certain number of more less reliable pieces of information to be merged, one of them being the new one. If we postulate that all uncertain observations play the same role and have the same reliability, a symmetric (and possibly associative) merging process can take place.

Reliability degrees are no longer a matter of recency, but can be decided on other grounds. In (Delgrande, Dubois, & Lang 2006), four axioms, for the prioritized merging of unreliable propositions into a supposedly accepted one are proposed. They embody the BRPM scenario of evidence collection and sorting producing a clearly established fact (a propositional formula representing a belief set). Informally they express the following requirements:

- A piece of information at a given reliability level should never make us disbelieve something we accepted after merging pieces of information at strictly higher reliability levels.
- The result of merging should be consistent.
- Vacuous evidence does not affect merging.
- Optimism: The result of merging consistent propositions is the conjunction thereof.

The important postulate is optimism, which suggests that if supposedly reliable pieces of information do not conflict, we can take them for granted. In case of conflicts, one may then assume as many reliable pieces of information as possible so as to maintain local consistency. It leads to optimistic assumptions on the number of truthful sources, and justify procedures for extracting maximal consistent subsets of items of information, see (Dubois & Prade 2001). This may be viewed as an extended view of the minimal change postulate, via the concern of keeping as many information items as possible. A restricted form of associativity stating that merging can be performed incrementally, from the most reliable to the least reliable pieces of information is proposed as optional. These axioms for prioritized merging recover Darwiche and Pearl postulates (except the controversial C2 dealing with two successive contradictory inputs) as well as two other more recent postulates from (Nayak *et al.* 1996; Nayak, Pagnucco, & Peppas 2003), and from (Jin & Thielscher 2005), when the reliability ordering corresponds to recency. It also recovers the setting of Konieczny and Pino-Perez (2002) for flat merging under integrity constraints for the fusion of equally reliable items in the face of more reliable ones. The prioritized merging setting of (Delgrande, Dubois, & Lang 2006) can also be viewed as a framework for extracting a set of preferred models from a potentially inconsistent prioritized belief base. Extending the postulates to outputs in the form of an ordered belief set is a matter of further research.

Interestingly, the BRPM scenario can be articulated with the previous BRDI scenario. One may see the former as a prerequisite for the latter: first evidence must be sorted out using a BRPM step, and then once a fact has been sufficiently validated, the agent can revise plausible conclusions about the world, based on this fact using BRDI (in order to suggest the plausible guilty person in a case, thus guiding further evidence collection).

AGM = BRDI or BRPM ?

Considering the relative state of confusion in the iterated revision literature, it is not completely clear what the AGM theory is talking about: BRDI or BRMP. Due to the stress given subsequently by Gärdenfors and Makinson (1994) to the similarity between non-monotonic reasoning and belief revision, it is natural to consider that BRDI is the natural framework for understanding their results. But then it follows that iterated revision deals with a different problem, and the above discussion suggests it can be BRMP.

1. In the AGM theory you never need K to derive $K * A$, you only need the revision operation * (in other words the plausibility ordering) and A. So the notation $K * A$ is in some sense misleading, since it suggests an operation combining K and A. This point was also made by Friedman and Halpern (1996a) In the BRPM view, the result-

ing epistemic state is also a function of the prior epistemic state and the input information only.

2. The AGM postulates of belief revision are in some sense written from a purely external point of view, as if an observer had access to the agent's belief set from outside, would notice its evolution under input information viewed as stimuli, and describe its evolution laws (the AGM theory says: if from the outside, an agent's beliefs seem to evolve according to the postulates, then it is as if there were a plausibility ordering that drives the belief flux). In this view, the background knowledge remains hidden to the observer, and its existence is only revealed through the postulates (like small particles are revealed by theories of microphysics, even if not observed yet). In the BRPM problem, the prior plausibility ordering is explicitly stated. Under the BRDI view, for practical purposes, it also looks more natural to use the plausibility ordering as an explicit primitive ingredient (as done by (Gärdenfors & Makinson 1994) and to take an insider point of view on the agent's knowledge, rather than observing beliefs change from the outside.
3. The belief revision step in the AGM theory leaves the ordering of states unchanged under the BRDI view. This is because inputs and the plausible ordering deal with different matters, resp. the particular world of interest, and the class of worlds the plausible ordering refers to. The AGM approach, in the BRDI view is a matter of "querying" the epistemic entrenchment relation, basically, by focusing it on the available observation. Under this point of view, axioms for revising the plausibility ordering, as proposed by (Darwiche & Pearl 1997), for instance, cannot be seen as additional axioms completing the AGM axioms. On the contrary, the prioritized merging view understands the AGM axioms as relevant for the revision of epistemic states and apply them to the plausibility ordering. As such they prove to be insufficient for its characterization, hence the necessity for additional axioms.
4. In BRDI, while belief sets seem to evolve (from K to $K*A$ to $(K*A)*B\ldots$) as if iterated belief revision would take place, $(K*A)*B$ is really obtained by gathering the available observations A and B and inferring plausible beliefs from them. Again we do not compute $(K*A)*B$ from $K*A$. But $(K*A)*B$ means $K*(A\wedge B)$ (itself not obtained from K), with the proviso that A and B should be consistent. And indeed, within the BRDI view,

$$(K*A)*B = K*(A\wedge B) \text{ if } A\wedge B\neq\bot$$

is a consequence of AGM revisions (especially Axioms 7 and 8), if we consider that after revision by A the plausibility ordering does not change (we just restrict it to the A-worlds). Strictly speaking, these axioms say that the identity holds if B is consistent with $K*A$ (not with A). However, if the relative plausibility of worlds is not altered after observing A, the subsequent revision step by observation B will further restrict $\geq_\pi$ to the $A\wedge B$-worlds since $A\wedge B\neq\bot$, and the corresponding belief set is thus exactly $K*(A\wedge B)$ corresponding the most plausible among $A\wedge B$-worlds. It underlies an optimistic assumption about input information, namely that both A and B are reliable if consistent (a postulate of prioritized merging). This situation is similar to probabilistic conditioning whereby iterated conditioning ($P(C\mid A\mid B)$) comes down to simple conditioning on the conjunction of antecedents ($P(C\mid A\wedge B)$). Of course this is also a restricted view of the AGM theory, forbidding not only the revision by $\bot$, but also by a sequence of consistent inputs that are globally inconsistent. But we claim that this restriction is sensible in the BRDI scenario.
5. If in the AGM setting, observations A, B are inconsistent then the BRDI scenario collapses, because it means that some of the input facts are wrong. In this case, even if the AGM theory proposes something, the prospect it offers is not so convincing, as this is clearly a pathological situation. Similarly, in probabilistic reasoning, conditioning on a sequence of contradicting pieces of evidence makes no sense. Within the BRDI view, the natural approach is to do a merging of observations so as to restore a consistent context prior to inferring plausible beliefs (and as suggested above, the BRPM could be applied to the merging of such inconsistent input observations). In the medical example, it is clear that the physician receiving contradictory reports about the patient will first try to sort out the correct information prior to formulating a diagnosis. In the BRPM view, there is nothing anomalous with the situation of several conflicting inputs, because this conflict is expected as being of the same nature as the possible conflict between the agent's epistemic state and one piece of input information.

In summary, under the BRDI view, the belief revision problem (moving from K to $K*A$) is totally different from the problem of revising the plausible ordering of states of nature, while in the BRPM view both are essentially the same problem and must be carried out conjointly. In particular, it makes no sense to "revise an ordering by a formula", in the AGM framework. In the BRPM view, the input proposition A is viewed as an ordering of worlds such that at least one world where A is true is more likely than any world where A is false. In other words, belief revision can be cast within a more general setting of merging uncertain pieces of evidence (encoded by plausibility orderings).

Revision of Background Knowledge by Generic Information (RBKGI)

In the BRDI view, apart from the (contingent) belief revision problem addressed by the non-pathological part of the AGM theory and non-monotonic inference, there remains the problem of revising the generic knowledge itself (encoded or not as a plausibility ordering) by means of input information of the same kind. The AGM theory tells nothing about it. This problem is also the one of revising a set of conditionals by a new conditional (Boutilier & Goldszmidt 1993). Comparing again to probabilistic reasoning, contingent belief revision is like computing a conditional probability using observed facts instantiating some variables, while

revising a plausibility ordering is like revising a Bayes net (changing the probability tables and/or the topology of the graph). In the medical example, the background knowledge of the physician is altered when reading a book on medicine or attending a specialized conference on latest developments of medical practice.

One interesting issue is the following: since background knowledge can be either encoded as a plausibility ordering $\geq_\pi$ or as a conditional knowledge base Δ, should we pose the RBKGI problem in terms of revising Δ or revising $\geq_\pi$?

Suppose Δ is a conditional knowledge base, which, using rational closure, delivers a plausibility ordering $\geq_\pi$ of possible worlds. Let $A \rightarrow B$ be an additional generic rule that is learned by the agent. If $\Delta \cup \{A \rightarrow B\}$ is consistent (in the sense that a plausibility ordering $\geq_{\pi'}$ can be derived from it), it is natural to consider that the revision of $\geq_\pi$ yields the plausibility ordering $\geq_{\pi'}$, obtained from $\Delta \cup \{A \rightarrow B\}$ via rational closure. Viewed from the conditional knowledge base this form of revision is just an expansion process. The full-fledged revision would take place when the conditional $A \rightarrow B$ contradicts Δ, so that no plausibility ordering is compatible with $\Delta \cup \{A \rightarrow B\}$ (Freund 2004). This kind of knowledge change needs specific rationality postulates for the revision of conditional knowledge bases, in a logic that is not classical logic, but the logic of conditional assertions of Kraus et al.(1990).

Alternatively, one may attempt to revise the plausibility ordering $\geq_\pi$ (obtained from Δ via a default information minimisation principle), using a constraint of the form $A \wedge B \succ A \wedge \neg B$. To do so, Darwiche-Pearl postulates can be a starting point, but they need to be extended in the context of this particular type of change. Results of (Freund 2004) and (Kern-Isberner 2001) seem to be particularly relevant in this context. For instance it is not clear that the change process should be symmetric. One might adopt a principle of minimal change of the prior beliefs under the constraint of accepting the new conditional or ordering as a constraint (Domotor 1980). A set of postulates for revising a plausibility ordering (encoded by a kappa-function) by a conditional input information of the form $A \wedge B \succ A \wedge \neg B$ is proposed by Kern-Isberner (2001). They extend the Darwiche-Pearl postulates and preserve the minimal change requirement in the sense that they preserve the plausibility ordering $\geq_\pi$ among the examples $A \wedge B$ of the input conditionals, its counterexamples $A \wedge \neg B$, and its irrelevant cases $\neg A$.

Some insights can also be obtained from the probabilistic literature (van Fraassen 1980) (Domotor 1985). For instance Jeffrey's rule consists in revising a probability distribution P, enforcing a piece of knowledge, of the form $P(A) = \alpha$, as a constraint which the resulting probability measure P^* must satisfy. The probability measure "closest" to P in the sense of relative entropy, and obeying $P^*(A) = \alpha$ is of the form $P^*(.) = \alpha.P(. \mid A) + (1 - \alpha)P(. \mid \neg A)$. The problem of revising a probability distribution by means of a conditional input of the form $P(A|B) = \alpha$ has been considered in the probabilistic literature by (van Fraassen 1981). Rules for revising a plausibility ordering can be found in (Williams 1995), (Weydert 2000), (Kern-Isberner 2001) (using the kappa functions of (Spohn 1988)) and (Dubois & Prade 1997) using possibility distributions.

However it is not clear that revising the plausibility ordering $\geq_\pi$ obtained from Δ by a constraint of the form $A \wedge B \succ A \wedge \neg B$ has any chance to always produce the same result as deriving the plausibility ordering $\geq'_\pi$ from the revised conditional knowledge base Δ after enforcing a new rule $A \rightarrow B$.

While our aim is not to solve this question, at least our paper claims that revising generic knowledge whether in the form of a conditional knowledge base, or in the form of a plausibility ordering, is a problem distinct from the one of contingent belief revision (BRDI, which is only a problem of inferring plausible conclusions), and from the prioritized merging of uncertain information. The RBKGI problem can be subject to iterated revision, as well. One may argue that RBKGI underlies an evolving world in the sense of accounting for a global evolution of the context in which we live. In some respects, the normal course of things to-day is not the same as it used to be fifty years ago, and we must adapt our generic knowledge accordingly. The distinction between updates and revision is not so clear when generic knowledge is the subject of change.

Conclusion

This position paper tried to lay bare three problems of belief change corresponding to different scenarios. Results in the literature of iterated belief change should be scrutinized further in the context of these problems. It is clear that addressing these problems separately is a simplification. For instance in the BRDI approach, observations are always considered as sure facts, but one may consider the more complex situation of inferring plausible conclusions from uncertain contingent information using background knowledge. Also the assumption that in the BRDI approach, contingent inputs never alter the background knowledge is also an idealization: some pieces of information may destroy part of the agent's generic knowledge, if sufficiently unexpected (think of the destruction of the Twin Towers); moreover, an intelligent agent is capable of inducing generic knowledge from a sufficient amount of contingent observations. The latter is a matter of learning, and the question of the relationship between learning and belief revision is a natural one even if beyond the scope of this paper.

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change : partial meet contraction and revision functions. *J. Symbolic Logic* 50:510–530.

BenAmor, N.; Benferhat, S.; and Mellouli, K. 2003. Anytime propagation algorithm for min-based possibilistic graphs. *Soft Computing* 8:150–161.

Benferhat, S.; Dubois, D.; Prade, H.; and Williams, M. 1999. A practical approach to fusing prioritized knowledge bases. In *Proc. 9th Portuguese Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence, 222–236. Springer.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2000. Encoding information fusion in possibilistic logic:a general framework for rational syntactic merging. In *Proc.14th Europ.Conf.on Artificial Intelligence (ECAI2000)*, 3–7. IOS Press.

Benferhat, S.; Dubois, D.; Garcia, L.; and Prade., H. 2002a. On the transformation between possibilistic logic bases and possibilistic causal networks. *Int. J. Approximate Reasoning* 29:135–173.

Benferhat, S.; Dubois, D.; Lagrue, S.; and Papini, O. 2002b. Making revision reversible: an approach based on polynomials. *Fundamenta Informaticae* 53:251–280.

Benferhat, S.; Dubois, D.; Prade, H.; and Williams, M. 2002c. A practical approach to revising prioritized knowledge bases. *Studia Logica* 70:105–130.

Benferhat, S.; Dubois, D.; and Prade, H. 1997. Nonmonotonic reasoning, conditional objects and possibility theory. *Artificial Intelligence* 92:259– 276.

Benferhat, S.; Dubois, D.; and Prade, H. 1999. Possibilistic and standard probabilistic semantics of conditional knowledge. *J. Logic and Computation* 9:873–895.

Biazzo, V.; Gilio, A.; Lukasiewicz, T.; and Sanfilippo, G. 2002. Probabilistic Logic under Coherence, Model-Theoretic Probabilistic Logic, and Default Reasoning in System P. *J. Applied Non-Classical Logics* 12(2):189–213.

Boutilier, C., and Goldszmidt, M. 1993. Revision by conditionals beliefs. In *Proc. of the 11th National Conf. on Artificial Intelligence (AAAI'93)*.

Boutilier, C. 1993. Revision sequences and nested conditionals. In *Proceedings of IJCAI'93*.

Darwiche, A., and Pearl, J. 1997. On the logic of iterated belief revision. *Artificial Intelligence* 89:1–29.

Delgrande, J.; Dubois, D.; and Lang, J. 2006. Iterated belief revision as prioritized merging. In *Proceedings of KR'06, Windermere, U.K.*

Domotor, Z. 1980. Probability kinematics and representation of belief change. *Philosophy of Science* 47:284–403.

Domotor, Z. 1985. Probability kinematics - conditional and entropy principles. *Synthese* 63:74–115.

Dubois, D., and Prade, H. 1992. Belief change and possibility theory. In Gärdenfors, P., ed., *Belief Revision*. Cambridge University Press. 142–182.

Dubois, D., and Prade, H. 1994. Non-standard theories of uncertainty in knowledge representation and reasoning. *The Knowledge Engineering Review* 9:399–416.

Dubois, D., and Prade, H. 1997. A synthetic view of belief revision with uncertain inputs in the framework of possibility theory. *Int. J. Approximate Reasoning* 17:295–324.

Dubois, D., and Prade, H. 2001. Possibility theory in information fusion. In *Data Fusion and Perception*, volume 431 of *CISM Courses and Lectures*. Springer. 53–76.

Dubois, D.; Fargier, H.; and Prade, H. 2004. Ordinal and probabilistic representations of acceptance. *J. Artificial Intelligence Research* 22:23–56.

Dubois, D.; Fargier, H.; and Prade, H. 2005. Acceptance, conditionals, and belief revision. In *Conditionals, Information, and Inference*, volume 3301 of *Lecture Notes in Artificial Intelligence*. Springer. 38–58.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic. In Gabbay, D.; Hogger, C.; and Robinson, J., eds., *Handbook of logic in Artificial Intelligence and logic programming*, volume 3. Clarendon Press - Oxford. 439–513.

Dubois, D.; Moral, S.; and Prade, H. 1998. Belief change rules in ordinal and numerical uncertainty theories. In *Belief Change*. Kluwer. 311–392.

Fishburn, P. C. 1986. The axioms of subjective probabilities. *Statistical Science* 1:335–358.

Freund, M. 2004. On the revision of preferences and rational inference processes. *Artificial Intelligence* 152:105–137.

Friedman, N., and Halpern, J. 1996a. Belief revision: A critique. In *Proceedings of KR'96*, 421–631.

Friedman, N., and Halpern, J. Y. 1996b. Plausibility measures and default reasoning. In *Proceedings of AAAI'96*, 1297–1304.

Gärdenfors, P., and Makinson, D. 1994. Nonmonotonic inference based on expectations. *Artificial Intelligence* 65:197–245.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Grove, A. 1988. Two modellings for theory change. *J. Philos. Logic* 17:157–170.

Jeffrey, R. 1965. *The logic of decision*. McGraw-Hill.

Jin, Y., and Thielscher, M. 2005. Iterated revision, revised. In *Proc. IJCAI'05*, 478–483.

Kern-Isberner, G. 2001. *Conditionals in Nonmonotonic Reasoning and Belief Revision*, volume 2087 of *Lecture Notes in Artificial Intelligence*. Springer.

Konieczny, S., and Pino Pérez, R. 2002. Merging information under constraints: a qualitative framework. *J. Logic and Computation* 12(5):773–808.

Kraus, S.; Lehmann, D.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44(1-2):167–207.

Lehmann, D., and Magidor, M. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55:1–60.

Lehmann, D. 1995. Belief revision, revised. In *Proceedings of IJCAI'95*, 1534–1540.

Lewis, D. 1973. *Counterfactuals*. Basil Blackwell, U.K.

Maung, I. 1995. Two characterizations of a minimum information principle for possibilistic reasoning. *Int. J. Approximate Reasoning* 12:133–156.

Nayak, A.; Foo, N.; Pagnucco, M.; and Sattar, A. 1996. Changing conditional beliefs unconditionally. In *Proceedings of TARK96*, 119–135.

Nayak, A.; Pagnucco, M.; and Peppas, P. 2003. Dynamic belief revision operators. *Artificial Intelligence* 146:193–228.

Nayak, A. 1994. Iterated belief change based on epistemic entrenchment. *Erkenntnis*.

Paris, J. 1994. *The Uncertain Reasoner's Companion.* Cambridge University Press, Cambridge.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. 1990. System z: A natural ordering of defaults with tractable applications to default reasoning. In *Proc. of the 3rd Conf. on Theoretical Aspects of Reasoning about Knowledge (TARK'90)*, 121–135. Morgan & Kaufmann, San Mateo, CA.

Shafer, G. 1976. A mathematical theory of evidence. *Princeton University Press.*

Spohn, W. 1988. Ordinal conditional functions: a dynamic theory of epistemic states. In Harper, W. L., and Skyrms, B., eds., *Causation in Decision, Belief Change and Statistics*, volume 2. Kluwer Academic Pub. 105–134.

van Fraassen, B. 1980. Rational belief and probability kinematics. *Philosophy of Science* 47:165–187.

van Fraassen, B. 1981. A problem for relative information minimizers. *British J. Philosophy of Science* 33:375–379.

Weydert, E. 2000. How to revise ranked probabilities. In *Proc.14th Europ.Conf.on Artificial Intelligence (ECAI2000)*, 38–44. IOS Press.

Williams, M. 1995. Iterated theory-based change. In *Proc. of the 14th Inter. Joint Conf. on Artificial Intelligence (IJCAI'95)*, 1541–1550.

2.2 A revision-based approach for handling inconsistency in description logics

A revision-based approach for handling inconsistency in description logics

Guilin Qi, Weiru Liu, David A. Bell
School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast
Belfast, BT7 1NN, UK
{G.Qi, W.Liu, DA.Bell}@qub.ac.uk

Abstract

Recently, the problem of inconsistency handling in description logics has attracted a lot of attention. Many approaches were proposed to deal with this problem based on existing techniques for inconsistency management. In this paper, we first define two revision operators in description logics, one is called the weakening-based revision operator and the other is its refinement. The logical properties of the operators are analyzed. Based on the revision operators, we then propose an algorithm to handle inconsistency in a *stratified* description logic knowledge base. We show that when the weakening-based revision operator is chosen, the resulting knowledge base of our algorithm is semantically equivalent to the knowledge base obtained by applying *refined conjunctive maxi-adjustment* (RCMA) which refines the disjunctive maxi-adjusment (DMA), a good strategy for inconsistency handling in classical logic.

Introduction

Ontologies play a crucial role for the success of the Semantic Web (Berners-Lee, Hendler, and Lassila 2001). There are many representation languages for ontologies, such as description logics (or DLs for short) and F-logic (Staab and Studer 2004). Recently, the problem of inconsistency (or incoherence) handling in ontologies has attracted a lot of attention and research addressing this problem has been reported in many papers (Baader and Hollunder; Baader and Hollunder 1995; Parsia, Sirin, and Kalyanpur 2005; Haase et. al. 2005; Schlobach 2005; Schlobach and Cornet 2003; Flouris, Plexousakis and Antoniou 2005; Huang, Harmelen, and Teije 2005; Meyer, Lee, and Booth 2005; Friedrich and Shchekotykhin 2005). Inconsistency can occur due to several reasons, such as modelling errors, migration or merging ontologies, and ontology evolution. Current DL reasoners, such as RACER (Haarslev and Möller 2005) and FaCT (Horrocks 1998), can detect logical inconsistency. However, they only provide lists of unsatisfiable classes. The process of *resolving* inconsistency is left to the user or ontology engineers. The need to improve DL reasoners to reason with inconsistency is becoming urgent to make them more applicable.Many approaches were proposed to handle inconsistency in ontologies based on existing techniques for inconsistency management in traditional logics, such as propositional logic and nonmonotonic logics (Schlobach and Cornet 2003; Parsia, Sirin, and Kalyanpur 2005; Huang, Harmelen, and Teije 2005).

It is well-known that priority or preference plays an important role in inconsistency handling (Baader and Hollunder; Benferhat and Baida 2004; Meyer, Lee, and Booth 2005). In (Baader and Hollunder), the authors introduced priority to default terminological logic such that more specific defaults are preferred to more general ones. When conflicts occur in reasoning with defaults, defaults which are more specific should be applied before more general ones. In (Meyer, Lee, and Booth 2005), an algorithm, called *refined conjunctive maxi-adjustment* (RCMA for short) was proposed to weaken conflicting information in a *stratified* DL knowledge base and some consistent DL knowledge bases were obtained. To weaken a terminological axiom, they introduced a DL expression, called *cardinality restrictions* on concepts. However, to weaken an assertional axiom, they simply delete it. An interesting problem is to explore other DL expressions to weaken a *conflicting* DL axiom (both terminological and assertional).

In this paper, we first define two revision operators in description logics, one is called a weakening-based revision operator and the other is its refinement. The revision operators are defined by introducing a DL constructor called *nominals*. The idea is that when a terminology axiom or a value restriction is in conflict, we simply add explicit exceptions to weaken it and assume that the number of exceptions is minimal. Based on the revision operators, we then propose an algorithm to handle inconsistency in a *stratified* description logic knowledge base. We show that when the weakening-based revision operator is chosen, the resulting knowledge base of our algorithm is semantically equivalent to that of the RCMA algorithm. However, their syntactical forms are different.

This paper is organized as follows. Section 2 gives a brief review of description logics. We then define two revision operators in Section 3. The revision-based algorithm for inconsistency handling is proposed in Section 4. Before conclusion, we have a brief discussion on related work.

Description logics

In this section, we introduce some basic notions of Description Logics (DLs), a family of well-known knowledge rep-

resentation formalisms (Baader et al. 2003). To make our approach applicable to a family of interesting DLs, we consider the well-known DL $\mathcal{ALC}$ (Schmidt-Schaußand Smolka 1991), which is a simple yet relatively expressive DL. Let N_C and N_R be pairwise disjoint and countably infinite sets of *concept names* and *role names* respectively. We use the letters A and B for concept names, the letter R for role names, and the letters C and D for concept. The set of $\mathcal{ALC}$ concepts is the smallest set such that: (1) every concept name is a concept; (2) if C and D are concepts, R is a role name, then the following expressions are also concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$ and $\exists R.C$.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every concept C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and every role R to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D, role R, the following properties are satisfied:

(1) $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
(2) $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$,
(3) $(\exists R.C)^{\mathcal{I}} = \{x | \exists y \; s.t. (x, y) \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$,
(4) $(\forall R.C)^{\mathcal{I}} = \{x | \forall y (x, y) \in R^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}\}$.

We introduce an extra expression of DLs called *nominals* (also called *individual names*) (Schaerf 1994). A nominal has the form $\{a\}$, where a is an individual name. It can be viewed as a powerful generalization of DL Abox individuals. The semantics of $\{a\}$ is defined by $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$ for an interpretation $\mathcal{I}$. Nominals are included in many DLs, such as $\mathcal{SHOQ}$ (Horrocks and Sattler 2001) and $\mathcal{SHOIQ}$ (Horrocks and Sattler 2005).

A general concept inclusion axiom (GCI) or *terminology* is of the form $C \sqsubseteq D$, where C and D are two (possibly complex) $\mathcal{ALC}$ concepts. An interpretation $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A finite set of $GCIs$ is called a *Tbox*. We can also formulate statements about individuals. We denote individual names as $a,\ b,\ c$. A *concept* (*role*) assertion axiom has the form $C(a)$ ($R(a, b)$), where C is a concept description, R is a role name, and $a,\ b$ are *individual names*. To give a semantics to Aboxes, we need to extend interpretations to individual names. For each individual name a, $\cdot^{\mathcal{I}}$ maps it to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The mapping $\cdot^{\mathcal{I}}$ should satisfy the *unique name assumption* (UNA)[1], that is, if a and b are distinct names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a concept axiom $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, it satisfies a role axiom $R(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An *Abox* contains a finite set of concept and role axioms. A DL knowledge base K consists of a Tbox and an Abox, i.e. it is a set of GCIs and assertion axioms. An interpretation $\mathcal{I}$ is a *model* of a DL (Tbox or Abox) axiom iff it satisfies this axiom, and it is a model of a DL knowledge base K if it satisfies every axiom in K. In the following, we use $M(\phi)$ (or $M(K)$) to denote the set of models of an axiom ϕ (or DL knowledge base K). K is consistent iff $M(K) \neq \emptyset$. Let K be an inconsistent DL knowledge base, a set $K' \subseteq K$ is a *conflict* of K if K' is inconsistent, and any sub-knowledge base $K'' \subset K'$ is consistent. Given a DL knowledge base K and a DL axiom ϕ, we say K *entails* ϕ, denoted as $K \models \phi$, iff $M(K) \subseteq M(\phi)$.

[1] In some very expressive DLs, such as $\mathcal{SHOQ}$, this assumption is dropped. Instead, they use *inequality assertions* of the form $a \neq b$ for individual names a and b, with the semantics that an interpretation $\mathcal{I}$ satisfies $a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Revision Operators for DLs

Definition

Belief revision is a very important topic in knowledge representation. It deals with the problem of consistently accommodating new information received by an existing knowledge base. Recently, Flouris et al. discuss how to apply the famous AGM theory (Gardenfors 1988) in belief revision to DLs and OWL (Flouris, Plexousakis and Antoniou 2005). However, they only evaluate the feasibility of apply the *AGM postulates for contraction* in DLs. There is no explicit construction of a revision operator in their paper. In this subsection, we propose a revision operator for DLs and provide a semantic explanation of this operator.

We need some restrictions on the knowledge base to be revised. First, the original DL knowledge base should be consistent. Second, we only consider inconsistencies arising due to objects explicitly introduced in the Abox. That is, suppose K and K' are the original knowledge base and the newly received knowledge base respectively, then for each conflict K_c of $K \cup K'$, K_c must contain an Abox statement. For example, we exclude the following case: $\top \sqsubseteq \exists R.C \in K$ and $\top \sqsubseteq \forall R.\neg C \in K'$. The handling of conflicting axioms in the Tbox has been discussed in much work recently (Schlobach and Cornet 2003; Parsia, Sirin, and Kalyanpur 2005). In this section, we discuss the resolution of conflicting information which contains assertional axioms in the context of knowledge revision.

We give a method to weaken a GCI first. To weaken a GCI, we simply add some explicit exceptions, and the number of exceptions is called the degree of the weakened GCI.

Definition 1 *Let $C \sqsubseteq D$ be a GCI. A weakened GCI $(C \sqsubseteq D)_{weak}$ of $C \sqsubseteq D$ has the form $(C \sqcap \neg\{a_1\} \sqcap ... \sqcap \neg\{a_n\}) \sqsubseteq D$, where n is the number of individuals to be removed from C. We use $d((C \sqsubseteq D)_{weak}) = n$ to denote the degree of $(C \sqsubseteq D)_{weak}$.*

It is clear that when $d((C \sqsubseteq D)_{weak}) = 0$, $(C \sqsubseteq D)_{weak} = C \sqsubseteq D$. The idea of weakening a GCI is similar to weaken an uncertain rule in (Benferhat and Baida 2004). That is, when a GCI is involved in conflict, instead of dropping it completely, we remove those individuals which cause the conflict.

The weakening of an assertion is simpler than that of a GCI. The weakened assertion ϕ_{weak} of an Abox assertion ϕ is of the form either $\phi_{weak} = \top$ or $\phi_{weak} = \phi$. That is, we either delete it or keep it intact. The degree of ϕ_{weak}, denoted as $d(\phi_{weak})$, is defined as $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top$ and 0 otherwise.

Next, we consider the weakening of a DL knowledge base.

Definition 2 *Let K and K' be two consistent DL knowledge bases. Suppose $K \cup K'$ is inconsistent. A DL knowledge base $K_{weak,K'}$ is a weakened knowledge base of K $w.r.t$ K' if it satisfies:*

- *$K_{weak,K'} \cup K'$ is consistent, and*

- *There is a bijection f from K to $K_{weak,K'}$ such that for each $\phi \in K$, $f(\phi)$ is a weakening of ϕ.*

The set of all weakened base of K $w.r.t$ K' is denoted by $Weak_{K'}(K)$.

In Definition 2, the first condition requires that the weakened base should be consistent with K'. The second condition says that each element in $K_{weak,K'}$ is uniquely weakened from an element in K.

Example 1 *Let $K = \{bird(tweety), bird \sqsubseteq flies\}$ and $K' = \{\neg flies(tweety)\}$, where $bird$ and $flies$ are two concepts and $tweety$ is an individual name. It is easy to check that $K \cup K'$ is inconsistent. Let $K' = \{\top, bird \sqsubseteq flies\}$, $K'' = \{bird(tweety), bird \sqcap \neg\{tweety\} \sqsubseteq flies\}$, then both K' and K'' are weakened bases of K $w.r.t$ K'.*

The degree of a weakened base is defined as the sum of the degrees of its elements.

Definition 3 *Let $K_{weak,K'}$ be a weakened base of a DL knowledge base K $w.r.t$ K'. The degree of K_{weak} is defined as*

$$d(K_{weak,K'}) = \Sigma_{\phi \in K_{weak,K'}} d(\phi)$$

In Example 1, we have $d(K') = d(K'') = 1$.

We now define a revision operator.

Definition 4 *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. The result of weakening-based revision of K $w.r.t$ K', denoted as $K \circ_w K'$, is defined as*

$$K \circ_w K' = \{K' \cup K_i : K_i \in Weak_{K'}(K), \text{ and } \nexists K_j \in Weak_{K'}(K), d(K_j) < d(K_i)\}.$$

The result of revision of K by K' is a set of DL knowledge bases, each of which is the union of K' and a weakened base of K with the minimal degree. $K \circ_w K'$ is a *disjunctive DL knowledge base*[2] defined in (Meyer, Lee, and Booth 2005).

We now consider the semantic aspect of our revision operator.

In (Meyer, Lee, and Booth 2005), an ordering relation was defined to compare interpretations. It was claimed that only two interpretations having the same domain and mapping the same individual names to the same element in the domain can be compared. Given a domain Δ, a *denotation function* d is an injective mapping which maps every individual a to a different $a^{\mathcal{I}}$ in Δ. Then a *pre-interpretation* was defined as an ordered pair $\pi = (\Delta^{\pi}, d^{\pi})$, where Δ^{π} is a domain and d^{π} is a denotation function. For each interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, its denotation function is denoted as $d^{\mathcal{I}}$. Given a pre-interpretation $\pi = (\Delta^{\pi}, d^{\pi})$, $\mathbf{I}^{\pi}$ is used to denote the class of interpretations $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \Delta^{\pi}$ and $d^{\mathcal{I}} = d^{\pi}$. It is also assumed that a DL knowledge base is a multi-set[3] of GCIs and assertion axioms.We now introduce the ordering between two interpretations defined in (Meyer, Lee, and Booth 2005).

[2] A disjunctive DL knowledge (or DKB) is a set of DL knowledge bases. A DKB $\mathcal{K}$ is satisfied by an interpretation $\mathcal{I}$ iff $\mathcal{I}$ is a model of at least one of the elements of $\mathcal{K}$.

[3] A multi-set is a set in which an element can appear more than once.

Definition 5 *Let π be a pre-interpretation, $\mathcal{I} \in \mathbf{I}^{\pi}$, ϕ a DL axiom, and K a multi-set of DL axioms. If ϕ is an assertion, the number of ϕ-exceptions $e^{\phi}(\mathcal{I})$ is 0 if $\mathcal{I}$ satisfies ϕ and 1 otherwise. If ϕ is a GCI of the form $C \sqsubseteq D$, the number of ϕ-exceptions for $\mathcal{I}$ is:*

$$e^{\phi}(\mathcal{I}) = \begin{cases} |C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}})| & \text{if } C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}}) \text{ is finite} \\ \infty & \text{otherwise.} \end{cases} \quad (1)$$

The number of K-exceptions for $\mathcal{I}$ is $e^{K}(\mathcal{I}) = \Sigma_{\phi \in K} e^{\phi}(\mathcal{I})$. The ordering $\preceq_K^{\pi}$ on $\mathbf{I}^{\pi}$ is: $\mathcal{I} \preceq_K^{\pi} \mathcal{I}'$ iff $e^{K}(\mathcal{I}) \leq e^{K}(\mathcal{I}')$.

We give a proposition to show that our weakening-based revision operator captures some kind of minimal change.

Proposition 1 *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. Let Π be the class of all pre-interpretations. $\circ_w$ is the weakening-based revision operator. We then have*

$$M(K \circ_w K') = \cup_{\pi \in \Pi} min(M(K'), \preceq_K^{\pi}).$$

Proposition 1 says that the models of the resulting knowledge base of our revision operator are models of K' which are minimal $w.r.t$ the ordering $\preceq_K^{\Pi}$ induced by K. The proofs of proposition 2 and other propositions can be found in the appendix.

Let us look at an example.

Example 2 *Let $K = \{\forall hasChild.RichHuman(Bob), hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom)\}$. Suppose we now receive new information $K' = \{hasChild\ (Bob, John), \neg RichHuman(John)\}$. It is clear that $K \cup K'$ is inconsistent. Since $\forall hasChild.\ RichHuman(Bob)$ is the only assertion axiom involved in conflict with K', we only need to delete it to restore consistency, that is, $K \circ_w K' = \{hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom), hasChild(Bob, John), \neg RichHuman(John)\}$.*

Refined weakening-based revision

In weakening-based revision, to weaken a conflicting assertion axiom, we simply delete it. However, this may result in counterintuitive conclusions. In Example 2, after revising K by K' using the weakening-based operator, we cannot infer that $RichHuman(Tom)$ because $\forall hasChild.RichHuman(Bob)$ is discarded, which is counterintuitive. From $hasChild(Bob, Tom)$ and $\forall hasChild.RichHuman(Bob)$ we should have known that $RichHuman(Tom)$ and this assertion is not in conflict with information in K'. The solution for this problem is to treat *John* as an *exception* and that all children of *Bob* other than *John* are rich humans.

Next, we propose a new method for weakening Abox assertions. For an Abox assertion of the form $\forall R.C(a)$, it is weakened by dropping some individuals which are related to the individual a by the relation R, i.e. its weakening has the form $\forall R.(C \sqcup \{b_1, ..., b_n\})(a)$, where b_i $(i = 1, n)$ are individuals to be dropped. For other Abox assertions ϕ, we either keep them intact or replace them by $\top$.

Definition 6 *Let ϕ be an assertion in an Abox. A weakened assertion ϕ_{weak} of ϕ is defined as:*

$$\phi_{weak} = \begin{cases} \forall R.(C \sqcup \{b_1, ..., b_n\})(a) & \text{if } \phi = \forall R.C(a) \\ \top \text{ or } \phi & \text{otherwise.} \end{cases} \quad (2)$$

The degree of ϕ_{weak} is $d(\phi_{weak}) = n$ if $\phi = \forall R.C$ and $\phi_{weak} = \forall R.(C \sqcup \{b_1, ..., b_n\})(a)$, $d(\phi_{weak}) = 1$ if $\phi \neq \forall R.C$ and $\phi_{weak} = \top$ and $d(\phi_{weak}) = 0$ otherwise.

We call the weakened base obtained by applying weakening of GCIs in Definition 1 and weakening of assertions in Definition 6 as a refined weakened base. We then replace the weakened base by the refined weakened base in Definition 4 and get a new revision operator, which we call a refined weakening-based revision operator and is denoted as $\circ_{rw}$.

Let us have a look at Example 2 again.

Example 3 *(Example 2 Continued) According to our discussion before, $\forall hasChild.RichHum- an(Bob)$ is the only assertion axiom involved in conflict in K and John is the only* exception *which makes $\forall hasChild.RichHuman(Bob)$ conflicting, so $K \circ_{rw} K' = \{\forall hasChild.(RichHuman \sqcup \{John\})(Bob), hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom), hasChild(Bob, John), \neg RichHuman(John)\}$. We then can infer that RichHuman(Tom) from $K \circ_{rw} K'$.*

To give a semantic explanation of the refined weakening-based revision operator, we need to define a new ordering between interpretations.

Definition 7 *Let π be a pre-interpretation, $\mathcal{I} \in \mathbf{I}^\pi$, ϕ a DL axiom, and K a multi-set of DL axioms. If ϕ is an assertion of the form $\forall R.C(a)$, the number of ϕ-exceptions for $\mathcal{I}$ is:*

$$e_r^\phi(\mathcal{I}) = \begin{cases} |R^\mathcal{I}(a^\mathcal{I}) \cap (\neg C^\mathcal{I})| & \text{if } R^\mathcal{I}(a^\mathcal{I}) \cap (\neg C^\mathcal{I}) \text{ is finite} \\ \infty & \text{otherwise,} \end{cases} \quad (3)$$

where $R^\mathcal{I}(a^\mathcal{I}) = \{b \in \Delta^\mathcal{I} : (a^\mathcal{I}, b) \in R^\mathcal{I}\}$. If ϕ is an assertion which is not of the form $\forall R.C(a)$, the number of ϕ-exceptions $e_r^\phi(I)$ is 0 if $\mathcal{I}$ satisfies ϕ and 1 otherwise. If ϕ is a GCI of the form $C \sqsubseteq D$, the number of ϕ-exceptions for $\mathcal{I}$ is:

$$e_r^\phi(\mathcal{I}) = \begin{cases} |C^\mathcal{I} \cap (\neg D^\mathcal{I})| & \text{if } C^\mathcal{I} \cap (\neg D^\mathcal{I}) \text{ is finite} \\ \infty & \text{otherwise.} \end{cases} \quad (4)$$

The number of K-exceptions for $\mathcal{I}$ is $e_r^K(\mathcal{I}) = \Sigma_{\phi \in K} e_r^\phi(\mathcal{I})$. The refined ordering $\preceq_{r,K}^\pi$ on $\mathbf{I}^\pi$ is: $\mathcal{I} \preceq_{r,K}^\pi \mathcal{I}'$ iff $e_r^K(\mathcal{I}) \leq e_r^K(\mathcal{I}')$.

We have the following propositions for the refined weakening-based revision operator.

Proposition 2 *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. Let Π be the class of all pre-interpretations. $\circ_{rw}$ is the weakening-based revision operator. We then have*

$$M(K \circ_{rw} K') = \cup_{\pi \in \Pi} min(M(K'), \preceq_{r,K}^\pi).$$

Proposition 2 says that the refined weakening-based operator can be accomplished with minimal change.

Proposition 3 *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. We then have*

$$K \circ_{rw} K' \models \phi, \ \forall \phi \in K \circ_w K'.$$

By Example 3, the converse of Proposition 3 is false. Thus, we have shown that the resulting knowledge base of the refined weakening-based revision contains more important information than that of the weakening-based revision.

Logical properties of the revision operators

In belief revision theory, a set of postulates or logical properties are proposed to characterize a "rational" revision operator. The most famous postulates are so-called AGM postulates (Gardenfors 1988) which were reformulated in (Katsuno and Mendelzon 1992). We now generalize AGM postulates for revision to DLs.

Definition 8 *Given two DL knowledge bases K and K'. A revision operator $\circ$ is said to be AGM-compliant if it satisfies the following properties:*
(R1) $K \circ K' \models \phi$ *for all* $\phi \in K'$
(R2) *If* $K \cup K'$ *is consistent, then* $M(K \circ K') = M(K \cup K')$
(R3) *If* K' *is consistent, then* $K \circ K'$ *is also consistent*
(R4) *If* $M(K) = M(K_1)$ *and* $M(K') = M(K_2)$*, then* $M(K \circ K') = M(K_1 \circ K_2)$
(R5) $M(K \circ K') \cap M(K'') \subseteq M(K \circ (K' \cup K''))$
(R6) *If* $M(K \circ K') \cap M(K'')$ *is not empty, then* $M(K \circ (K' \cup K'')) \subseteq M(K \circ K') \cap M(K'')$

(R1) says that the new information must be accepted. (R2) requires that the result of revision be equivalent to the union of the existing knowledge base and the newly arrived knowledge base if this union is satisfiable. (R3) is devoted to the satisfiability of the result of revision. (R4) is the syntax-irrelevance condition. (R5) and (R6) together are used to ensure minimal change. (R4) states that the operator is independent of the syntactical form of both the original knowledge base and the new knowledge base. The following property is obviously weaker than (R4)
(*R4'*) If $M(K_1) = M(K_2)$, then $M(K \circ K_1) = M(K \circ K_2)$.

Definition 9 *A revision operator $\circ$ is said to be quasi-AGM compliant if it satisfies (R1)-(R3), (R4'), (R5-R6).*

The following proposition tells us the logical properties of our revision operators.

Proposition 4 *Given two DL knowledge bases K and K'. Both the weakening-based revision operator and the refined weakening-based revision operator are not AGM-compliant but they satisfy postulates (R1), (R2), (R3), (R4'), (R5) and (R6), that is, they are quasi-AGM compliant.*

Proposition 4 is a positive result. Our revision operators satisfy all the AGM postulates except (R4), i.e. the syntax-irrelevant condition.

A Revision-based Algorithm

It is well-known that priorities or preferences play an important role in inconsistency handling (Baader and Hollunder; Benferhat and Baida 2004; Benferhat et al. 2004; Meyer,

Lee, and Booth 2005). In this section, we define an algorithm for handling inconsistency in a stratified DL knowledge base, i.e. each element of the base is assigned a rank, based on the weakening-based revision operator. More precisely, a stratified DL knowledge base is of the form $\Sigma = K_1 \cup ... \cup K_n$, where for each $i \in \{1, ..., n\}$, K_i is a finite multi-set of DL sentences. Sentences in each stratum K_i have the same rank or reliability, while sentences contained in K_j such that $j > i$ are seen as less reliable.

Revision-based algorithm

We first need to generalize the (refined) weakening-based revision by allowing the newly received DL knowledge base to be a disjunctive DL knowledge base. That is, we have the following definition.

Definition 10 *Let K be a consistent DL knowledge base. $\mathcal{K}'$ is a newly received disjunctive DL knowledge base. The result of (refined) weakening-based revision of K $w.r.t$ $\mathcal{K}'$, denoted as $K \circ_w \mathcal{K}'$, is defined as*

$$K \circ_w \mathcal{K}' = \{K' \cup K_{weak,K'} : K' \in \mathcal{K}',\ K_{weak,K'} \in Weak_{K'}(K)\ \&\ \nexists K_i \in Weak_{K'}(K), d(K_i) < d(K_{weak,K'})\}.$$

Revision-based Algorithm (R-Algorithm)
Input: a stratified DL knowledge base $\Sigma = \{K_1, ..., K_n\}$, a (refined) weakening-based revision operator $\circ$ (i.e. $\circ = \circ_w$ or $\circ_{rw}$), a new DL knowledge base K
Result: a disjunctive DL knowledge base $\mathcal{K}$
begin
  $\mathcal{K} \leftarrow K_1 \circ K$;
  for $i = 2$ to n **do**
    $\mathcal{K} \leftarrow K_i \circ \mathcal{K}$;
  return $\mathcal{K}$
end

The idea originates from the revision-based algorithms proposed in (Qi, Liu, and Bell 2005). That is, we start by revising the set of sentences in the first stratum using the new DL knowledge base K, and the result of revision is a disjunctive knowledge base. We then revise the set of sentences in the second stratum using the disjunctive knowledge base obtained by the first step, and so on.

Example 4 *Let $\Sigma = (K_1, K_2)$ and $K = \{\top\}$, where $K_1 = \{W(t), \neg F(t), B(c)\}$ and $K_2 = \{B \sqsubseteq F, W \sqsubseteq B\}$ (W, F, B, t and c abbreviate $Wing$, $Flies$, $Bird$, $Tweety$ and $Chirpy$). Let $\circ = \circ_w$ in R-Algorithm. Since K_1 is consistent, we have $\mathcal{K} = K_1 \circ_w \{\top\} = \{K_1\}$. Since $K_1 \cup K_2$ is inconsistent, we need to weaken K_2. Let $K_2' = \{B \sqcap \neg\{t\} \sqsubseteq F, W \sqsubseteq B\}$ and $K_2'' = \{B \sqsubseteq F, W \sqcap \neg\{t\} \sqsubseteq B\}$, so K_2', $K_2'' \in Weak(K_2)$ and $d(K_2') = d(K_2'') = 1$. It is easy to check that $K_2' \cup K_1$ and $K_2'' \cup K_1$ are both consistent and they are the only weakened bases of K_2 which are consistent with K_1. So $K_2 \circ_w \mathcal{K} = \{K_1 \cup K_2', K_1 \cup K_2''\} = \{\{W(t), \neg F(t), B(c), B \sqcap \neg\{t\} \sqsubseteq F, W \sqsubseteq B\}, \{W(t), \neg F(t), B(c), B \sqsubseteq F, W \sqcap \neg\{t\} \sqsubseteq B\}\}$. It is easy to check that $F(c)$ can be inferred from $K_2 \circ_w \mathcal{K}$.*

Based on Proposition 3, it is easy to prove the following proposition.

Proposition 5 *Let $\Sigma = \{K_1, ..., K_n\}$ be a stratified DL knowledge base and K be a DL knowledge base. Suppose $\mathcal{K}_1$ and $\mathcal{K}_2$ are disjunctive DL knowledge bases resulting from R-Algorithm using the weakening-based operator and refined weakening-based operator respectively. We then have, for each DL axiom ϕ, if $\mathcal{K}_1 \models \phi$ then $\mathcal{K}_2 \models \phi$.*

Proposition 5 shows that the resulting knowledge base of R-Algorithm $w.r.t$ the refined weakening-based operator contains more important information than that of R-Algorithm $w.r.t$ the weakening-based operator.

In the following we show that if the weakening-based revision operator is chosen, then our revision-based approach is equivalent to the refined conjunctive maxi-adjustment (RCMA) approach (Meyer, Lee, and Booth 2005). The RCMA approach is defined in a model-theoretical way as follows.

Definition 11 *(Meyer, Lee, and Booth 2005) Let $\Sigma = (K_1, ..., K_n)$ be a stratified DL knowledge base. Let Π be the class of all pre-interpretations. Let $\pi \in \Pi$, $\mathcal{I}$, $\mathcal{I}' \in \mathbf{I}^\pi$. The lexicographically combined preference ordering $\preceq^\pi_{lex}$ is defined as $\mathcal{I} \preceq^\pi_{lex} \mathcal{I}'$ iff $\forall j \in \{1, ..., n\}$, $\mathcal{I} \preceq^\pi_{K_j} \mathcal{I}'$ or $\mathcal{I} \prec^\pi_{K_i} \mathcal{I}'$ for some $i < j$. Then the set of models of the consistent DL knowledge base extracted from Σ by means of $\preceq^\pi_{lex}$ is $\cup_{\pi \in \Pi} min(\mathbf{I}^\pi, \preceq^\pi_{lex})$.*

The following proposition shows that our revision-based approach is equivalent to the RCMA approach when the weakening-based revision operator is chosen.

Proposition 6 *Let $\Sigma = (K_1, ..., K_n)$ be a stratified DL knowledge base and $K = \{\top\}$. Let $\mathcal{K}$ be the resulting DL knowledge base of R-Algorithm. We then have*

$$M(\mathcal{K}) = \cup_{\pi \in \Pi} min(\mathbf{I}^\pi, \preceq^\pi_{lex}).$$

In (Meyer, Lee, and Booth 2005), an algorithm was proposed to compute the RCMA approach in a syntactical way. The main difference between our algorithm and the RCMA algorithm is that the strategies for resolving terminological information are different. The RCMA algorithm uses a preprocess to transform all the GCIs $C_i \sqsubseteq D_i$ to cardinality restrictions (Baader, Buchheit, and Hollander 1996) of the form $\leq_0 C_i \sqcap \neg D_i$, i.e. the concepts $C_i \sqcap \neg D_i$ do not have any elements. Then those conflicting cardinality restrictions $\leq_0 C_i \sqcap D_i$ are weakened by relaxing the restrictions on the number of elements C may have, i.e. a weakening of $\leq_0 C_i \sqcap D_i$ is of the form $\leq_n C_i \sqcap D_i$ where $n > 1$. The resulting knowledge base contains cardinality restrictions and assertions and is no longer a DL knowledge base in a strict sense. By contrast, our algorithm weakens the GCIs by introducing *nominal* and role constructors. So the resulting DL knowledge base of our algorithm still contains GCIs and assertions.

Application to revising a stratified DL knowledge base

We can define two revision operators based on R-Algorithm. Let $\Sigma = (K_1, ..., K_n)$ be a stratified knowledge base and

K be a new DL knowledge base. Let $\circ$ be the (refined) weakening-based revision operator. The prioritized (refined) weakening-based revision operator, denoted as $\circ^g$, is defined in a model-theoretic way as: $M(\Sigma \circ^g K) = \cup_{\pi \in \Pi} min(\{\mathcal{I} \in \mathbf{I}^\pi : \mathcal{I} \models K\}, \preceq^\pi_{lex})$. We now look at the logical properties of the newly defined operator.

Proposition 7 *Let Σ be a stratified DL knowledge base, K and K' be two DL knowledge bases. The revision operator $\circ^g$ satisfies the following properties:*

(P1) If K is satisfiable, then $\Sigma \circ^g K$ is satisfiable.

(P2) $\Sigma \circ^g K \models \phi$, for all $\phi \in K$.

(P3) If $M(\Sigma) \cap M(K)$ is not empty, then $M(\Sigma \circ^g K) = M(\Sigma) \cap M(K)$.

(P4) Given a stratified DL knowledge base $K = \{S_1, ..., S_n\}$, and two DL knowledge bases K and K', if $K \equiv K'$, then $Mod(\Sigma \circ^g K) = Mod(\Sigma \circ^g K')$.

(P5) $M(\Sigma \circ^g K') \cap M(K'') \subseteq M(\Sigma \circ^g (K' \cup K''))$.

(P6) If $M(\Sigma \circ^g K') \cap M(K'')$ is not empty, then $M(\Sigma \circ^g (K' \cup K'')) \subseteq M(\Sigma \circ^g K') \cap M(K'')$.

(P1)-(P3) correspond to Conditions (R1)-(R3) in Definition 8. (P4) is a generalization of the weakening condition (R4') of the principle of irrelevance of syntax. (P5) and (P6) are generalization of (R5) and (R6).

Related Work

This work is closely related to the work on inconsistency handling in propositional and first-order knowledge bases in (Benferhat et al. 2004; Benferhat and Baida 2004), the work on knowledge integration in DLs in (Meyer, Lee, and Booth 2005) and the work on revising-based inconsistency handling approaches in (Qi, Liu, and Bell 2005). In (Benferhat et al. 2004), a very powerful approach, called disjunctive maxi-adjustment (DMA) approach, was proposed for weakening conflicting information in a stratified propositional knowledge base. The basic idea of the DMA approach is that starting from the information with the lowest stratum where formulae have highest level of priority, when inconsistency is encountered in the knowledge base, it weakens the conflicting information in those strata. When applied to a first-order knowledge base directly, the DMA approach is not satisfactory because some important information is lost. A new approach was proposed in (Benferhat and Baida 2004). For a first-order formula, called an *uncertain rule*, with the form $\forall x P(x) \Rightarrow Q(x)$, when it is involved in a conflict in the knowledge base, instead of deleting it completely, the formula is weakened by dropping some of the instances of this formula that are responsible for the conflict. The idea of weakening GCIs in Definition 1 is similar to this idea. In (Meyer, Lee, and Booth 2005), the authors proposed an algorithm for inconsistency handling by transforming every GCI in a DL knowledge base into a cardinality restriction, and a cardinality restriction responsible for a conflict is weakened by relaxing the restrictions on the number of elements it may have. So their strategy of weakening GCIs is different from ours. Furthermore, we proposed a refined revision operator which not only weakens the GCIs but also assertions of the form $\forall R.A(a)$. The idea of applying revision operators to deal with inconsistency in a stratified knowledge base was proposed in (Qi, Liu, and Bell 2005). However, this work is only applicable in propositional stratified knowledge bases. The R-Algorithm is a successful application of the algorithm to DL knowledge bases.

There are many other work on inconsistency handling in DLs (Baader and Hollunder; Baader and Hollunder 1995; Parsia, Sirin, and Kalyanpur 2005; Quantz and Royer 1992; Haase et. al. 2005; Schlobach 2005; Schlobach and Cornet 2003; Flouris, Plexousakis and Antoniou 2005; Huang, Harmelen, and Teije 2005; Friedrich and Shchekotykhin 2005). In (Baader and Hollunder 1995; Baader and Hollunder), Reiter's default logic (Reiter 1987) is embedded into terminological representation formalisms, where conflicting information is treated as *exceptions*. To deal with conflicting default rules, each rule is instantiated using individuals appearing in an Abox and two existing methods are applied to compute all extensions. However, in practical applications, when there is a large number of individual names, it is not advisable to instantiate the default rules. Moreover, only conflicting default rules are dealt with and it is assumed that information in the Abox is absolutely true. This assumption is dropped in our algorithm, that is, an assertion in an Abox may be weakened when it is involved in a conflict. Another work on handling conflicting defaults can be found in (Quantz and Royer 1992). The authors proposed a preference semantics for defaults in terminological logics. As pointed out in (Meyer, Lee, and Booth 2005), this method does not provide a weakening of the original knowledge base and the formal semantics is not cardinality-based. Furthermore, it is also assumed that information in the Abox was absolutely true. In recent years, several methods have been proposed to debug erroneous terminologies and have them repaired when inconsistencies are detected (Schlobach and Cornet 2003; Schlobach 2005; Parsia, Sirin, and Kalyanpur 2005; Friedrich and Shchekotykhin 2005). A general framework for reasoning with inconsistent ontologies based on *concept relevance* was proposed in (Huang, Harmelen, and Teije 2005). The idea is to select from an inconsistent ontology some consistent subtheories based on a *selection function*, which is defined on the syntactic or semantic relevance. Then standard reasoning on the selected sub-theories is applied to find *meaningful* answers. A problem with debugging of erroneous terminologies methods in (Schlobach and Cornet 2003; Schlobach 2005; Parsia, Sirin, and Kalyanpur 2005; Friedrich and Shchekotykhin 2005) and the reasoning method in (Huang, Harmelen, and Teije 2005) is that both approaches delete terminologies in a DL knowledge base to obtain consistent subbases, thus the structure of DL language is not exploited.

Conclusions and Further Work

In this paper, we propose a revision-based algorithm for handling inconsistency in description logics. We mainly considered the following issues:

1. A weakening-based revision operator was defined in both syntactical and semantic ways. Since the weakening-based revision operator may result in counter-intuitive

conclusions in some cases, we defined a refined version of this operator by introducing additional expressions in DLs.

2. The well-known AGM postulates are reformulated and we showed that our operators satisfy most of the postulates. Thus they have good logical properties.

3. A revision-based algorithm was presented to handle inconsistency in a stratified knowledge base. When the weakening-based revision operator is chosen, the resulting knowledge base of our algorithm is semantically equivalent to that of the RCMA algorithm. The main difference between our algorithm and the RCMA algorithm is that the strategies for resolving terminological information are different.

4. Two revision operators were defined on stratified DL knowledge bases and their logical properties were analyzed.

There are many problems worthy of further investigation. Our R-Algorithm is based on two particular revision operators. Clearly, if a normative definition of revision operators in DLs is provided, then R-Algorithm can be easily extended. Unfortunately, such a definition does not exist now. As far as we know, the first attempt to deal with this problem can be found in (Flouris, Plexousakis and Antoniou 2005). However, the authors only studied the feasibility of AGM's postulates for a *contraction* operator and their results are not so positive. That is, they showed that in many important DLs, such as $\mathcal{SHOIN}(\mathcal{D})$ and $\mathcal{SHIQ}$, it is impossible to define a contraction operator that satisfies the AGM postulates. Moreover, they didn't apply AGM's postulates for a revision operator and explicit construction of a revision operator was not considered in their paper. We generalized AGM postulates for revision in Definition 8 and we showed that our operators satisfied most of the generalized postulates. An important future work is to construct a revision operator in DLs which satisfies all the generalized AGM postulates.

Proofs

Proof of Proposition 1: Before proving Proposition 1, we need to prove two lemmas.

Lemma 1 *Let K and K' be two consistent DL knowledge bases and $\mathcal{I}$ be an interpretation such that $\mathcal{I} \models K'$. Suppose $K \cup K'$ is inconsistent. Let $l = min(d(K_{weak,K'}) : K_{weak,K'} \in Weak_{K'}(K), \mathcal{I} \models K_{weak,K'})$. Then $e^K(\mathcal{I}) = l$.*

Proof: We only need to prove that for each $K_{weak,K'} \in Weak_{K'}(K)$ such that $\mathcal{I} \models K_{weak,K'}$ and $d(K_{weak,K'}) = l$, $e^K(\mathcal{I}) = d(K_{weak,K'})$.

(1) Let $\phi \in K$ be an assertion axiom. Suppose $e^\phi(I) = 1$, then $\mathcal{I} \not\models \phi$. Since $\mathcal{I} \models K_{weak,K'}$, $\phi \notin K_{weak,K'}$ So $\phi_{weak} = \top$ and then $d(\phi_{weak}) = 1$. Conversely, suppose $d(\phi_{weak}) = 1$, then $\phi_{weak} = \top$. We must have $\mathcal{I} \not\models \phi$. Otherwise, let $K''_{weak,K'} = (K_{weak,K'} \setminus \{\top\}) \cup \{\phi\}$. Since $\mathcal{I} \models \phi$, then $K''_{weak,K'}$ is consistent. It is clear $d(K''_{weak,K'}) < d(K_{weak,K'})$, which is a contradiction. So $\mathcal{I} \not\models \phi$, we then have $e^\phi(\mathcal{I}) = 1$. Thus, $e^\phi = 1$ iff $d(\phi) = 1$.

(2) Let $\phi = C \sqsubseteq D$ be a GCI axiom and $\phi_{weak} = (C \sqsubseteq D)_{weak} \in K_{weak,K'}$. Suppose $d(\phi_{weak}) = n$. That is, $\phi_{weak} = C \sqcap \neg\{a_1, ..., a_n\} \sqsubseteq D$. Since $\mathcal{I} \models K_{weak,K'}$, $\mathcal{I} \models \phi_{weak}$. Moreover, for any other weakening ϕ'_{weak} of ϕ, if $d(\phi'_{weak}) < n$, then $\mathcal{I} \not\models \phi'_{weak}$ (because otherwise, we find another weakening $K'_{weak,K'} = (K_{weak,K'} \setminus \{\phi_{weak}\}) \cup \{\phi'_{weak}\}$ such that $d(K'_{weak,K'}) < d(K_{weak,K'})$ and $\mathcal{I} \models K'_{weak,K'}$). Since $\mathcal{I} \models \phi_{weak}$, $C^\mathcal{I} \setminus \{a_1^\mathcal{I}, ..., a_n^\mathcal{I}\} \subseteq D^\mathcal{I}$. For each a_i, we must have $a_i \in C$ and $a_i \notin D$. Otherwise, we can delete such a_i and obtain $\phi'_{weak} = C \sqcap \{a_1, ..., a_{i-1}, a_{i+1}, ..., a_n\} \sqsubseteq D$ such that $d(\phi'_{weak}) < d(\phi_{weak})$ and $\mathcal{I} \models \phi'_{weak}$, which is a contradiction. So $|C^\mathcal{I} \cap \neg D^\mathcal{I}| \leq n$. Since for each a_i, let $\phi'_{weak} = C \sqcap \{a_1, ..., a_{i-1}, a_{i+1}, ..., a_n\} \sqsubseteq D$, then $\mathcal{I} \not\models \phi'_{weak}$, so $|C^\mathcal{I} \cap \neg D^\mathcal{I}| \geq n$. Therefore, we have $|C^\mathcal{I} \cap \neg D^\mathcal{I}| = n = d(\phi_{weak})$.

(1) and (2) together show that $e^K(\mathcal{I}) = l$.

Lemma 2 *Let K and K' be two consistent knowledge bases and $\mathcal{I}$ be an interpretation such that $\mathcal{I} \models K'$. Suppose $K \cup K'$ is inconsistent. Let $d_m = min(d(K_{weak,K'}) : K_{weak,K'} \in Weak_{K'}(K))$. Then $\mathcal{I} \in \bigcup_{\pi \in \Pi} min(M(K'), \preceq^\pi_K)$ iff $e^K(\mathcal{I}) = d_m$.*

Proof: "If Part"

Suppose $e^K(\mathcal{I}) = d_m$. By Lemma 1, for each $\mathcal{I}'$ such that $\mathcal{I}' \models K'$, $e^K(\mathcal{I}') = l$, where $l = min(d(K_{weak,K'}) : K_{weak,K'} \in Weak_{K'}(K), \mathcal{I}' \models K_{weak,K'})$. That is, there exits $K_{weak,K'} \in Weak_{K'}(K)$ such that $\mathcal{I}' \models K_{weak,K'}$ and $e^K(\mathcal{I}') = d(K_{weak,K'})$. Since $d(K_{weak,K'}) \leq d_m$, we have $e^K(\mathcal{I}') \leq e^K(\mathcal{I})$. So $\mathcal{I} \in \bigcup_{\pi \in \Pi} min(M(K'), \preceq^\pi_K)$.

"Only If Part"

Suppose $\mathcal{I} \in \bigcup_{\pi \in \Pi} min(M(K'), \preceq^\pi_K)$. We need to prove that for all $\mathcal{I}' \models K'$, $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$. Suppose $\mathcal{I} \in \mathbf{I}^\pi$ for some $\pi = (\Delta^\pi, d^\pi)$. It is clear that $\forall \mathcal{I}' \in \mathbf{I}^\pi$, $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$. Now suppose $\mathcal{I}' \in \mathbf{I}^{\pi'}$ for some $\pi' \neq \pi$ such that $\pi' = (\Delta^{\pi'}, d^{\pi'})$. We further assume that $e^K(\mathcal{I}') = min(e^K(\mathcal{I}_i) : \mathcal{I}_i \models K')$. Let $Ind(K)$ and $Ind(K')$ be sets of individual names appearing in K and K' respectively. By unique name assumption, for each individual name a in $Ind(K) \cup Ind(K')$, there is a unique element a_1 in $\Delta^\mathcal{I}$ and a unique element a_2 in $\Delta^{\mathcal{I}'}$ such that $a^\mathcal{I} = a_1$ and $a^{\mathcal{I}'} = a_2$. For notational simplicity, we assume that $a^\mathcal{I} = a^{\mathcal{I}'} = a$ for every individual name a. So $Ind(K) \cup Ind(K') \subseteq \Delta^\pi \cap \Delta^{\pi'}$. We take an $\mathcal{I}'' \in \mathbf{I}^\pi$ which satisfies the following conditions: 1) for each concept C appearing in K, suppose $\Delta = C^\mathcal{I} \cap (Ind(K) \cup Ind(K'))$, then $\Delta \subseteq C^{\mathcal{I}''}$; 2) $e^K(\mathcal{I}'') = min(e^K(\mathcal{I}) : \mathcal{I} \models K' \, \mathcal{I} \in \mathbf{I}^\pi)$. We now prove $\Sigma_{\phi \in K} e^\phi(\mathcal{I}') = \Sigma_{\phi \in K} e^\phi(\mathcal{I}'')$. By 1) and 2), suppose ϕ is an assertion of the form $C(a)$, where C is a concept, then $a^{\mathcal{I}'} \in C^{\mathcal{I}'}$ iff $a^{\mathcal{I}''} \in C^{\mathcal{I}''}$, so $e^\phi(\mathcal{I}') = e^\phi(\mathcal{I}'')$. Suppose ϕ is a GCI of the form $C \sqsubseteq D$ and $b \in C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'}$. Then we must have $b \in Ind(K) \cup Ind(K')$. Otherwise, if we define $\mathcal{I}''' = (\Delta^{\mathcal{I}'} \setminus \{b\}, \cdot^{\mathcal{I}'''})$ such that for each concept name C,

$C^{\mathcal{I}'''} = C^{\mathcal{I}'} \setminus \{b\}$ *and for all* R, $R^{\mathcal{I}'''} = R^{\mathcal{I}'} \setminus (\{(b, a_i) : a_i \in \Delta^{\mathcal{I}'}\} \cup \{(a_i, b) : a_i \in \Delta^{\mathcal{I}'}\})$. *It is easy to check that* $\mathcal{I}''' \models K'$ *and* $e^K(\mathcal{I}''') < e^K(\mathcal{I}')$, *which is a contradiction. So* $b \in C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'} \cap (Ind(K) \cup Ind(K'))$. *Since* $C^{\mathcal{I}'} \cap (Ind(K) \cup Ind(K')) = C^{\mathcal{I}''} \cap (Ind(K) \cup Ind(K'))$ *and* $D^{\mathcal{I}'} \cap (Ind(K) \cup Ind(K')) = D^{\mathcal{I}''} \cap (Ind(K) \cup Ind(K'))$, *we have* $C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'} \cap (Ind(K) \cup Ind(K')) = C^{\mathcal{I}''} \cap \neg D^{\mathcal{I}''} \cap (Ind(K) \cup Ind(K'))$. *It follows that* $b \in C^{\mathcal{I}''} \cap \neg D^{\mathcal{I}''} \cap (Ind(K) \cup Ind(K'))$. *We then have* $C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'} \subseteq C^{\mathcal{I}''} \cap \neg D^{\mathcal{I}''}$. *Similarly, we can prove that* $C^{\mathcal{I}''} \cap \neg D^{\mathcal{I}''} \subseteq C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'}$. *So* $C^{\mathcal{I}''} \cap \neg D^{\mathcal{I}''} = C^{\mathcal{I}'} \cap \neg D^{\mathcal{I}'}$. *That is,* $e^{\phi}(\mathcal{I}) = e^{\phi}(\mathcal{I}'')$. *Thus, we can conclude that* $e^K(\mathcal{I}') = e^K(\mathcal{I}'')$. *Since* $e^K(\mathcal{I}'') = e^K(\mathcal{I})$, *we have* $e^K(\mathcal{I}) = e^K(\mathcal{I}')$. *Therefore, for all* $\mathcal{I}' \models K'$, $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$. *It is clear that there exists an* $\mathcal{I}' \models K'$ *such that* $e^{\mathcal{I}'} = d_m$. *So* $e^K(\mathcal{I}) = d_m$.

We continue the proof of Proposition 1. Suppose $\mathcal{I} \models K \circ_w K'$, then $\mathcal{I} \models K' \cup K_{weak,K'}$, for some $K_{weak,K'} \in Weak_{K'}(K)$ such that $d(K_{weak,K'}) = d_m$ (d_m is defined in Lemma 2). By Lemma 1, $\mathcal{I} \models K'$ and $e^K(\mathcal{I}) = d_m$. By Lemma 2, $\mathcal{I} \in \bigcup_{\pi \in \Pi} min(M(K'), \preceq^{\pi}_K)$. Conversely, suppose $\mathcal{I} \in \bigcup_{\pi \in \Pi} min(M(K'), \preceq^{\pi}_K)$. By Lemma 2, $\mathcal{I} \models K'$ and $e^K(\mathcal{I}) = d_m$. By Lemma 1, $\mathcal{I} \models K' \cup K_{weak,K'}$, for some $K_{weak,K'} \in Weak_{K'}(K)$ such that $d(K_{weak,K'}) = d_m$. So $\mathcal{I} \models K \circ_w K'$. This completes the proof.

Proof of Proposition 2: The proof of Proposition 2 is similar to that of Proposition 1. The only problem is that we need to extend the proofs of Lemma 1 and Lemma 2 by considering the weakening of assertion axioms of the form $\forall R.C(a)$, which can be proved similar to the case of GCIs.

Proof of Proposition 3: We only need to prove that $M(K \circ_{rw} K') \subseteq M(K \circ_w K')$. Suppose $\mathcal{I} \models K \circ_{rw} K'$, then by Proposition 2, $\mathcal{I} \models K'$ and $e^K_r(\mathcal{I}) = min(e^K_r(\mathcal{I}') : \mathcal{I}' \models K')$. We now prove that for any $\mathcal{I}' \neq \mathcal{I}$, $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$. Suppose ϕ is an assertion of the form $\forall R.C(a)$ and $e^{\phi}_r(\mathcal{I}) \geq 1$, then there exists b such that $b^{\mathcal{I}} \in R^{\mathcal{I}}(a^{\mathcal{I}}) \cap (\neg D^{\mathcal{I}})$. Since $\mathcal{I} \not\models \forall R.C(a)$, we have $e^{\phi}(\mathcal{I}) = 1$. Since $e^{\phi}_r(\mathcal{I}') \geq e^{\phi}_r(\mathcal{I})$, we have $e^{\phi}_r(\mathcal{I}') \geq 1$. Similarly, we have $e^{\phi}(\mathcal{I}') = 1$. So $e^{\phi}(\mathcal{I}) = e^{\phi}(\mathcal{I}')$. Suppose $e^{\phi}_r(\mathcal{I}) = 0$ and $e^{\phi}_r(\mathcal{I}') \geq 1$, then $e^{\phi}(\mathcal{I}) = 0 < 1 = e^{\phi}(\mathcal{I}')$. Thus, $e^{\phi}(\mathcal{I}) \leq e^{\phi}(\mathcal{I}')$. If ϕ is an assertion which is not of the form $\forall R.C(a)$ or a GCI, then it is easy to prove that $e^{\phi}(\mathcal{I}) = e^{\phi}(\mathcal{I}')$. Therefore, $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$. By Proposition 1, $\mathcal{I} \in M(K \circ_w K')$.

Proofs of Proposition 4 and Proposition 5: Proposition 4 and Proposition 5 are easily to be checked and we do not provide their proofs here.

Proof of Proposition 6: Let $\mathbf{I}^{\pi}_1 = min(\mathbf{I}^{\pi}, \preceq^{\pi}_{K_1})$, and $\mathbf{I}^{\pi}_i = min(\mathbf{I}^{\pi}_{i-1}, \preceq^{\pi}_{K_i})$ for all $i > 1$. It is clear that $M(\mathcal{K}) = \mathbf{I}^{\pi}_n$. So we only need to prove that $\mathbf{I}^{\pi}_n = min(\mathbf{I}^{\pi}, \preceq^{\pi}_{lex})$. Suppose $\mathcal{I} \in \mathbf{I}^{\pi}_n$, then we must have $\mathcal{I} \in min(\mathbf{I}^{\pi}, \preceq^{\pi}_{lex})$. Otherwise, there exists $\mathcal{I}' \in \mathbf{I}^{\pi}$ such that $\mathcal{I}' \prec_{lex} \mathcal{I}$. That is, there exists i such that $\mathcal{I}' \prec^{\pi}_{K_i} \mathcal{I}$ and $\mathcal{I}' \simeq^{\pi}_{K_j} \mathcal{I}$ for all $j < i$, where $\mathcal{I}' \simeq^{\pi}_{K_j} \mathcal{I}$ means $\mathcal{I}' \preceq^{\pi}_{K_j} \mathcal{I}$ and $\mathcal{I} \preceq^{\pi}_{K_j} \mathcal{I}'$. Since $\mathcal{I}' \simeq^{\pi}_{K_j} \mathcal{I}$, it is clear that $\mathcal{I}, \mathcal{I}' \in \mathbf{I}^{\pi}_{i-1}$ by the definition of $\mathbf{I}^{\pi}_{i-1}$. Since $\mathcal{I} \in \mathbf{I}^{\pi}_n$, we have $\mathcal{I} \in \mathbf{I}^{\pi}_i = min(\mathbf{I}^{\pi}_{i-1}, \preceq^{\pi}_{K_i})$, which is contradictory to the assumption that $\mathcal{I}' \prec^{\pi}_{K_i} \mathcal{I}$. Thus we prove that $\mathbf{I}^{\pi}_n \subseteq min(\mathbf{I}^{\pi}, \preceq^{\pi}_{lex})$. Conversely, suppose $\mathcal{I} \in min(\mathbf{I}^{\pi}, \preceq^{\pi}_{lex})$, then we must have $\mathcal{I} \in \mathbf{I}^{\pi}_n$. Otherwise, there exists an i such that $\mathcal{I} \notin \mathbf{I}^{\pi}_i$ and $\mathcal{I} \in \mathbf{I}^{\pi}_j$ for all $j < i$. Suppose $\mathcal{I}' \in \mathbf{I}^{\pi}_i$, then $\mathcal{I}' \in \mathbf{I}^{\pi}_j$ for all $j < i$. We then have $\mathcal{I}' \simeq^{\pi}_{K_j} \mathcal{I}$ for all $j < i$. Since $\mathcal{I}' \in \mathbf{I}^{\pi}_i$ and $\mathcal{I} \notin \mathbf{I}^{\pi}_i$, it follows that $\mathcal{I}' \prec^{\pi}_{K_i} \mathcal{I}$. That is, $\mathcal{I}' \prec^{\pi}_{lex} \mathcal{I}$, which is a contradiction. Thus we prove that $min(\mathbf{I}^{\pi}, \preceq^{\pi}_{lex}) \subseteq \mathbf{I}^{\pi}_n$. This completes the proof.

References

F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms, *Journal of Automated Reasoning*, 14(1):149-180, 1995.

F. Baader and B. Hollunder. Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic, *Journal of Automated Reasoning*, 15(1): 41-68, 1995.

F. Baader, M. Buchheit, and B. Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88:195-213, 1996.

F. Baader, D.L. McGuiness, D. Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, implementation and application*, Cambridge University Press, 2003.

S. Benferhat, C. Cayrol, D. Dubois, L. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In*Proceedings of IJCAI'93*, 640-645, 1993.

S. Benferhat, and R.E. Baida. A stratified first order logic approach for access control. *International Journal of Intelligent Systems*, 19:817-836, 2004.

S. Benferhat, S. Kaci, D.L. Berre, and M.A. Williams. Weakening conflicting information for iterated revision and knowledge integration. *Artificail Intelligence*, vol. 153(1-2):339-371, 2004.

T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web, *Scientific American*, 284(5):3443, 2001.

G. Flouris, D. Plexousakis and G. Antoniou. On applying the AGM theory to DLs and OWL, In *Proc. of 4th International Conference on Semantic Web (ISWC'05)*, 216-231, 2005.

G. Friedrich and K.M. Shchekotykhin. A General Diagnosis Method for Ontologies, In *Proc. of 4th International Conference on Semantic Web (ISWC'05)*, 232-246, 2005.

P. Gärdenfors, *Knowledge in Flux-Modeling the Dynamic of Epistemic States*, The MIT Press, Cambridge, Mass, 1988.

V. Haarslev and R. Möller, RACER System Description, In *IJCAR'01*, 701-706, 2001.

P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies, In *ISWC'05, LNCA3729*, 353-367, 2005.

I. Horrocks. The FaCT system, In de Swart, H., ed., *Tableaux'98, LNAI 1397*, 307-312, 1998.

I. Horrocks, and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic, In *Proceedings of IJCAI'01*, 199-204, 2001.

I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$, In *Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 448-453, 2005.

Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies, In *Proceedings of IJCAI'05*, 254-259, 2005.

H. Katsuno and A.O. Mendelzon. Propositional Knowledge Base Revision and Minimal Change, *Artificial Intelligence*, 52(3): 263-294, 1992.

C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains, *Journal of Artificial Intelligence Research*, 23:667-726, 2005.

T. Meyer, K. Lee, and R. Booth. Knowledge integration for description logics, In *Proceedings of AAAI'05*, 645-650, 2005.

B. Nebel. *What is Hybrid in Hybrid Representation and Reasoning Systems?*, In F. Gardin and G. Mauri and M. G. Filippini, editors, *Computational Intelligence II: Proc. of the International Symposium Computational Intelligence 1989*, North-Holland, Amsterdam, 217-228, 1990.

B. Parsia, E. Sirin and A. Kalyanpur. Debugging OWL ontologies, In *Proc. of WWW'05*, 633-640, 2005.

J. Quantz and V. Royer. A Preference Semantics for Defaults in Terminological Logics, In *Proc. of the 3th Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 294-305, 1992.

G. Qi, W. Liu, and D.A. Bell. A revision-based approach to resolving conflicting information, In *Proceedings of twenty-first Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 477-484.

R. Reiter. A Theory of Diagnosis from First Principles, *Artificial Intelligence*, 32(1): 57-95, 1987.

A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141-176, 1994.

S. Schlobach, and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies, In *Proceedings of IJCAI'2003*, 355-360, 2003.

S. Schlobach. Diagnosing Terminologies, In *Proc. of AAAI'05*, 670-675, 2005.

M. Schmidt-Schauß, and G. Smolka. Attributive Concept descriptions with complements, *Artificial Intelligence*, 48:1-26, 1991.

S. Staab and R. Studer. *Handbook on Ontologies*, International Handbooks on Information Systems, Springer, 2004.

H. Wang, A.L. Rector, N. Drummond and J. Seidenberg. Debugging OWL-DL Ontologies: A Heuristic Approach, In *Proc. of 4th International Conference on Semantic Web (ISWC'05)*, 745-757, 2005.

2.3 Merging stratified knowledge bases under constraints

Merging stratified knowledge bases under constraints

Guilin Qi, Weiru Liu, David A. Bell
School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast
Belfast, BT7 1NN, UK
{G.Qi, W.Liu, DA.Bell}@qub.ac.uk

Abstract

In this paper, we propose a family of operators for merging stratified knowledge bases under integrity constraints. The operators are defined in a model-theoretic way. Our merging operators can be used to merge stratified knowledge bases where no numerical information is available. Furthermore, the original knowledge bases to be merged can be individually inconsistent. In the flat case, our merging operators are good alternatives of model-based merging operators in the propositional setting. Both logical properties and computational complexity issues of the operators are studied.

Keywords: Belief merging; stratified knowledge base; preference representation

Introduction

Fusion of information coming from different sources is crucial to build an intelligent system (Abidi and Gonzalez 1992; Bloch and Hunter 2001). In classical logic, this problem is often called belief merging, which defines the beliefs (resp. goals) of a group of agents from their individual beliefs (resp. goals). There are mainly two families of belief merging operators: the model-based ones which select some interpretations that are the "closest" to the original bases (Revesz 1997; Konieczny and Pino Pérez 1998; Konieczny and Pino Pérez 2002; Liberatore and Schaerf 1998; Everaere, Konieczny, and Marquis 2005) and the formula-based ones which pick some formulae in the union of the original bases (Baral, Kraus, and Minker 1991; Baral et al. 1992; Konieczny 2000). In (Konieczny, Lang, and Marquis 2004), a class of distance-based merging operators, called DA^2 operators, were defined based on two aggregation functions. DA^2 operators capture many merging operators (including both model-based ones and syntax-based ones) as special cases. In (Everaere, Konieczny, and Marquis 2005), two families of interesting merging operators are proposed. One is called *Quota* operators, which select possible worlds satisfying "sufficient many" bases from the given profile (a multi-set of bases) as the models of the resulting knowledge base. The other is called *Gmin* operators, which are intended to refine quota operators to preserve more information.

It is well-known that priority or preference (either implicit or explicit) plays an important role in many Artificial Intelligence areas, such as inconsistency handling (Benferhat et al. 1993), belief revision (Gärdenfors 1988), belief merging (Benferhat et al. 2002). When explicit priority or preference information is available, a knowledge base is stratified or ranked. In that case, the merging operators in classical logic are not appropriate to merge those knowledge bases because the priority information is not used. Merging of stratified knowledge bases is often handled in the framework of possibilistic logic (Dubois, Lang, and Prade 1994) or ordinal conditional function (Spohn 1988). The merging methods are usually based on the commensurability assumption, that is, all knowledge bases share a common scale (usually ordinal scales such as [0,1]) to order their beliefs. However, this assumption is too strong in practice-we may only have knowledge bases with a total pre-order relation on their elements. Furthermore, different agents may use different strategies to order their beliefs or interpretations. Even a single agent may have different ways of modeling her preferences for different aspects of a problem (Brewka 2004). Without the commensurability assumption, the previous merging methods are hard to apply. For example, suppose there are two agents whose beliefs are represented as $B_1 = \{(p, 3), (q, 2), (r, 1)\}$ and $B_2 = \{(\neg q, 2), (r, 1)\}$ respectively, where the number i ($i = 1, 2.3$) denotes the level of relative importance or priority of a formula. That is, p is more important than q in B_1 and $\neg q$ is more important than r in B_2. Although q and $\neg q$ have the same number (i.e. 2) attached to them, they may not have the same level of importance or priority. In this case, previous merging operators under commensurability assumption cannot be applied to merging B_1 and B_2.

In this paper, we propose a family of operators for merging stratified knowledge bases under integrity constraints. The operators are defined in a model-theoretic way. We assume that each stratified knowledge base is assigned to an ordering strategy. First, for each stratified knowledge base K, the set Ω of possible worlds is stratified as $\Omega_{K,X}$ according to its ordering strategy X. In this way, a possible world has a priority level with regard to each knowledge base which is its priority level in $\Omega_{K,X}$. Second, each possible world or interpretation is associated with a list of *priority levels* in all the original knowledge bases. Then a possible world is viewed as a model of the resulting knowledge base of merging if it is a model of the formula representing the integrity constraint and it is minimal among models of the

integrity constraint $w.r.t$ the lexicographical order induced by the natural order.

The main contributions of this paper are summarized as follows.

(1) First, we define our merging operators in a model-theoretic way. When original knowledge bases are flat, i.e. there is no rank between their elements, some of our operators are reduced to existing classical merging operators.

(2) Second, the commensurability assumption is not necessary for our operators. Moreover, each knowledge base can have its own ordering strategy. By considering the pros and cons of different ordering strategies, we can deal with merging of knowledge bases in a more flexible way.

(3) Third, the original knowledge bases are not necessary to be self-consistent and our operators resolve the conflicting information among different knowledge bases and result in a consistent knowledge base.

(4) Fourth, we provide a family of syntactic methods to merge stratified knowledge bases under integrity constraints. These methods are the syntactical counterparts of our merging operators.

(5) Finally, we generalize the set of postulates proposed in (Konieczny and Pino Pérez 2002) for merging operators applied to stratified knowledge bases and discuss the logical properties of our operators based on these postulates.

This paper is organized as follows. Some preliminaries are introduced in Section 2. In Section 3, we consider the preference representation of stratified knowledge bases. A new ordering strategy is proposed. The Δ^{PLMIN} operators are proposed in Section 4. Section 5 analyzes the computational complexity of our merging operators. We then study the logical properties of our merging operators in Section 6. Section 7 is devoted to discussing related work. Finally, we conclude the paper in Section 8.

Preliminaries

Classical logic: In this paper, we consider a propositional language $\mathcal{L}_{PS}$ from a finite set PS of propositional symbols. The classical consequence relation is denoted as $\vdash$. An interpretation (or world) is a total function from PS to $\{0,1\}$, denoted by a bit vector whenever a strict total order on PS is specified. Ω is the set of all possible interpretations. An interpretation w is a model of a formula ϕ iff $w(\phi)=1$. p,q,r,... represent atoms in PS. We denote formulae in $\mathcal{L}_{PS}$ by ϕ,ψ,γ,... For each formula ϕ, we use $M(\phi)$ to denote its set of models. A *classical knowledge base K* is a finite set of propositional formulae (we can also identify K with the conjunction of its elements). K is consistent iff there exists an interpretation w such that $w(\phi)=true$ for all $\phi\in K$. A knowledge *profile E* is a multi-set of knowledge bases, i.e. $E=\{K_1,...,K_n\}$, where K_i may be identical to K_j for $i\neq j$. Let $\bigcup(E)=\cup_{i=1}^{n}K_i$. Two knowledge profiles E_1 and E_2 are equivalent, denoted $E_1\equiv E_2$ iff there exists a bijection f between E_1 and E_2 such that for each $K\in E_1$, $f(K)\equiv K$.

Stratified knowledge base: A *stratified* knowledge base, sometimes also called ranked knowledge base (Brewka 2004) or prioritized knowledge base (Benferhat et al. 1993), is a set K of (finite) propositional formulas together with a total preorder $\leq$ on K (a preorder is a transitive and reflexive relation, and $\leq$ is a total preorder if either $\phi\leq\psi$ or $\psi\leq\phi$ holds for any $\phi,\psi\in K$)[1]. Intuitively, if $\phi\leq\psi$, then ϕ is considered to be less important than ψ. K can be equivalently defined as a sequence $K=(S_1,...,S_n)$, where each S_i $(i=1,...,n)$ is a non-empty set which contains all the maximal elements of $K\setminus(\cup_{j=1}^{i-1}S_j)$ w.r.t $\leq$ (Coste-Marquis and Marquis 2000), i.e. $S_i=\{\phi\in K\setminus(\cup_{j=1}^{i-1}S_j):\forall\psi\in K\setminus(\cup_{j=1}^{i-1}S_j),\psi\leq\phi\}$. Each subset S_i is called a stratum of K and i the priority level of each formula of S_i. Therefore, the lower the stratum, the higher the priority level of a formula in it. There are many ways to generate a stratified knowledge base (Benferhat et al. 1993; Benferhat and Baida 2004; Brewka 1989; Pearl 1990). A stratified knowledge profile (SKP) E is a multi-set of stratified knowledge bases. Given a stratified knowledge base $K=(S_1,...,S_n)$, the i-cut of K is defined as $K_{\geq i}=S_1\cup...\cup S_i$, for $i\in\{1,...,n\}$. A subbase A of K is also stratified, that is, $A=(A_1,...,A_n)$ such that $A_i\subseteq S_i$, $i=1,...,n$. Two SKPs E_1 and E_2 are equivalent, denoted $E_1\equiv_s E_2$ iff there exists a bijection between E_1 and E_2 such that $n=m$ and for each $K=(S_1,...,S_l)\in E_1$, $f(K)=(S'_1,...,S'_l)$ and $S_i\equiv S'_i$ for all $i\in\{1,...,l\}$.

There are several inconsistency-tolerant inference methods for stratified knowledge bases. In this paper, we use one defined in (Benferhat, Dubois, and Prade 1998) which is related to the consequence relation in possibilistic logic (Dubois, Lang, and Prade 1994).

Definition 1 *Let $K=(S_1,...,S_n)$ be a stratified knowledge base. A formula ϕ is said to be an i-consequence of K, denoted by $K\vdash_i\phi$, if and only if: (1) $K_{\geq i}$ is consistent; (2) $K_{\geq i}\vdash\phi$; (3) $\forall j<i$, $K_{\geq j}\not\vdash\phi$. We say ϕ is a π-consequence of K, denoted by $K\vdash_\pi\phi$, if ϕ is an i-consequence of K for some i.*

Preference Representation of Stratified Knowledge Base

Ordering strategies

Given a stratified knowledge base, we can define some total pre-orders on Ω.

- **best out ordering** (Benferhat et al. 1993): let $r_{BO}(\omega)=min\{i:\omega\not\models S_i\}$, for $\omega\in\Omega$. By convention, we have $min\emptyset=+\infty$. Then the best out ordering $\preceq_{bo}$ on Ω is defined as: $\omega\preceq_{bo}\omega'\ iff\ r_{BO}(\omega)\geq r_{BO}(\omega')$
- **maxsat ordering** (Brewka 2004): let $r_{MO}(\omega)=min\{i:\omega\models S_i\}$, for $\omega\in\Omega$. Then the maxsat ordering $\preceq_{maxsat}$ on Ω is defined as: $\omega\preceq_{maxsat}\omega'\ iff\ r_{MO}(\omega)\leq r_{MO}(\omega')$
- **leximin ordering** (Benferhat et al. 1993): let $K^i(\omega)=\{\phi\in S_i:\omega\models\phi\}$. Then the leximin ordering $\preceq_{leximin}$ on Ω is defined as:

  $\omega\preceq_{leximin}\omega'$ iff $|K^i(\omega)|=|K^i(\omega')|$ for all i, or there is an i such that $|K^i(\omega')|<|K^i(\omega)|$, and for all $j<i$: $|K^j(\omega)|=|K^j(\omega')|$, where $|K_i|$ denote the cardinality of the sets K_i.

[1] For simplicity, we use K to denote a stratified knowledge base and ignore the total preorder $\leq$.

Given a preorder $\preceq$ on Ω, as usual, the associated strict partial order is defined by $\omega \prec \omega'$ iff $\omega \preceq \omega'$ and not $\omega' \preceq \omega$. An ordering $\preceq_X$ is more specific than another $\preceq_{X'}$ iff $\omega \prec_{X'} \omega'$ implies $\omega \prec_X \omega'$. The total preorders on Ω defined above are not independent of each other.

Proposition 1 *(Brewka 2004) Let $\omega, \omega' \in \Omega$, K a stratified knowledge base. The following relationships hold:*
(1) $\omega \prec_{bo} \omega'$ implies $\omega \prec_{leximin} \omega'$;
(2) $\omega \prec_{bo} \omega'$ implies $\omega \preceq_{maxsat} \omega'$ and $\omega \prec_{maxsat} \omega'$ implies $\omega \preceq_{bo} \omega'$

Proposition 1 shows that the $leximin$ ordering is more specific than the best-out ordering.

A new ordering strategy

We now define a new ordering strategy by considering the "distance" between an interpretation and a knowledge base.

Definition 2 *(Everaere, Konieczny, and Marquis 2005) A pseudo-distance between interpretations is a total function d from $\Omega \times \Omega$ to N such that for every ω_1, $\omega_2 \in \Omega$: (1) $d(\omega_1, \omega_2) = d(\omega_2, \omega_1)$; and (2) $d(\omega_1, \omega_2) = 0$ if and only if $\omega_1 = \omega_2$.*

A "distance " between an interpretation ω and a knowledge base S can then be defined as $d(\omega, S) = min_{\omega' \models S} d(\omega, \omega')$. When S is inconsistent, $d(\omega, S) = +\infty$. That is, all the possible worlds have the same distance with an inconsistent knowledge base. Two common examples of such distances are the *drastic distance* d_D and the *Dalal distance* d_H, where $d_D(\omega_1, \omega_2) = 0$ when $\omega_1 = \omega_2$ and 1 otherwise, and $d_H(\omega_1, \omega_2)$ is the Hamming distance between ω_1 and ω_2.

Definition 3 *The distance-based ordering $\preceq_d$ on Ω is defined as:*
$\omega \preceq_d \omega'$ iff $d(\omega, S_i) = d(\omega', S_i)$ for all i, or there is an i such that $d(\omega, S_i) < d(\omega', S_i)$, and for all $j < i$: $d(\omega, S_j) = d(\omega', S_j)$.

It is clear that the distance-based orderings are total preorders on Ω. Suppose $d = d_H$, the ordering $\preceq_{d_H}$ is equivalent to the total preorder $\leq_{K,Lex}$ which is defined to characterize the minimal change of a revision operator in (Qi, Liu, and Bell 2005). The following proposition states the relationships among distance based orderings and other orderings.

Proposition 2 *Let $\omega, \omega' \in \Omega$, and K be a stratified knowledge base. Suppose $d = d_D$ or d_H, then we have:*
(1) $\omega \preceq_d \omega'$ implies $\omega \preceq_{bo} \omega'$ and $\omega \preceq_d \omega'$ implies $\omega \preceq_{maxsat} \omega'$; (2) $\omega \prec_{bo} \omega'$ implies $\omega \prec_d \omega'$

[2]Given a stratified knowledge base K, Ω can be stratified with regard to the total preorder $\preceq$ on it obtained by an ordering strategy X as $\Omega_{K,X} = (\Omega_1, ..., \Omega_m)$ in the same way as stratifying a knowledge base. For two interpretations ω_1, ω_2, if $\omega_1 \in \Omega_i$ and $\omega_2 \in \Omega_j$, where $i < j$, then ω_1 is preferred to ω_2. We use $l_{K,X}(\omega)$ to denote the priority level of the stratum where ω belongs to, i.e. if $\omega \in W_i$, then $l_{K,X}(\omega) = i$.

[2]All proofs of this paper can be found at http://www.cs.qub.ac.uk/∼G.Qi/papers/MergProof.ps

Δ^{PLMIN} Operators

Definition

We use $\preceq_X$ to denote a total preorder on Ω, where X represents an ordering strategy. For example, if $X = bo$, then $\preceq_X$ is the best-out ordering. We now define the Δ^{PLMIN} operators.

Definition 4 *Let $E = \{K_1, ..., K_n\}$ be a multi-set of stratified knowledge bases, where $K_i = \{S_{i1}, ..., S_{im_i}\}$, and μ be an integrity constraint. Let $\mathbf{X} = (X_1, ..., X_n)$ be a set of ordering strategies, where X_i are ordering strategies attached to K_i. Let $\preceq_{K_i,X_i}$ be the total preorder on Ω induced by the ordering strategy X_i. For each interpretation ω, we can associate with it a list of numbers $(l_{K_1,X_1}(\omega), ..., l_{K_n,X_n}(\omega))$, where $l_{K_i,X_i}(\omega)$ is the priority level of the stratum of Ω_{K_i,X_i} where ω belongs to. Let $L_E(\omega) = (l_1(\omega), ..., l_n(\omega))$ be obtained by sorting in increasing order $(l_{K_1,X_1}(\omega), ..., l_{K_n,X_n}(\omega))$, that is, $l_1(\omega) \leq ... \leq l_n(\omega)$. The resulting knowledge base of lexicographical minimum and preference representation based merging operator, denoted by $\Delta_\mu^{PLMIN,\mathbf{X}}(E)$, is defined as follows:*
$\omega \in M(\Delta_\mu^{PLMIN,\mathbf{X}}(E))$ iff $\omega \in M(\mu)$ and $\forall \omega' \in M(\mu)$, $l_i(\omega) = l_i(\omega')$ for all i or $\exists i$ such that $l_i(\omega) < l_i(\omega')$ and $l_j(\omega) = l_j(\omega')$ for all $j < i$.

In Definition 4, each possible world is associated with a list of numbers consisting of the priority levels of the strata of Ω_{K_i} in an increasing order. Then the models of the resulting stratified knowledge base of the $\Delta_\mu^{PLMIN,\mathbf{X}}$ merging operator is the models of the integrity constraint that are minimal $w.r.t$ the lexicographic order induced by the natural order. In our definition, different stratified knowledge bases may have different ordering strategies. That is, each agent can choose her own strategy to order interpretations. We go back to the example in the Introduction section. Suppose the best out ordering strategy is attached to both B_1 and B_2. B_1 and B_2 are stratified as $B_1 = (\{p\}, \{q\}, \{r\})$ and $B_2 = (\{\neg q\}, \{r\})$. Let $\mu = \top$. We have $M(\mu) = \{\omega_1 = pqr, \omega_2 = pq\neg r, \omega_3 = p\neg qr, \omega_4 = p\neg q\neg r, \omega_5 = \neg pqr, \omega_6 = \neg pq\neg r, \omega_7 = \neg p\neg qr, \omega_8 = \neg p\neg q\neg r\}$. It is easy to check that $\Omega_{B_1,bo} = (\{\omega_1\}, \{\omega_2\}, \{\omega_3, \omega_4\}, \{\omega_5, ..., \omega_8\})$ and $\Omega_{B_2,bo} = (\{\omega_3, \omega_7\}, \{\omega_4, \omega_8\}, \{\omega_1, \omega_2, \omega_5, \omega_6\})$. So ω_1 and ω_3 are two minimal possible worlds. That is, the result of merging is $p \wedge r$.

Let us consider the following example.

Example 1 *Let $E = \{K_1, K_2, K_3\}$ be a set of three stratified knowledge bases, where*

- *$K_1 = \{S_{11}, S_{12}, S_{13}\}$, where $S_{11} = \{p_1 \vee p_2, p_3\}$, $S_{12} = \{\neg p_1, \neg p_2, p_2 \vee \neg p_3, p_4\}$, $S_{13} = \{\neg p_3 \vee \neg p_4\}$*
- *$K_2 = \{S_{21}, S_{22}\}$, where $S_{21} = \{p_1, p_2 \vee p_3\}$ and $S_{22} = \{\neg p_2, p_4\}$*
- *$K_3 = \{S_{31}, S_{32}\}$, where $S_{31} = \{p_1, p_3\}$ and $S_{32} = \{p_2\}$.*

The integrity constraint is $\mu = \{\neg p_1 \vee p_2\}$. The set of models of μ is $M(\mu) = \{\omega_1 = 0111, \omega_2 = 0101, \omega_3 = 0110, \omega_4 = 0100, \omega_5 = 0011, \omega_6 = 0001, \omega_7 = 0010, \omega_8 =$

$0000, \omega_9 = 1111, \omega_{10} = 1101, \omega_{11} = 1110, \omega_{12} = 1100\}$. *We denote each model by a bit vector consisting of truth values of* (p_1, p_2, p_3, p_4). *For example,* $\omega_1 = 0111$ *means that the truth value of* p_1 *is 0 and the truth values of other atoms are all 1. Let* $\mathbf{X} = \{X_1, X_2, X_3\}$, *where* $X_1 = X_2 = bo$ *and* $X_3 = d_H$. *That is, the best out ordering strategy is chosen for both* K_1 *and* K_2, *whilst the Dalal distance-based ordering is chosen for* K_3. *The computations are given in Table 1 below.*

ω	K_1	K_2	K_3	E
0111	1	3	3	(1,3,3)
0101	2	3	5	(2,3,5)
0110	1	3	3	(1,3,3)
0100	2	3	5	(2,3,5)
0011	2	3	4	(2,3,4)
0001	2	3	6	(2,3,6)
0010	2	3	4	(2,3,4)
0000	2	3	6	(2,3,6)
1111	**1**	**2**	**1**	**(1,1,2)**
1101	2	2	3	(2,2,3)
1110	**1**	**2**	**1**	**(1,1,2)**
1100	2	2	3	(2,2,3)

Table 1: $\Delta_\mu^{PLMIN,\mathbf{X}}$ *operator*

In Table 1, the column corresponding to K_i *gives the priority levels of strata of* Ω_{K_i} *where* ω_i *belongs to (*Ω *is stratified by an ordering strategy induced by* K_i*). The column corresponding to* E *gives the lists of numbers of the priority levels of possible worlds in an ascending order. Let us explain how to obtain the column corresponding to* K_2 *(other columns can be obtained similarly). Let* $\omega_{13} = 1011$, $\omega_{14} = 1001$, $\omega_{15} = 1010$ *and* $\omega_{16} = 1000$. *Since* $r_{BO}(\omega_i) = 1$ *for all* $1 \leq i \leq 8$, $r_{BO}(\omega_i) = 2$ *for* $9 \leq i \leq 12$ *and* $14 \leq i \leq 16$, $r_{BO}(\omega_{13}) = +\infty$, *we have* $\Omega_{K_2,bo} = (\{\omega_{13}\}, \{\omega_9, ..., \omega_{12}, \omega_{14}, ..., \omega_{16}\}, \{\omega_1, ..., \omega_8\})$. *So* $l_{K_2,bo}(\omega_i) = 3$ *for* $1 \leq i \leq 8$ *and* $l_{K_2,bo}(\omega_i) = 2$ *for* $9 \leq i \leq 12$. *By Def. 4, it is easy to see that* ω_9 *and* ω_{11} *are two minimal possible worlds in Table 1. So* $M(\Delta_\mu^{PLMIN,\mathbf{X}}(E)) = \{1111, 1110\}$. *That is,* $\Delta_\mu^{PLMIN,\mathbf{X}}(E) = p_1 \wedge p_2 \wedge p_3$.

The following proposition states relationships between different Δ^{PLMIN} operators when considering different ordering strategies.

Proposition 3 *Let* $E = \{K_1, ..., K_n\}$ *be a SKP, and* μ *be the integrity constraint. Let* $\mathbf{X_1} = \{X_1, ..., X_n\}$ *and* $\mathbf{X_2} = \{X_1', ..., X_n'\}$ *be two vectors of ordering strategies, where both* X_i *and* X_i' *are ordering strategies for* K_i. *Suppose* $\preceq_{X_i}$ *is more specific than* $\preceq_{X_i'}$, *for all* i, *where* $X_i \in \mathbf{X_1}$ *and* $X_i' \in \mathbf{X_2}$, *then* $\Delta_\mu^{PLMIN,\mathbf{X_2}}(E) \models \Delta_\mu^{PLMIN,\mathbf{X_1}}(E)$.

Proposition 3 shows that the operator with regard to the set of more specific ordering strategies can result in a knowledge base which has stronger inferential power. By Proposition 2 and 3, we have the following result: Suppose $X_i = bo$ and $X_i' = d$ for all i, then $\Delta_\mu^{PLMIN,\mathbf{X_2}}(E) \models \Delta_\mu^{PLMIN,\mathbf{X_1}}(E)$.

Example 2 *(continue Example 1) Suppose* $\mathbf{X}' = \{X_1', X_2', X_3'\}$, *where* $X_1' = bo$, $X_2' = X_3' = d_H$. *The computations are given in Table 2 below.*

ω	K_1	K_2	K_3	E
0111	1	5	3	(1,3,5)
0101	2	5	5	(2,5,5)
0110	1	6	3	(1,3,6)
0100	2	6	5	(2,5,6)
0011	2	4	4	(2,4,4)
0001	2	7	6	(2,6,7)
0010	2	5	4	(2,4,5)
0000	2	8	6	(2,6,8)
1111	**1**	**2**	**1**	**(1,1,2)**
1101	2	2	3	(2,2,3)
1110	1	3	1	(1,1,3)
1100	2	3	3	(2,3,3)

Table 2: $\Delta_\mu^{PLMIN,\mathbf{X}'}$ *operator*

According to Table 2, $\omega_9 = 1111$ *is the only minimal model in* $M(\mu)$. *So the result of merging by the* $\Delta^{PLMIN,d}$ *operator is* $M(\Delta_\mu^{PLMIN,\mathbf{X}'}(E)) = \{1111\}$. *So* $\Delta_\mu^{PLMIN,\mathbf{X}'}(E) = p_1 \wedge p_2 \wedge p_3 \wedge p_4$. *It is clear that* $M(\Delta_\mu^{PLMIN,\mathbf{X}'}(E)) \models M(\Delta_\mu^{PLMIN,\mathbf{X}}(E))$.

Syntactical counterpart of the Δ^{PLMIN} operators

The Δ^{PLMIN} operators are defined in a model-theoretic way in Definition 4. In this section, we propose an algorithm to compute the Δ^{PLMIN} operators syntactically.

Definition 5 *Let* ϕ *be a formula and* K *be a stratified knowledge base. Let* X *be an ordering strategy for* K. *The level of* ϕ *with regard to* K *and* X, *denoted as* $L(\phi, K_X)$ *is defined as:*

$$L(\phi, K_X) = min\{l_{K,X}(\omega_i) : \omega_i \in M(\phi)\}$$

$L(\phi, K_X)$ is the minimum priority level of models of ϕ with regard to K and X.

Given a stratified knowledge base K together with an ordering strategy X and a formula μ, a revision operator, denoted $\circ_X$ can be defined as $M(K \circ_X \mu) = Min(M(\mu), \preceq_X)$. We then have the following proposition for the $\circ_X$ revision operators.

Proposition 4 *Let* K *be a stratified knowledge base and* X *be an ordering strategy for it. Let* ϕ *be a formula. We then have,*

1. *for* X*=bo, let* $l = max\{i : \{\phi\} \cup K_{\geq i} \not\vdash \bot\}$, *then* $K \circ_{bo} \phi \equiv \phi \wedge \bigwedge_{\psi \in K_{\geq l}} \psi$.
2. *for* $X = maxsat$, *let* $l = min\{i : \{\phi\} \cup S_i \not\vdash \bot\}$, *then* $K \circ_{maxsat} \phi \equiv \phi \wedge \bigwedge_{\psi \in S_l} \psi$.
3. *for* $X = leximin$, *let* $SMC(K) = \{A = A_1 \cup ... \cup A_n \subseteq K : \forall i, \{\phi\} \cup A_1 \cup ... \cup A_i \not\vdash \bot$ *and* $\forall B_i \subseteq S_i$, *if* $A_i \subset B_i$, *then* $\{\phi\} \cup A_1 \cup ... \cup B_i \vdash \bot\}$. *Suppose* $Lex(K) = \{A = A_1 \cup ... \cup A_n \in SMC(K) : \forall B = B_1 \cup ... \cup B_n \in SMC(K), / \exists i, |B_i| > |A_i|$ *and* $\forall j < i, |B_j| = |A_j|\}$, *then* $K \circ_{leximin} \phi = \phi \wedge (\vee_{A_i \in Lex(K)} \wedge_{\psi_{ij} \in A_i} \psi_{ij})$.
4. *for* $X = d_H$, *let* $G_{S,\psi}$ *be the syntactical result of revising a knowledge base* S *using* ψ *by the Dalal revision method in (Dalal 1988). Then* $K \circ_d \phi = \phi \wedge G_{S_1,\phi} \wedge G_{S_2,\psi_1} \wedge ... \wedge G_{S_n,\psi_{n-1}}$, *where* $\psi_1 = \phi \wedge G_{S_1,\phi}$ *and* $\psi_i = \phi \wedge G_{S_1,\phi} \wedge ... \wedge G_{S_i,\psi_{i-1}}$.

By Proposition 4, the operator $\circ_{bo}$ is the cut base-revision operator defined in (Nebel 1994) and the operator $\circ_{leximin}$ is the lex-revision operator defined in (Benferhat, Dubois, and Prade 1998).

Algorithm 1

Input: a set of n stratified knowledge bases $E = \{K_1, ..., K_n\}$; a formula μ representing the integrity constraints; a set of ordering strategies $\mathbf{X} = (X_1, ..., X_n)$, where X_i is the ordering strategy for K_i.

Result: a formula $\psi_{E,\mathbf{X}}$

Let $\Phi = \{(\mu, E)\}$, $l = +\infty$, ind=1

while ($\exists\ (\phi_i, E_i) \in \Phi$, $E_i \neq \emptyset$)

  for each $(\phi_i, E_i) \in \Phi$

    for each $K_j \in E_i$, compute $L(\phi_i, (K_j)_{X_j})$

  let $l_{\phi_i} = min_{K_j \in E_i} L(\phi_i, (K_j)_{X_j})$; $l = min_{(\phi_i, E_i) \in \Phi} l_{\phi_i}$

  let $\Phi' = \{(\phi_i, E_i) \in \Phi, l_{\phi_i} \neq l\}$; $\Phi = \Phi \setminus \Phi'$

  set $\Phi' = \emptyset$

  for each $(\phi_i, E_i) \in \Phi$

    let $I_i = \{j : L(\phi_i, (K_j)_{X_j}) = l\}$

    compute $MCS(I_i) = \{J \subseteq I_i : \wedge_{j \in J} K_j \circ_{X_j} \phi_i \not\vdash \bot$ and $\forall k \in I_i \setminus J, \wedge_{j \in J} K_j \circ_{X_j} \phi_i \wedge K_k \circ_{X_k} \phi_i \vdash \bot\}$

    let $\lambda_i = max_{J' \in MCS(I)} |J'|$, where $|J|$ is the cardinality of J

    $CardM(I_i) = \{J \in MCS(I) : |J| = \lambda_i\}$

  let $\lambda = min_{(\phi_i, E_i) \in \Phi} \lambda_i$; $\Phi = \{(\phi_i, E_i) \in \Phi : \lambda_i = \lambda\}$

  for each $(\phi_i, E_i) \in \Phi$

    for each $J \in CardM(I_i)$

      let $\phi_J = \wedge_{j \in J} (K_j \circ_{X_j} \phi_i)$;

        $E_{\phi_J} = E_i \setminus \{K_j \in E_i : j \in J\}$

    $\Phi' = \Phi' \cup \{(\phi_J, E_{\phi_J}) : J \in CardM(I)\}$

  let $\Phi = \Phi'$; $\Phi' = \emptyset$

  ind=ind+1

let $\psi = \vee_{(\phi_i, E_i) \in \Phi} \phi_i$

return ψ

end

In Algorithm 1, we use Φ to denote the set of pairs consisting of a formula ϕ_i and a set E_i of knowledge bases, where ϕ_i is obtained by merging some selected knowledge bases from E and E_i contains knowledge bases which are left to be merged under the integrity constraint ϕ_i. Initially, Φ contains a single element (μ, E). In the "while" step, we check whether there is a pair (ϕ_i, E_i) in Φ such that $E_i \neq \emptyset$. If not, then the algorithm stops. Otherwise, for each element (ϕ_i, E_i) in Φ, we compute the priority level l_{ϕ_i} of ϕ_i with regard to E_i and let l be the minimal priority level among all l_{ϕ_i}. We then delete those pairs (ϕ_j, E_j) such that $l_{\phi_j} \neq l$ from Φ. For each $(\phi_i, E_i) \in \Phi$, we find all the maximal subsets of E_i which contains those stratified knowledge bases such that the levels of ϕ_i $w.r.t$ them are equal to l and whose revised formulae by ϕ_i are consistent altogether, i.e. their union is consistent. This step is a competition step. That is, the knowledge bases are defeated and will be left to be dealt with in another "while" loop when either the level of ϕ_i $w.r.t$ them are not equal to l or they are not chosen in a cardinally maximal subset. We then compare the cardinality of the maximal subsets and only keep those pairs whose maximal subsets have the maximal cardinality. After that, for each such cardinally maximal subset, we revise all the knowledge bases in it by ϕ_i. A new formula ϕ_J is then obtained by taking the conjunction of the resulting formulae of revision. A set E_{ϕ_J}, which is the complement of this cardinally maximal subset by E_i, is then attached to the new formula ϕ_J for further merging. Φ is reset to contain all those pairs of (ϕ_J, E_{ϕ_J}) and we go back to the "while" step again.

Example 3 *(Continue Example 2) Initially, we have* $\Phi = \{(\mu = \neg p_1 \vee p_2, E = \{K_1, K_2, K_3\})\}$ *and* $X_1 = bo$ *and* $X_2 = X_3 = d_H$. *We have* $L(\mu, (K_1)_{X_1}) = 1$, $L(\mu, (K_2)_{X_2}) = 2$ *and* $L(\mu, (K_3)_{X_3}) = 2$ *(which can be checked by Table 2). So* $l = 1$. *It is clear that* $\Phi' = \emptyset$. *For* $(\mu, E) \in \Phi$, *we have* $I = \{1\}$ *because only* $L(\mu, (K_1)_{X_1}) = 1$. *By Proposition 4,* $K_1 \circ_{bo} \mu = (\neg p_1 \vee p_2) \wedge (p_1 \vee p_2) \wedge p_3$. *Let* $\phi = (\neg p_1 \vee p_2) \wedge (p_1 \vee p_2) \wedge p_3$ *and* $E_\phi = \{K_2, K_3\}$. *So* $\Phi = \{(\phi, E_\phi)\}$. *We then have* $L(\phi, (K_2)_{X_2}) = 2$ *and* $L(\phi, (K_3)_{X_3}) = 2$. *So* $l = 2$. $\Phi' = \emptyset$. *For* $(\phi, E_\phi) \in \Phi$, *we have* $I = \{2, 3\}$. *By Proposition 4,* $K_2 \circ_{d_H} \phi \equiv p_1 \wedge p_2 \wedge p_3 \wedge p_4$ *and* $K_3 \circ_{d_H} \phi \equiv p_1 \wedge p_2 \wedge p_3$. *It is clear that* $K_2 \circ_{d_H} \phi$ *and* $K_3 \circ_{d_H} \phi$ *are consistent with each other, so* $MSC(I) = CardM(I) = \{J = \{2, 3\}\}$. $\phi_J = (K_2 \circ_{d_H} \phi) \wedge (K_2 \circ_{d_H} \phi) = p_1 \wedge p_2 \wedge p_3 \wedge p_4$ *and* $E_{\phi_J} = \emptyset$. *We have* $\Phi = \{(\phi_J, \emptyset)\}$. *The algorithm terminates. So* $\psi_{E,\mathbf{X}} = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, *which is the same as* $\triangle_\mu^{PLMIN,\mathbf{X}}(E)$ *in Example 2.*

Proposition 5 *Let* $E = \{K_1, ..., K_n\}$ *be a set of* n *stratified knowledge bases and* $\mathbf{X} = \{X_1, ..., X_n\}$ *be a set of ordering strategies, where* X_i *is the ordering strategy for* K_i. μ *is the integrity constraint. Suppose* $\Delta_\mu^{PLMIN,\mathbf{X}}(E)$ *is the knowledge base obtained by* K_i *under constraints* μ *using the* $\Delta_\mu^{PLMIN,\mathbf{X}}$ *operator and* $\psi_{E,\mathbf{X}}$ *is the knowledge base obtained by Algorithm 1, then* $\Delta_\mu^{PLMIN,\mathbf{X}}(E) \equiv \psi_{E,\mathbf{X}}$.

Proposition 5 tells us that the resulting knowledge base of Algorithm 1 is equivalent to that of the $\Delta_\mu^{PLMIN,\mathbf{X}}$ operator. Therefore, the syntactical merging methods obtained by Algorithm 1 are the syntactical counterparts of our merging operators.

Flat case

In this section, we apply our merging operators to the classical knowledge bases. Since our merging operators are based on the ordering strategies, we need to consider the ordering strategies for classical knowledge bases.

Proposition 6 *Let* K *be a classical knowledge base. Suppose* X *is an ordering strategy, then*

1. *for* $X = bo$ *and* $X = maxsat$, *we have* $\omega \preceq_X \omega'$ *iff* $\omega \models K$
2. *for* $X = leximin$, *let* $K(\omega) = \{\phi \in K : \omega \models \phi\}$, *we have* $\omega \preceq_X \omega'$ *iff* $|K(\omega)| \geq |K(\omega')|$
3. *for* $X = d$, *we have* $\omega \preceq_X \omega'$ *iff* $d(\omega, K) \leq d(\omega, K')$.

By Proposition 6, the best out ordering and the maxsat ordering are reduced to the same ordering when knowledge base is classical. Furthermore, the leximin ordering can be used to order possible worlds when the knowledge base is inconsistent.

Proposition 7 *Let E be a knowledge profile and μ be a formula. Let $MAXCONS(E,\mu) = \{F \subseteq E : \bigcup(F) \cup \{\mu\} \not\models \bot,$ and if $F \subset E' \subseteq E,$ then $\bigcup(E') \cup \{\mu\} \models \bot\}$. That is, $MAXCONS(E,\mu)$ is the set of maximal subsets of E which are consistent with μ. Let $CardM(E,\mu) = \{F \in MAXCONS(E,\mu) : \nexists F' \in MAXCONS(E,\mu), |F| < |F'|\}$. Suppose $X_i = bo$ or $maxsat$ for all i, then $\Delta_\mu^{PLMIN,\mathbf{X}}(E) = \bigvee_{F \in CardM(E,\mu)}(\wedge_{\phi \in F}\phi \wedge \mu)$.*

Proposition 7 shows that the $\Delta^{PLMIN,\mathbf{X}}$ operator is equivalent to the $\triangle^{C4}$ operator defined in (Konieczny, Lang, and Marquis 2004), which selects the set of consistent subsets of $E \cup \{\mu\}$ that contain the constraints μ and that are maximal with respect to cardinality, when each knowledge base is viewed as a formula and ordering strategy of it is the *best out* strategy or *maxsat* strategy.

When $X_i = d$ for all i, the corresponding $\Delta_\mu^{PLMIN,\mathbf{X}}$ operators are similar to the $\triangle_\mu^{d,Gmin}$ operators defined as follows.

Definition 6 *(Everaere, Konieczny, and Marquis 2005) Let d be a pseudo-distance, μ an integrity constraint, $E = \{K_1, ..., K_n\}$ a profile and let ω be an interpretation. The "distance" between ω and E, denoted by $d_{d,Gmin}(\omega, E)$, is defined as the list of numbers $(d_1, ..., d_n)$ obtained by sorting in increasing order the set $\{d(\omega, K_i) : K_i \in E\}$. The models of $\triangle_\mu^{d,Gmin}(E)$ are the models of μ that are minimal $w.r.t$ the lexicographical order induced by the natural order.*

Our Δ_μ^{PLMIN} operators and the $\triangle_\mu^{d,Gmin}$ operators differ in that the lists of numbers attached to models are different. The former uses the priority levels of a model $w.r.t$ all the knowledge bases and the latter uses the distance between a model and each knowledge base.

Proposition 8 *Let $E = \{K_1, ..., K_n\}$ a profile and μ an integrity constraint. d_D is the drastic distance and $\mathbf{X} = (X_1, ..., X_n)$ is a set of ordering strategies attached to K_i ($i = 1, ..., n$), where $X_i = d_D$ for all i. Then $\Delta_\mu^{PLMIN,\mathbf{X}}(E) \equiv \triangle_\mu^{d_D,Gmin}(E)$.*

Proposition 8 shows that the $\Delta_\mu^{PLMIN,\mathbf{X}}$ operator and the $\triangle_\mu^{d_D,Gmin}$ operator are equivalent when the *drastic distance* is chosen.

Propositions 7 and 8 only consider $\triangle_\mu^{d,Gmin}$ operators where all knowledge bases have the same ordering strategy. When hybrid ordering strategies are used, we can get more operators. For example, if we use the $leximin$ ordering for those knowledge bases which are inconsistent, then our operators can be applied to merging a set of knowledge bases which may be individually inconsistent. Now let us look at an example.

Example 4 *Let $E = \{K_1, K_2\}$, where $K_1 = \{p_1 \vee p_2, p_3, \neg p_3\}$ and $K_2 = \{p_1, p_2, p_3\}$, and $\mu = \{(p_1 \vee p_3) \wedge p_2\}$. So $Mod(\mu) = \{\omega_1 = 110, \omega_2 = 111, \omega_3 = 011\}$. Let $\mathbf{X} = (X_1, X_2)$, where $X_1 = leximin$ and $X_2 = bo$ are ordering strategies of K_1 and K_2 respectively. The computations are given in Table 3 below.*

ω	K_1	K_2	E
110	1	2	(1,2)
111	**1**	**1**	**(1,1)**
011	1	2	(1,2)

Table 3: $\Delta^{PLMIN,\mathbf{X}}$ operator

According to Table 3, $\omega_2 = 111$ is the only minimal model in $M(\mu)$. So $M(\Delta_\mu^{PLMIN,\mathbf{X}}(E)) = \{111\}$. That is, $\Delta_\mu^{PLMIN,\mathbf{X}}(E) = p_1 \wedge p_2 \wedge p_3$.

Computational Complexity

We now discuss the complexity issue. First we need to consider the computational complexity of stratifying Ω from a stratified knowledge base. In (Lang 2004), two important problems for logical preference representation languages were considered. We express them as follows.

Definition 7 *Given a stratified knowledge base K and two interpretations ω and ω', the COMPARISON problem consists of determining whether $\omega \preceq_X \omega'$, where X denotes an ordering strategy. The NON-DOMINANCE problem consists of determining whether ω is* non-dominated *for $\preceq_X$, that is, there is not ω' such that $\omega' \prec_X \omega$.*

It was shown in (Lang 2004) that the NON-DOMINANCE problem is usually a hard problem, i.e **coNP**-complete. We have the following proposition on NON-DOMINANCE problem for ordering strategies in Section 3.

Proposition 9 *Let K be a stratified knowledge base. For $X = bo, maxsat,$ or $leximin$:*

(1) COMPARISON is in P*, where* P *denotes the class of problems decidable in deterministic polynomial time.*
*(2) NON-DOMINANCE is co***NP***-complete.*

To stratify Ω, we need to consider the problem *determining all non-dominated interpretations*, which is computational much harder than the NON-DOMINANCE problem (we believe it is Σ_2^p-hard). To simplify the computation of our merging operators, we assume that Ω is stratified from each stratified knowledge base during an off-line preprocessing stage.

Let Δ be a merging operator. The following decision problem is denoted as MERGE(Δ):

- **Input :** a 4-tuple $\langle E, \mu, \psi, \mathbf{X} \rangle$ where $E = \{K_1, ..., K_n\}$ is a multi-set of stratified knowledge bases, μ is a formula, and ψ is a formula; $\mathbf{X} = (X_1, ..., X_n)$, where X_i is the ordering strategy attached to K_i. $\Omega/\langle K_i, X_i \rangle = (\Omega_{i1}, ..., \Omega_{in_i})$ ($i = 1, ..., n$), where Ω_{ij} is the non-empty set which contains all the minimal elements of $\Omega \setminus (\cup_{l=1}^{j_1} \Omega_{il})$ with regard to an ordering strategy X_i of K_i.
- **Question :** Does $\Delta_\mu(E) \models \psi$ hold?

Proposition 10 *MERGE($\Delta^{PLMIN,\mathbf{X}}$) in Θ_2^p, where Θ_2^p is the class of all languages that can be recognized in polynomial time by a deterministic Turing machine using a number of calls to an* NP *oracle bounded by a logarithmic function of the size of the input data. Let $\mathbf{X} = (X_1, ..., X_n)$,*

where $X_i = bo$, $maxsat$, $leximin$, or $d_D (i = 1, ..., n)$, then MERGE($\Delta^{PLMIN,\mathbf{X}}$) is Θ_2^p-complete.

Proposition 10 shows that the computational complexity of inference for our merging operators is located at a low level of the boolean hierarchy under an additional assumption.

Logical Properties

Many logical properties have been proposed to characterize a belief merging operator. We introduce the set of postulates proposed in (Konieczny and Pino Pérez 2002), which is used to characterize Integrity Constraints (*IC*) merging operators.

Definition 8 *Let E, E_1, E_2 be knowledge profiles, K_1, K_2 be consistent knowledge bases, and μ, μ_1, μ_2 be formulas from $\mathcal{L}_{PS}$. Δ is an IC merging operator iff it satisfies the following postulates:*
(IC0) $\Delta_\mu(E) \models \mu$
(IC1) *If μ is consistent, then $\Delta_\mu(E)$ is consistent*
(IC2) *If $\bigwedge E$ is consistent with μ, then $\Delta_\mu(E) \equiv \bigwedge E \wedge \mu$, where $\bigwedge(E) = \wedge_{K_i \in E} K_i$*
(IC3) *If $E_1 \equiv E_2$ and $\mu_1 \equiv \mu_2$, then $\Delta_{\mu_1}(E_1) \equiv \Delta_{\mu_2}(E_2)$*
(IC4) *If $K_1 \models \mu$ and $K_2 \models \mu$, then $\Delta_\mu(\{K_1, K_2\}) \wedge K_1$ is consistent iff $\Delta_\mu(\{K_1, K_2\}) \wedge K_2$ is consistent*
(IC5) $\Delta_\mu(E_1) \wedge \Delta_\mu(E_2) \models \Delta_\mu(E_1 \sqcup E_2)$
(IC6) *If $\Delta_\mu(E_1) \wedge \Delta_\mu(E_2)$ is consistent, then $\Delta_\mu(E_1 \sqcup E_2) \models \Delta_\mu(E_1) \wedge \Delta_\mu(E_2)$*
(IC7) $\Delta_{\mu_1}(E) \wedge \mu_2 \models \Delta_{\mu_1 \wedge \mu_2}(E)$
(IC8) *If $\Delta_{\mu_1}(E) \wedge \mu_2$ is consistent, then $\Delta_{\mu_1 \wedge \mu_2}(E) \models \Delta_{\mu_1}(E) \wedge \mu_2$*

The postulates are used to characterize an IC merging operator in classical logic. Detailed explanation of the above postulates can be found in (Konieczny and Pino Pérez 2002).

Some postulates in Definition 8 need to be modified if we consider merging postulates for stratified knowledge bases, i.e., ($IC2$), ($IC3$) should be modified as:
(IC2$'$) Let $\bigwedge E = \wedge_{K_i \in E} \wedge_{\phi_{ij} \in K_i} \phi_{ij}$. If $\bigwedge E$ is consistent with μ, then $\Delta_\mu(E) \equiv \bigwedge E \wedge \mu$
(IC3$'$) If $E_1 \equiv_s E_2$ and $\mu_1 \equiv \mu_2$, then $\Delta_{\mu_1}(E_1) \equiv \Delta_{\mu_2}(E_2)$

($IC3'$) is stronger than ($IC3$) because the condition of equivalence between two knowledge profiles is generalized to the condition of equivalence between two SKPs. We do not generalize ($IC4$), the fairness postulate, which says that the result of merging of two belief bases should not give preference to one of them. This postulate is controversial (Konieczny 2004). And it is hard to be adapted in the prioritized case because a stratified knowledge base may be inconsistent and there is no unique consequence relation for a stratified knowledge base (Benferhat et al. 1993).

Proposition 11 *$\Delta_\mu^{PLMIN,\mathbf{X}}$ satisfies (IC0), (IC1), ($IC2'$), (IC5), (IC6) (IC7), (IC8). The other postulates are not satisfied in the general case.*

($IC3'$) is not satisfied because some ordering strategies are syntax-sensitive. However, when the ordering strategies are either best-out ordering or maxsat ordering, then our merging operators satisfy all the generalized postulates.

Proposition 12 *Suppose $X_i = bo$ or $maxsat$, then $\Delta_\mu^{PLMIN,\mathbf{X}}$ satisfies (IC0), (IC1), ($IC2'$), ($IC3'$), (IC5), (IC6), (IC7), (IC8). The other postulates are not satisfied in the general case.*

Related Work

Merging of stratified knowledge bases is often handled in the framework of possibilistic logic (Dubois, Lang, and Prade 1994) or ordinal conditional function (Spohn 1988). In possibilistic logic, the merging problems are often solved by aggregating *possibility distributions*, which are mappings from Ω to a common scale such as [0,1], using some *combination modes*. Then the syntactic counterpart of these combination modes can be defined accordingly (Benferhat, Dubois, and Prade 1997; Benferhat et al. 2002). In (Chopra, Ghose, and Meyer 2005; Meyer, Ghose, and Chopra 2002), the merging is conducted by merging *epistemic states* which are (total) functions from the set of interpretations to **N**, the set of natural numbers. There are many other merging methods in possibilistic logic (Benferhat, Dubois, and Prade 1998; Benferhat et al. 1999; Qi, Liu, and Glass, 2004a; Qi, Liu, and Glass 2004b) and in ordinal conditional function framework (Benferhat et al. 2004; Qi, Liu, and Bell 2005). Our merging operators differs from previous ones at least in two aspects:

First, our operators are semantically defined in a model-theoretic way and others are semantically defined by distribution functions such as possibility distributions. In the flat case, our merging operators belong to model-based merging operators, and they capture some notion of minimal change. Whilst other merging operators are usually syntax-based ones in the flat case.

Second, most of previous merging operators are based on the commensurability assumption. In (Benferhat et al. 1999), a merging approach for stratified knowledge base is proposed which drops the commensurability assumption. However, their approach is based on the assumption that there is an ordering relation between two stratified knowledge bases K_1 and K_2, i.e. K_1 has priority over K_2. In contrast, our merging operators do not require any of above assumptions and are flexible enough to merge knowledge bases which are stratified by a total pre-ordering on their elements. So our merging operators are more general and practical than other methods.

This work is also related to the logical preference description language (LPD) in (Brewka 2004). The language LPD uses binary operators $\vee$, $\wedge$ and $>$ to connect two (or more) *basic orderings* and get more complex orderings. In contrast, when defining our merging operators, we use an adaptive method which is based on a lexicographical preference to combine orderings assigned to original knowledge bases.

Conclusions and Further Work

In this paper, we proposed a family of model-theoretic operators to merge stratified knowledge bases with integrity constraints. We also considered the syntactical counterpart of merging operators. Our operators can be applied to classical knowledge bases. In that case, some of our operators are

reduced to existing merging operators. The computational complexity of our merging operators was analyzed. Under an additional assumption, the computation of Δ^{PLMIN} is equivalent to that of Δ^{GMIN} in (Everaere, Konieczny, and Marquis 2005). Finally, we revised the set of postulates defined in (Konieczny and Pino Pérez 2002) and shown that our operators satisfy most of the revised postulates.

There are several problems that will be left as further work. First, we have applied our merging operators to classical bases and got some interesting results. By Proposition 11 and Proposition 12, it is easy to conclude that our operators have good logical properties in flat cases. However, to have a thorough evaluation of our operators, we need to consider other important criteria to compare operators, such as the strategy-proofness and discriminating power. Second, we revised the set of postulates defined in (Konieczny and Pino Pérez 2002). However, the revision is a simple extension of existing postulates. Due to the additional information of stratified knowledge bases, the postulates of a "rational" merging operators for stratified knowledge bases should be much more complex than what we have considered in this paper. More postulates will be explored in the future.

References

Abidi, M.A., and Gonzalez, R.C. eds. 1992. *Data Fusion in Robotics and Machine Intelligence.* Academic Press.

Baral, C.; Kraus, S. and Minker, J. 1991. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208-220.

Baral, C.; Kraus, S.; Minker, J.; and Subrahmanian, V.S. 1992. Combining knowledge bases consisting in first order theories. *Computational Intelligence* 8(1):45-71.

Benferhat, S.; Cayrol, C.; Dubois, D.; Lang, L. and Prade, H. 1993a. Inconsistency management and prioritized syntax-based entailment. In*Proc. IJCAI'93*, 640-645.

Benferhat, S.; Dubois, D.; and Prade, H. 1997. From semantic to syntactic approaches to information combination in possibilistic logic. In Bouchon-Meunier, B. eds., *Aggregation and Fusion of Imperfect Information*, 141-151. Physica. Verlag.

Benferhat, S.; Dubois, D. and Prade, H.: Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study. Part 2: The prioritized case. In *Logic at work : essays dedicated to the memory of Helena Rasiowa* / Ewa Orow. - New York : Physica-Verlag, pp. 473-511, 1998.

Benferhat S.; Dubois, D.; Prade, H. and Williams, M.A. 1999. A Practical Approach to Fusing Prioritized Knowledge Bases, *Proc. 9th Portu. Conf. Artificial Intelligence*, pp. 223-236, 1999.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2002. Possibilistic merging and distance-based fusion of propositional information. *Annals of Mathematics and Artificial Intelligence* 34:217-252.

Benferhat, S. and Baida, R.E. 2004. A stratified first order logic approach for access control. *International Journal of Intelligent Systems*, 19:817-836.

Benferhat S., Kaci S., Berre D.L., and Williams M.A. Weakening conflicting information for iterated revision and knowledge integration. *Artificail Intelligence*, vol. 153(1-2), pp. 339-371, 2004.

Brewka, G. 2004. A rank-based description language for qualitative preferences. In *Proc. of ECAI'04*, 303-307.

Bloch, I. and Hunter, A. 2001. Fusion: General concepts and characteristics. *International Journal of Intelligent Systems*, 16(10):1107-1134 (special issue on Data and Knowledge Fusion).

Brewka, G. 1989. Prefered subtheories-an extended logical framework for default reasoning. In *Proc. of IJCAI'89*, 1043-1048.

Brewka, G. 2004. A rank-based description language for qualitative preferences. In *Proceedings of sixteenth European Conference on Artificial Intelligence (ECAI'04)*, 303-307.

Cholvy, L. 1992. A logical approach to multi-sources reasoning. In *Proceedings of International Conference Logic at Work on Knowledge Representation and Reasoning Under Uncertainty, Logic at Work*, 183-196. Springer-Verlag.

Chopra, S.; Ghose, A. and Meyer, T. 2003. Non-prioritized ranked belief change. *Journal of Philosophical Logic*. 32(4):417-443.

Chopra, S.; Ghose, S. and Meyer, T. 2005. Social choice theory, belief merging, and strategy-proofness. *Journal of Information Fusion*, to appear.

Coste-Marquis, S. and Marquis, P. 2000. Compiling stratified belief bases. In *Proc. of ECAI'00*, 23-27.

Coste-Marquis, S.; Lang, J.; Liberatore, P. and Marquis, P. Expressive power and succintness of propositional languages for preference representation. In *Proceedings of Ninth International Conference on Knowledge Representation and Reasoning (KR'04)*, 203-213.

Dalal, M. 1988. Investigations into a theory of knowledge base revision: Preliminary report, *Proc. of AAAI'88*, 3-7.

Dubois, D.; Lang, J.; and Prade, H. 1992. Dealing with Multi-Source Information in Possibilistic Logic. In *Proceedings of 10th European Conference on Artificial Intelligence(ECAI 92)*, 38-42.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic. In *Handbook of logic in Aritificial Intelligence and Logic Programming*, Volume 3. Oxford University Press, 439-513.

Everaere, P.; Konieczny, S. and Marquis, P. 2005. Quota and Gmin merging operators. In *IJCAI'05*, 424-429.

Fagin, R. and Ullman, J.D. 1983. On the semantics of updates in Databases. In *Proceedings of Second ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, Atlanta, 352-265.

Gärdenfors P. 1988. *Knowledge in Flux-Modeling the Dynamic of Epistemic States.* Mass.: MIT Press.

Konieczny, S. and Pino Pérez, R. 1998. On the logic of merging. In *Proceedings of the Sixth International Confer-*

ence on Principles of Knowledge Representation and Reasoning (KR'98), 488-498. Morgan Kaufmann.

Konieczny, S. 2000. On the difference between merging knowledge bases and combining them. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 135-144.

Konieczny, S. and Pino Pérez, R. 2002. Merging information under constraints: a qualitative framework. *Journal of Logic and Computation* 12(5):773-808.

Konieczny, S.; Lang, J. and Marquis, P. 2004. DA^2 operators. *Artificial Intelligence*, 157(1-2):49-79.

Konieczny, S. Propositional belief merging and belief negotiation model, In *NMR'04*, 249-257, 2004.

Liberatore, P. and Schaerf, M. 1998. Arbitration (or How to Merge Knowledge Bases). *IEEE Transaction on Knowledge and Data Engineering* 10(1):76-90.

Lafage, L. and Lang, J. 2000. Logical representation of preference for group decision making. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 457-468. Morgan Kaufmann.

Lang, J. 2004. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 4(1-3):37-71.

Meyer, T.; Ghose A. and Chopra S. 2002. Syntactic representations of semnatic merging operations. In *Proceedings of sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI'02)*, 620.

Nebel, B. 1994. Belief Revision operators and schemes: Semantics representation and complexity. In *Proceedings of eleventh European Conference on Artificial Intelligence*, 341-345.

Nebel, B. 1998. How hard is it to revise a belief base? In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 3: Belief change, Kluwer Academic, Dubois and Prade (eds.), 77-145.

Pearl, J. 1990. System Z: A natural ordering of defaults with tractable applications to default reasoning. In *Proc. of third International Conference on Theoretical Aspects of Reasoning about Knowledge*, 121-135.

Qi, G.; Liu, W. and Glass, D. 2004. A split-combination methods for merging possibilistic knowledge bases. In *Proceedings of the Ninth International Conference on Principles of Knoweldge Presentation and Reasoning (KR'04)*, 348-356. Morgan Kaufmann.

Qi, G.; Liu, W. and Glass, D. 2004. Combining individually inconsistent prioritized knowledge bases. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR'04)*, 342-349. Canada.

Qi, G.: Liu, W. and Bell, D.A. 2005. A revision-based approach to resolving conflicting information. In *Proceedings of twenty-first Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 477-484.

Revesz, P.Z. 1997. On the semantics of arbitration. *International Journal of Algebra and Computation*, 7(2):133-160.

Spohn, W. 1988. Ordinal conditional functions. In William L. Harper and Brian Skyrms (eds.), *Causation in Decision, Belief Change, and Statistics*, 11, 105-134. Kluwer Academic Publisher.

2.4 Merging Optimistic and Pessimistic Preferences

Merging Optimistic and Pessimistic Preferences

Souhila Kaci
CRIL
Rue de l'Université SP 16
62307 Lens Cedex, France
kaci@cril.univ-artois.fr

Leendert van der Torre
ILIAS
University of Luxembourg
Luxembourg
leon.vandertorre@uni.lu

Abstract - ***In this paper we consider the extension of non-monotonic preference logic with the distinction between controllable (or endogenous) and uncontrollable (or exogenous) variables, which can be used for example in agent decision making and deliberation. We assume that the agent is optimistic about its own controllables and pessimistic about its uncontrollables, and we study ways to merge these two distinct dimensions. We also consider complex preferences, such as optimistic preferences conditional on an uncontrollable, or optimistic preferences conditional on a pessimistic preference.***

Keywords: Preference logic, preference merging, non-monotonic reasoning.

Introduction

In many areas such as cooperative information systems, multi-databases, multi-agents systems, information comes from multiple sources. The multiplicity of sources providing information makes that information is often contradictory which requires conflict resolution. This problem has been widely studied in literature where implicit priorities, based on Dalal's distance, (Lin 1996; Lin & Mendelzon 1998; Konieczny & Pérez 1998; Revesz 1993; 1997) or explicit priorities (Benferhat *et al.* 1999; 2002) are used in order to solve conflicts.

Our concern in this paper is the merging of preferences of a single agent when they are expressed in a logic of preferences. Logics of preferences attract much attention in knowledge representation and reasoning, where they are used for a variety of applications such as qualitative decision making (Doyle & Thomason 1999). In this paper we oppose to the common wisdom that the very efficient specificity algorithms used in some non-monotonic preference logics are too simple to be used for knowledge representation and reasoning applications. In that logics we distinguish minimal and maximal specificity principles which correspond to a gravitation towards the ideal and the worst respectively. We counter the argument that a user is forced to chose among minimal and maximal specificity by introducing the fundamental distinction between controllable and uncontrollable variables from decision and control theory, and merging preferences on the two kinds of variables as visualized in Figure 1. Our work is based on the hypothesis that each set of preferences on controllable and uncontrollable variables is consistent. The merging process aims to cohabit controllable and uncontrollable variables in an intuitive way. Preferences on controllable variables are called optimistic preferences since minimal specificity principle is used for such variables. This principle is a gravitation towards the ideal and thus corresponds to an optimistic reasoning. Preferences on uncontrollable variables are called pessimistic preferences since maximal specificity principle is used for such variables. This principle is a gravitation towards the worst and thus corresponds to an pessimistic reasoning.

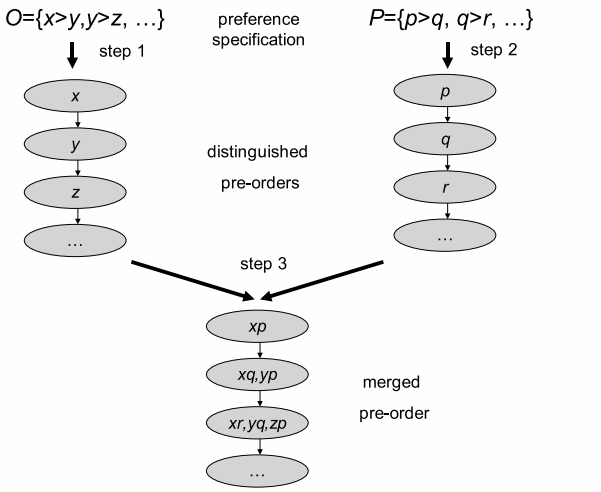


Figure 1: Merging optimistic and pessimistic preferences.

A preference specification contains optimistic preferences (O) defined on controllables $x, y, z, \ldots$, and pessimistic preferences (P) defined on uncontrollables $q, r, t, \ldots$, which are interpreted as constraints on total preorders on worlds. The efficient specificity algorithms (step 1 and 2 in Figure 1) calculate unique distinguished total preorders, which are thereafter merged (step 3) by symmetric or a-symmetric mergers. If the optimistic and pessimistic preferences in Figure 1 are defined on separate languages, then for step 1 and 2 we can use existing methods in preference

logic, such as (Kaci & van der Torre 2005a). In this paper we also consider more general languages, in which preferences on controllables are conditional on uncontrollables, or on preferences on uncontrollables (or vice versa).

The remainder of this paper is organized as follows. After a necessary background, we present a logic of optimistic preferences defined on controllable variables and a logic of pessimistic preferences defined on uncontrollable variables. Then we propose some merging approaches of optimistic and pessimistic preferences. We also introduce a logic of preferences where pessimistic and optimistic preferences are merged in the logic itself. Lastly we conclude with future research.

Background

Let W be the set of propositional interpretations of $\mathcal{L}$, and let $\succeq$ be a total pre-order on W (called also a preference order), i.e., a reflexive, transitive and connected ($\forall \omega, \omega' \in W$ we have either $\omega \succeq \omega'$ or $\omega' \succeq \omega$) relation. We write $w \succ w'$ for $w \succeq w'$ without $w' \succeq w$. Moreover, we write $\max(x, \succeq)$ for $\{w \in W \mid w \models x, \forall w' \in W : w' \models x \Rightarrow w \succeq w'\}$, and analogously we write $\min(x, \succeq)$ for $\{w \in W \mid w \models x, \forall w' \in W : w' \models x \Rightarrow w' \succeq w\}$.

The following definition illustrates how a preference order can also be represented by a well ordered partition of W. This is an equivalent representation, in the sense that each preference order corresponds to one ordered partition and vice versa. This equivalent representation as an ordered partition makes the definition of the non-monotonic semantics, defined later in the paper, easier to read.

Definition 1 (Ordered partition) *A sequence of sets of worlds of the form $(E_1, \ldots, E_n)$ is an ordered partition of W iff*

- *$\forall i$, E_i is nonempty,*
- *$E_1 \cup \cdots \cup E_n = W$ and*
- *$\forall i, j$, $E_i \cap E_j = \emptyset$ for $i \neq j$.*

An ordered partition of W is associated with pre-order $\succeq$ on W iff $\forall \omega, \omega' \in W$ with $\omega \in E_i, \omega' \in E_j$ we have $i \leq j$ iff $\omega \succeq \omega'$.

Preferences for controllables

Reasoning about controllables is optimistic in the sense that an agent or decision maker can decide the truth value of a controllable proposition, and thus may expect that the best state will be realized.

Optimistic reasoning semantics

A preference statement is a comparative statement "x is preferred to y", with x and y propositional sentences of a propositional language on a set of controllable propositional atoms. A reasoning about a preference can be optimistic or pessimistic with respect to both its left hand side and right hand side, indicated by o and p respectively. Formally we write $x \ {}^{a}\!>^{b} y$, where $a, b \in \{o, p\}$. An optimistic reasoning focuses on the best worlds while a pessimistic reasoning focuses on the worst worlds. For example, the preference $x \ {}^{p}\!>^{o} y$ indicates that we are drawing a pessimistic reasoning with respect to x, and an optimistic reasoning with respect to y. This means that we deal with the worst x-worlds i.e. $\min(x, \succeq)$ and the best y-worlds i.e. $\max(y, \succeq)$.
An optimistic reasoning on a preference statement over controllable variables consists of an optimistic reasoning w.r.t. its right and left hand side. This also includes the case where the reasoning is pessimistic w.r.t. its left hand side and optimistic w.r.t. its right hand side. This will be explained later in this subsection. For the sake of simplicity, such a preference is called *optimistic*. Indeed we define an *optimistic* preference specification as a set of strict and non-strict optimistic preferences:

Definition 2 (Optimistic preference specification) *Let $\mathcal{L}_\mathcal{C}$ be a propositional language on a set of controllable propositional atoms $\mathcal{C}$. Let $\mathcal{O}_\rhd$ be a set of optimistic preferences of the form $\{x_i \rhd y_i \mid i = 1, \cdots, n, x_i, y_i \in \mathcal{L}_\mathcal{C}\}$. A preference specification is a tuple $\langle \mathcal{O}_\rhd \mid \rhd \in \{ \ {}^{p}\!>^{o},\ {}^{p}\!\geq^{o}\ {}^{o}\!>^{o},\ {}^{o}\!\geq^{o}\}\rangle$.*

We define preferences of x over y as preferences of $x \wedge \neg y$ over $y \wedge \neg x$. This is standard and known as von Wright's expansion principle (Wright 1963). Additional clauses may be added for the cases in which sets of worlds are nonempty, to prevent the satisfiability of preferences like $x > \top$ and $x > \bot$. To keep the formal exposition to a minimum, we do not consider this borderline condition in this paper.

Definition 3 (Monotonic semantics) *Let $\succeq$ be a total pre-order on W.*

- $\succeq \models x \ {}^{o}\!>^{o} y$ *iff* $\forall w \in \max(x \wedge \neg y, \succeq)$ *and* $\forall w' \in \max(\neg x \wedge y, \succeq)$ *we have* $w \succ w'$
- $\succeq \models x \ {}^{o}\!\geq^{o} y$ *iff* $\forall w \in \max(x \wedge \neg y, \succeq)$ *and* $\forall w' \in \max(\neg x \wedge y, \succeq)$ *we have* $w \succeq w'$
- $\succeq \models x \ {}^{p}\!>^{o} y$ *iff* $\forall w \in \min(x \wedge \neg y, \succeq)$ *and* $\forall w' \in \max(\neg x \wedge y, \succeq)$ *we have* $w \succ w'$
- $\succeq \models x \ {}^{p}\!\geq^{o} y$ *iff* $\forall w \in \min(x \wedge \neg y, \succeq)$ *and* $\forall w' \in \max(\neg x \wedge y, \succeq)$ *we have* $w \succeq w'$.

A total pre-order $\succeq$ is a model of an optimistic preference specification $\mathcal{O}_\rhd$ if it is a model of each $p_i \rhd q_i \in \mathcal{O}_\rhd$.

Note that $x \ {}^{p}\!>^{o} y$ means that each x-world is preferred to all y-worlds w.r.t. $\succeq$. This preference can be equivalently written as a set of optimistic preferences of the form $\{x' \ {}^{o}\!>^{o} y : x' \text{ is a } x - \text{world}\}$. This is also true for $x \ {}^{p}\!\geq^{o} y$ preferences.

Example 1 *Consider an agent organizing his evening by deciding whether he goes to the cinema (c), with his friend (f) and whether he alos goes to the restaurant (r). We have $\mathcal{O} = \langle \mathcal{O}\ {}_{o}\!>^{o}, \mathcal{O}\ {}_{p}\!>^{o}, \mathcal{O}\ {}_{p}\!\geq^{o}\rangle$, where $\mathcal{O}\ {}_{o}\!>^{o} = \{c \wedge f \ {}^{o}\!>^{o} \neg(c \wedge f)\}$, $\mathcal{O}\ {}_{p}\!>^{o} = \{c \wedge r \ {}^{p}\!>^{o} c \wedge \neg r\}$, $\mathcal{O}\ {}_{p}\!\geq^{o} = \{c \wedge r \ {}^{p}\!\geq^{o} \neg c \wedge r\}$. The strict preference $c \wedge f \ {}^{o}\!>^{o} \neg(c \wedge f)$ means that there is at least a situation in which the agent goes to the cinema with his friend which is strictly preferred to all situations where the agent does not go to the cinema with his friend. The strict preference $c \wedge r \ {}^{p}\!>^{o} c \wedge \neg r$ means that each situation in which the agent goes to the cinema and the restaurant is strictly preferred to all situations in which the agent goes to the cinema but not to the restaurant. Finally the non-strict preference $c \wedge r \ {}^{p}\!\geq^{o} \neg c \wedge r$ means that*

each situation in which the agent goes to the cinema and the restaurant is at least as preferred as all situations in which the agent goes to the restaurant but not to the cinema.

We compare total pre-orders based on the so-called specificity principle. Optimistic reasoning is based on the minimal specificity principle, which assumes that worlds are as good as possible.

Definition 4 (Minimal specificity principle) *Let $\succeq$ and $\succeq'$ be two total pre-orders on a set of worlds W represented by ordered partitions $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_m)$ respectively. We say that $\succeq$ is at least as specific as $\succeq'$, written as $\succeq \sqsubseteq \succeq'$, iff $\forall \omega \in W$, if $\omega \in E_i$ and $\omega \in E'_j$ then $i \leq j$. $\succeq$ belongs to the set of the least specific pre-orders among a set of pre-orders $\mathcal{O}$ if there is no $\succeq' \in \mathcal{O}$ s.t. $\succeq' \sqsubset \succeq$, i.e., $\succeq' \sqsubseteq \succeq$ holds but $\succeq \sqsubseteq \succeq'$ does not.*

Algorithm 1 gives the (unique) least specific pre-order satisfying an optimistic preference specification. All the proofs can be found in (Kaci & van der Torre 2006).

Following Definition 2 an optimistic preference specification contains the following sets of preferences:

$$\mathcal{O}_{{}^{o}>^{o}} = \{C_{i_1} : x_{i_1} \, {}^{o}>^{o} y_{i_1}\},$$
$$\mathcal{O}_{{}^{o}\geq^{o}} = \{C_{i_2} : x_{i_2} \, {}^{o}\geq^{o} y_{i_2}\},$$
$$\mathcal{O}_{{}^{p}>^{o}} = \{C_{i_3} : x_{i_3} \, {}^{p}>^{o} y_{i_3}\},$$
$$\mathcal{O}_{{}^{p}\geq^{o}} = \{C_{i_4} : x_{i_4} \, {}^{p}\geq^{o} y_{i_4}\}.$$

Moreover, we refer to the constraints of these preferences by

$$\overline{\mathcal{C}} = \bigcup_{k=1,\cdots,4} \{\overline{\mathcal{C}}_{i_k} = (L(C_{i_k}), R(C_{i_k}))\},$$

where the left and right hand side of these constraints are $L(C_{i_k}) = |x_{i_k} \wedge \neg y_{i_k}|$ and $R(C_{i_k}) = |\neg x_{i_k} \wedge y_{i_k}|$ respectively; $|\phi|$ denotes the set of interpretations satisfying ϕ.

The basic idea of the algorithm is to construct the least specific pre-order by calculating the sets of worlds of the ordered partition, going from the ideal to the worst worlds.

At each step of the algorithm, we look for worlds which can have the current highest ranking in the preference order. This corresponds to the current minimal value l. These worlds are those which do not falsify any constraint in $\overline{\mathcal{C}}$. We first put in E_l worlds which do not falsify any strict preference. These worlds are those which do not appear in the right hand side of the strict preferences $\overline{\mathcal{C}}_{i_1}$ and $\overline{\mathcal{C}}_{i_3}$. Now we remove from E_l worlds which falsify constraints of the non-strict preferences $\overline{\mathcal{C}}_{i_2}$ and $\overline{\mathcal{C}}_{i_4}$. Constraints $\overline{\mathcal{C}}_{i_2}$ are violated if $L(C_{i_2}) \cap E_l = \emptyset$ and $R(C_{i_2}) \cap E_l \neq \emptyset$, while the constraints $\overline{\mathcal{C}}_{i_4}$ are violated if $L(C_{i_4}) \not\subseteq E_l$ and $R(C_{i_4}) \cap E_l \neq \emptyset$. Once E_l is fixed, satisfied constraints are removed. Note that constraints $\overline{\mathcal{C}}_{i_k}$ s.t. $k \in \{1, 2\}$ are satisfied if $L(C_{i_k}) \cap E_l \neq \emptyset$ since in this case, worlds of $R(C_{i_1})$ are necessarily in E_h with $h > l$ and worlds of $R(C_{i_2})$ are in $E_{h'}$ with $h' \geq l$. However constraints $\overline{\mathcal{C}}_{i_k}$ with $k \in \{3, 4\}$ are satisfied only when $L(C_{i_k}) \subseteq E_l$ otherwise they should be replaced by $(L(C_{i_k}) - E_l, R(C_{i_k}))$.

Algorithm 1: Handling optimistic preferences.

Data: An optimistic preference specification.
Result: A total preorder $\succeq$ on W.

begin
- $l \leftarrow 0$;
- **while** $W \neq \emptyset$ **do**
  - $-\ l \leftarrow l + 1, j \leftarrow 1$;
  - /** strict constraints **/
  - $-\ E_l = \{\omega : \forall \overline{\mathcal{C}}_{i_1}, \overline{\mathcal{C}}_{i_3} \in \overline{\mathcal{C}}, \omega \notin R(C_{i_1}) \cup R(C_{i_3})\}$;
  - **while** $j = 1$ **do**
    - $j \leftarrow 0$;
    - **for** *each* $\overline{\mathcal{C}}_{i_2}$ *and* $\overline{\mathcal{C}}_{i_4}$ *in* $\overline{\mathcal{C}}$ **do**
      - /** constraints induced by non-strict preferences **/
      - **if** $(L(C_{i_2}) \cap E_l = \emptyset$ *and* $R(C_{i_2}) \cap E_l \neq \emptyset)$ *or* $(L(C_{i_4}) \not\subseteq E_l$ *and* $R(C_{i_4}) \cap E_l \neq \emptyset)$ **then**
        - $E_l = E_l - R(C_{i_k})$;
        - $j \leftarrow 1$
  - **if** $E_l = \emptyset$ **then** Stop (inconsistent constraints);
  - – from W remove elements of E_l ;
  - /** remove satisfied constraints induced by ${}^{o}>^{o}$ preferences **/
  - – from $\overline{\mathcal{C}}$ remove $\overline{\mathcal{C}}_{i_k}$ $k \in \{1, 2\}$ such that $L(C_{i_k}) \cap E_l \neq \emptyset$;
  - /** update constraints induced by ${}^{p}>^{o}$ constraints **/
  - – replace constraints $\overline{\mathcal{C}}_{i_k}$ $(k \in \{3, 4\})$ by $(L(C_{i_k}) - E_l, R(C_{i_k}))$;
  - /** remove satisfied constraints induced by ${}^{p}>^{o}$ preferences **/
  - – from $\overline{\mathcal{C}}$ remove $\overline{\mathcal{C}}_{i_k}$ $(k \in \{3, 4\})$ with empty $L(C_{i_k})$.
- **return** $(E_1, \cdots, E_l)$

end

Example 2 *Let us consider again the optimistic preference specification given in Example 1.*
Let $W = \{\omega_0 : \neg c \neg f \neg r, \omega_1 : \neg c \neg f r, \omega_2 : \neg c f \neg r, \omega_3 : \neg c f r, \omega_4 : c \neg f \neg r, \omega_5 : c \neg f r, \omega_6 : c f \neg r, \omega_7 : c f r\}$.
We have $\overline{\mathcal{C}} = \{(\{\omega_6, \omega_7\}, \{\omega_0, \omega_1, \omega_2, \omega_3, \omega_4, \omega_5\})\} \cup \{(\{\omega_5, \omega_7\}, \{\omega_4, \omega_6\})\} \cup \{(\{\omega_5, \omega_7\}, \{\omega_1, \omega_3\})\}$.
We put in E_1 *all worlds which do not appear in the right hand side of strict constraints, we get* $E_1 = \{\omega_7\}$. *The constraint induced by* $c \wedge r \, {}^{p}\geq^{o} \neg c \wedge r$ *is not violated. The constraint induced by* $c \wedge f \, {}^{o}>^{o} \neg(c \wedge f)$ *is satisfied while the ones induced by* $c \wedge r \, {}^{p}>^{o} c \wedge \neg r$ *and* $c \wedge r \, {}^{p}\geq^{o} \neg c \wedge r$ *are not. So* $\overline{\mathcal{C}} = \{(\{\omega_5\}, \{\omega_4, \omega_6\})\} \cup \{(\{\omega_5\}, \{\omega_1, \omega_3\})\}$.
We repeat this process and get $E_2 = \{\omega_0, \omega_1, \omega_2, \omega_3, \omega_5\}$ *and* $E_3 = \{\omega_4, \omega_6\}$.

Preferences for uncontrollables

Reasoning about uncontrollables is pessimistic in the sense that an agent cannot decide the truth value of a uncontrol-

lable proposition, and thus may assume that the worst state will be realized (known as Wald's criterion).

Pessimistic reasoning semantics

A pessimistic preference specification contains four sets of preferences, which are pessimistic on their left and right hand side. This also includes the case where preferences are pessimistic with respect to their left hand side and optimistic with respect to their right side (as in optimistic reasoning semantics). This will be explained later in this section.

Definition 5 (Pessimistic preference specification) *Let $\mathcal{L}_\mathcal{U}$ be a propositional language on a set of uncontrollable propositional atoms $\mathcal{U}$. Let $\mathcal{P}_\rhd$ be a set of pessimistic preferences of the form $\{q_i \rhd r_i \mid i = 1, \cdots, n, q_i, r_i \in \mathcal{L}_\mathcal{U}\}$. A preference specification is a tuple $\langle \mathcal{P}_\rhd \mid \rhd \in \{ {}^p\!>^o,\ {}^p\!\geq^o,\ {}^p\!>^p,\ {}^p\!\geq^p\}\rangle$.*

Definition 6 (Monotonic semantics) *Let $\succeq$ be a total pre-order on W.*

- $\succeq \models q\ {}^p\!>^p r$ *iff* $\forall w \in \min(q \wedge \neg r, \succeq)$ *and* $\forall w' \in \min(\neg q \wedge r, \succeq)$ *we have* $w \succ w'$
- $\succeq \models q\ {}^p\!\geq^p r$ *iff* $\forall w \in \min(q \wedge \neg r, \succeq)$ *and* $\forall w' \in \min(\neg q \wedge r, \succeq)$ *we have* $w \succeq w'$
- $\succeq \models q\ {}^p\!>^o r$ *iff* $\forall w \in \min(q \wedge \neg r, \succeq)$ *and* $\forall w' \in \max(\neg q \wedge r, \succeq)$ *we have* $w \succ w'$
- $\succeq \models q\ {}^p\!\geq^o r$ *iff* $\forall w \in \min(q \wedge \neg r, \succeq)$ *and* $\forall w' \in \max(\neg q \wedge r, \succeq)$ *we have* $w \succeq w'$

A total pre-order $\succeq$ is a model of $\mathcal{P}_\rhd$ iff $\succeq$ satisfies each preference $q_i \rhd r_i$ in $\mathcal{P}_\rhd$.

Note that $q\ {}^p\!>^o r$ can be equivalently written as $\{q\ {}^p\!>^p r' : r'$ is a $r-$world$\}$. This is also true for $q\ {}^p\!\geq^o r$ preferences.

Pessimistic reasoning is based on the maximal specificity principle, which assumes that worlds are as bad as possible.

Definition 7 (Maximal specificity principle) *$\succeq$ belongs to the set of the most specific pre-orders among a set of pre-orders $\mathcal{O}$ if there is no $\succeq'$ in $\mathcal{O}$ such that $\succeq \sqsubset \succeq'$.*

Algorithm 2 gives the (unique) most specific preorder satisfying a pessimistic preference specification. It is similar to Algorithm 1.

This algorithm is based on the following four sets of preferences:

$$\mathcal{P}_{{}^p\!>^p} = \{C_{i_1} : q_{i_1}\ {}^p\!>^p r_{i_1}\},$$
$$\mathcal{P}_{{}^p\!\geq^p} = \{C_{i_2} : q_{i_2}\ {}^p\!\geq^p r_{i_2}\},$$
$$\mathcal{P}_{{}^p\!>^o} = \{C_{i_3} : q_{i_3}\ {}^p\!>^o r_{i_3}\},$$
$$\mathcal{P}_{{}^p\!\geq^o} = \{C_{i_4} : q_{i_4}\ {}^p\!\geq^o r_{i_4}\}.$$

Let $\overline{\mathcal{C}} = \bigcup_{k=1,\cdots,4}\{\overline{\mathcal{C}}_{i_k} = (L(C_{i_k}), R(C_{i_k}))\}$, where $L(C_{i_k}) = |q_{i_k} \wedge \neg r_{i_k}|$ and $R(C_{i_k}) = |\neg q_{i_k} \wedge r_{i_k}|$.

Algorithm 2: Handling pessimistic preferences.

Data: A pessimistic preference specification.
Result: A total pre-order $\succeq$ on W.

begin
  $l \leftarrow 0$;
  while $W \neq \emptyset$ **do**
    $l \leftarrow l + 1$, $j \leftarrow 1$;
    $E_l = \{\omega : \forall \overline{\mathcal{C}}_{i_1}, \overline{\mathcal{C}}_{i_3}$ in $\overline{\mathcal{C}}, \omega \notin L(C_{i_1}) \cup L(C_{i_3})\}$;
    while $j = 1$ **do**
      $j \leftarrow 0$;
      for *each* $\overline{\mathcal{C}}_{i_2}$ *and* $\overline{\mathcal{C}}_{i_4}$ *in* $\overline{\mathcal{C}}$ **do**
        /** constraints induced by non-strict preferences **/
        if *(*$L(C_{i_2}) \cap E_l \neq \emptyset$ *and* $R(C_{i_2}) \cap E_l = \emptyset$*) or (*$L(C_{i_4}) \cap E_l \neq \emptyset$ *and* $R(C_{i_4}) \not\subseteq E_l$*)* **then**
          $E_l = E_l - L(C_{i_k})$, $j \leftarrow 1$
    if $E_l = \emptyset$ **then** Stop (inconsistent constraints);
    – From W remove elements of E_l;
    /** remove satisfied constraints induced by ${}^p\!>^p$ preferences **/
    – From $\overline{\mathcal{C}}$ remove $\overline{\mathcal{C}}_{i_k}$ (for $k \in \{1, 2\}$) s.t. $E_l \cap R(C_{i_k}) \neq \emptyset$;
    /** update constraints induced by ${}^p\!>^o$ preferences **/
    – Replace $\overline{\mathcal{C}}_{i_k}$ (for $k \in \{3, 4\}$) in $\overline{\mathcal{C}}$ by $(L(C_{i_k}), R(C_{i_k}) - E_l)$;
    /** remove satisfied constraints induced by ${}^p\!>^o$ preferences **/
    – From $\overline{\mathcal{C}}$ remove $\overline{\mathcal{C}}_{i_k}$ ($k \in \{3, 4\}$) with empty $R(C_{i_k})$;
  return $(E'_1, \cdots, E'_l)$ *s.t.* $\forall 1 \leq h \leq l, E'_h = E_{l-h+1}$
end

Merging optimistic and pessimistic preferences

In this section we consider the merger of the least specific pre-order satisfying the optimistic preference specification, and the most specific pre-order satisfying the pessimistic preference specification. From now on, let $\mathcal{L}$ be a propositional language on disjoint sets of controllable and uncontrollable propositional atoms $\mathcal{C} \cup \mathcal{U}$. A preference specification $\mathcal{PS}$ consists of an optimistic and a pessimistic preference specification, i.e., optimistic preferences on controllables and pessimistic preferences on uncontrollables. In general, let $\succeq$ be the merger of $\succeq_o$ and $\succeq_p$. We assume that Pareto conditions hold:

Definition 8 *Let $\succeq_o$, $\succeq_p$ and $\succeq$ be three total pre-orders on the same set. $\succeq$ is a merger of $\succeq_o$ and $\succeq_p$ if and only if the following three conditions hold:*
If $w_1 \succ_o w_2$ *and* $w_1 \succ_p w_2$ *then* $w_1 \succ w_2$,
If $w_1 \succeq_o w_2$ *and* $w_1 \succeq_p w_2$ *then* $w_1 \succeq w_2$.

Given two arbitrary pre-orders, there are many possible mergers. We therefore again consider distinguished pre-orders in the subsections below. The desideratum of a merger operator is that the merger satisfies, in some sense, most of the preference specification. However, it is clearly unreasonable to ask for an operator that satisfies the whole preference specification. For example, we may have strong preferences $x\ {}^p\!>^o \neg x$ and $p\ {}^p\!>^o \neg p$, which can be satisfied

by a minimal and maximal specific pre-order separately, but which are contradictory given together. This motivates the next definition of partial satisfaction, which only considers some of the preference types.

Definition 9 *A pre-order partially satisfies a preference specification $\mathcal{PS}$ when it satisfies $\mathcal{PS}_\rhd$ with $\rhd \in \{ {}^{o}{>}^{o}, {}^{o}{\geq}^{o}, {}^{p}{>}^{p}, {}^{p}{\geq}^{p}\}$.*

The merging operators in this section satisfy our desideratum that the merger partially satisfies the preference specification, as a consequence of the following lemma. The two minimal and maximal specific pre-orders of optimistic and pessimistic preference specifications satisfy the property that no two sets are disjoint.

Lemma 1 *Let $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_m)$ be the ordered partitions of $\succeq_o$ and $\succeq_p$ respectively. We have for all $1 \leq i \leq n$ and all $1 \leq j \leq m$ that $E_i \cap E'_j \neq \emptyset$.*
Proof. Due to the fact that $\succeq_o$ and $\succeq_p$ are defined on disjoint sets of variables.

Symmetric mergers

Let $\succeq$ be the merger of $\succeq_o$ and $\succeq_p$. The least and most specific pre-orders $\succeq$ satisfying Pareto conditions, are *unique* and *identical*, and can be obtained as follows. Given Lemma 1, thus far nonempty sets E''_k do not exist, but they may exist in extensions discussed in future sections.

Proposition 1 *Let $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_m)$ be the ordered partitions of $\succeq_o$ and $\succeq_p$ respectively. The least/most specific merger of $\succeq_o$ and $\succeq_p$ is $\succeq= (E''_1, \cdots, E''_{n+m-1})$ such that if $\omega \in E_i$ and $\omega \in E'_j$ then $\omega \in E''_{i+j-1}$, and by eliminating nonempty sets E''_k and renumbering the nonempty ones in sequence.*

The symmetric merger, called also the least/most specific merger, is illustrated by the following example.

Example 3 *Consider the optimistic preference specification $p \, {}^{o}{>}^{o} \neg p$ and the pessimistic preference specification $m \, {}^{p}{>}^{p} \neg m$, where p and m stand respectively for "I will work on a* project *in order to get money" and "my boss accepts to give me* money *to pay the conference fee".*
Applying Algorithm 1 and Algorithm 2 on $p \, {}^{o}{>}^{o} \neg p$ and $m \, {}^{p}{>}^{p} \neg m$ respectively gives $\succeq_o= (\{mp, \neg mp\}, \{m\neg p, \neg m\neg p\})$ and $\succeq_p= (\{mp, m\neg p\}, \{\neg mp, \neg m\neg p\})$. The least/most specific merger is $\succeq= \{\{mp\}, \{\neg mp, m\neg p\}, \{\neg m\neg p\}\}$.

Proposition 2 *The least/most specific merger of two pre-orders satisfying Lemma 1 partially satisfies the preference specification.*

Proposition 3 *The least/most specific merger is not complete, in the sense that there are pre-orders which cannot be constructed in this way.*

Proof. Consider a language with only one controllable x and one uncontrollable p. The minimal and maximal specific pre-orders consist of at most two equivalence classes, and the least/most specific merger consists therefore of at most three equivalence classes. Hence, pre-orders in which all four worlds are distinct cannot be constructed.

We can also consider the product merger, which is a symmetric merger, defined by: if $\omega \in E_i$ and $\omega \in E'_j$ then $\omega \in E''_{i*j}$.

Dictators

We now consider dictator mergers that prefer one ordering over the other one. The *minimax merger* gives priority to the preorder $\succeq_o$ associated to the optimistic preference specification, computed following the minimal specificity principle, over $\succeq_p$ associated to the pessimistic preference specification, computed following the maximal specificity principle. Dictatorship relation of $\succ_o$ over $\succ_p$ means that worlds are first ordered with respect to $\succeq_o$ and only in the case of equality $\succeq_p$ is considered.

Definition 10 *$w_1 \succ w_2$ iff $w_1 \succ_o w_2$ or ($w_1 \sim_o w_2$ and $w_1 \succ_p w_2$).*

The minimax merger can be defined as follows.

Proposition 4 *Let $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_m)$ be the ordered partitions of $\succeq_o$ and $\succeq_p$ respectively. The result of merging $\succeq_o$ and $\succeq_p$ is $\succeq= (E''_1, \cdots, E''_{n*m})$ such that if $\omega \in E_i$ and $\omega \in E'_j$ then $\omega \in E''_{(i-1)*m+j}$.*

Example 4 *(continued) The minimax merger of the preference specification is $\{\{mp\}, \{\neg mp\}, \{m\neg p\}, \{\neg m\neg p\}\}$.*

The principle of the *maximin merger* is similar to minimax merger. The dictator here is the pre-order associated to the pessimistic preference specification and computed following the maximal specificity principle.

Definition 11 *$w_1 \succ w_2$ iff $w_1 \succ_p w_2$ or ($w_1 \sim_p w_2$ and $w_1 \succ_o w_2$).*

Example 5 *(continued) The maximin merger of the preference specification is $\{\{mp\}, \{m\neg p\}, \{\neg mp\}, \{\neg m\neg p\}\}$.*

Conditional preferences

The drawback of handling preferences on controllable and uncontrollable variables separately is the impossibility to express *interaction* between the two kinds of variables. For example my decision on whether I will work hard to finish a paper (which is a controllable variable) depends on the uncontrollable variable "money", decided by my boss. If my boss accepts to pay the conference fees then I will work hard to finish the paper. We therefore consider in the remainder of this paper preference formulas with both controllable and uncontrollable variables.

A general approach would be to define optimistic and pessimistic preference specifications on any combination of controllables and uncontrollables, such as an optimistic preference $p \, {}^{o}{>}^{o} x$ or even $q \, {}^{o}{>}^{o} r$. However, this approach blurs the idea that optimistic reasoning is restricted to controllables, and pessimistic reasoning is restricted to uncontrollables. We therefore define conditional preferences. Conditional optimistic and pessimistic preferences are defined as follows.

Definition 12 (Conditional optimistic preference specification) *Let $\mathcal{O}_\rhd$ be a set of conditional optimistic preferences of the form $\{q_i \rightarrow (x_i \rhd y_i) \mid i = 1, \cdots, n, q_i \in \mathcal{L}_\mathcal{U}, x_i, y_i \in \mathcal{L}_\mathcal{C}\}$,*

where $q \rightarrow (x \rhd y) = (q \wedge x) \rhd (q \wedge y)$. *A conditional optimistic preference specification is a tuple* $\langle \mathcal{O}_\rhd \mid \rhd \in \{ {}^p\!>^o,\ {}^p\!\geq^o\ {}^o\!>^o,\ {}^o\!\geq^o\}\rangle$.

Definition 13 (Conditional pessimistic preference specification) *Let* $\mathcal{P}_\rhd$ *be a set of conditional pessimistic preferences of the form* $\{x_i \rightarrow (q_i \rhd r_i) \mid i = 1, \cdots, n, x_i \in \mathcal{L}_\mathcal{C}, q_i, r_i \in \mathcal{L}_\mathcal{U}\}$, *where* $x \rightarrow (q \rhd r) = (x \wedge q) \rhd (x \wedge r)$. *A conditional pessimistic preference specification is a tuple* $\langle \mathcal{P}_\rhd \mid \rhd \in \{ {}^p\!>^o,\ {}^p\!\geq^o,\ {}^p\!>^p,\ {}^p\!\geq^p\}\rangle$.

In the following examples we merge the two pre-orders using the symmetric merger operator since there is no reason to give priority neither to $\succeq_o$ nor to $\succeq_p$. We start with some simple examples to illustrate that the results of the merger behaves intuitively.

Example 6 *The merger of optimistic preference* $m \rightarrow (p\ {}^o\!>^o \neg p)$ *and pessimistic preference* $\neg m\ {}^p\!>^p m$ *is the merger of* $\succeq_o = (\{mp, \neg mp, \neg m\neg p\}, \{m\neg p\})$ *and* $\succeq_p = (\{\neg mp, \neg m\neg p\}, \{mp, m\neg p\})$, *i.e.,* $\succeq = (\{\neg m\neg p, \neg mp\}, \{mp\}, \{m\neg p\})$.

The merger of optimistic preference $m \rightarrow (p\ {}^o\!>^o \neg p)$ *and pessimistic preference* $m\ {}^p\!>^p \neg m$ *is the merger of* $\succeq_o = (\{mp, \neg mp, \neg m\neg p\}, \{m\neg p\})$ *and* $\succeq_p = (\{mp, m\neg p\}, \{\neg mp, \neg m\neg p\})$, *i.e.,* $\succeq = (\{mp\}, \{\neg mp, m\neg p, \neg m\neg p\})$.

The merger of optimistic preference $m \rightarrow (p\ {}^o\!>^o \neg p)$ *and pessimistic preference* $p \rightarrow (m\ {}^p\!>^p \neg m)$ *is the merger of* $\succeq_o = (\{mp, \neg mp, \neg m\neg p\}, \{m\neg p\})$ *and* $\succeq_p = (\{mp\}, \{\neg mp, m\neg p, \neg m\neg p\})$, *i.e.,* $\succeq = (\{mp\}, \{\neg mp, \neg m\neg p\}, \{m\neg p\})$.

Proposition 5 *The most specific merger of two minimal and maximal pre-orders of conditional preference specifications does* not *necessarily partially satisfy the preference specification.*

Proof. The merger of optimistic preference $m \rightarrow (p\ {}^o\!>^o \neg p)$ *and pessimistic preference* $\neg p \rightarrow (m\ {}^p\!>^p \neg m)$ *is the merger of* $\succeq_o = (\{mp, \neg mp, \neg m\neg p\}, \{m\neg p\})$ *and* $\succeq_p = (\{m\neg p\}, \{mp, \neg mp, \neg m\neg p\})$, *i.e.,* $\succeq = (\{mp, m\neg p, \neg m\neg p, \neg mp\})$. *The merger is the universal relation which does not satisfy any non-trivial preference.*

We now consider an extension of our running example on working and money.

Example 7 *Let's consider another controllable variable* w *which stands for "I will* work *hard on the paper". Let*
$\mathcal{O} = \{money \rightarrow (work\ {}^o\!>^o \neg work),$
$\neg money \rightarrow (\neg work\ {}^o\!>^o work),$
$\neg money \rightarrow (project\ {}^p\!>^o \neg project)\}.$
This is equivalent to
$\{money \wedge work\ {}^o\!>^o money \wedge \neg work,$
$\neg money \wedge \neg work\ {}^o\!>^o \neg money \wedge work,$
$\neg money \wedge project\ {}^p\!>^o \neg money \wedge \neg project\}.$
Applying Algorithm 1 gives
$\succeq_o = (\{\neg m\neg wp, mwp, mw\neg p\}, \{m\neg w\neg p, m\neg wp, \neg mwp\},$
$\{\neg m\neg w\neg p, \neg mw\neg p\}).$

All preferences are true in $\succeq_o$. *According to these preferences, the best situations for the agent are when there is money and she works hard on the paper, or when there is no money, she works on a project but does not work hard on the paper. This is intuitively meaningful since when there is money the agent is motivated to work hard on the paper however when there is no money, it becomes necessary to work on a project which prevents her to work hard on the paper. The worst situations (as one would expect) are when there is no money and she does not work on a project.*

Example 8 *Let*
$\mathcal{P} = \{\neg project \rightarrow (money\ {}^p\!>^o \neg money),$
$\neg work \rightarrow (\neg money\ {}^p\!>^p money)\}.$
This is equivalent to
$\{\neg project \wedge money\ {}^p\!>^o \neg project \wedge \neg money,$
$\neg work \wedge \neg money\ {}^p\!>^p \neg work \wedge money\}.$
Applying Algorithm 2 gives
$\succeq_p = (\{mw\neg p, m\neg w\neg p\}, \{\neg m\neg w\neg p, \neg m\neg wp\},$
$\{\neg mw\neg p, \neg mwp, m\neg wp, mwp\}).$

Now given a preference specification $\mathcal{PS} = \mathcal{O} \cup \mathcal{P}$, the associated total pre-order is the result of combining $\succeq_o$ and $\succeq_p$ using the symmetric merger.

Example 9 *The merger of* $\succeq_o$ *and* $\succeq_p$ *given in Examples 7 and 8 respectively is* $\succeq = (\{mw\neg p\}, \{\neg m\neg wp, m\neg w\neg p\}, \{mwp\},$ $\{m\neg wp, \neg mwp, \neg m\neg w\neg p\}, \{\neg mw\neg p\})$. *The best situation is when there is money, the agent works hard on the paper and does not work on a project and the worst situation is when the agent works hard on the paper but unfortunately neither she works on a project nor there is money.*

The following example illustrates how our approach can be used in qualitative decision making. The distinction between controllable and uncontrollable variables exists in many qualitative decision theories, see e.g. (Boutilier 1994), and most recently preference logic for decision has been promoted in particular by Brewka (Brewka 2004). We use Savage's famous egg breaking example (Savage 1954), as also used by Brewka (Brewka 2004) to illustrate his extended logic programming approach in decision making.

Example 10 *An agent is preparing an omelette. 5 fresh eggs are already in the omelette. There is one more egg. She does not know whether this egg is fresh or rotten. The agent can (i) add it to the omelette which means the whole omelette may be wasted, (ii) throw it away, which means one egg may be wasted, or (iii) put it in a cup, check whether it is ok or not and put it to the omelette in the former case, throw it away in the latter. In any case, a cup has to be washed if this option is chosen.*
There is one controllable variable which consists in putting the egg $in\_omelette$, in_cup *or* $throw$ *it* $away$. *There is also an uncontrollable variable which is the state of the egg* $fresh$ *or* $rotten$. *The effects of controllable and uncontrollable variables are the following:*
$5\_omelette \leftarrow throw\_away,$
$6\_omelette \leftarrow fresh,\ in\_omelette$
$0\_omelette \leftarrow rotten,\ in\_omelette,$
$6\_omelette \leftarrow fresh,\ in\_cup,$
$5\_omelette \leftarrow rotten,\ in\_cup,$

$\neg wash \leftarrow$ *not* in_cup,
$wash \leftarrow in\_cup$.
Agent's desires are represented as follows:
$\neg wash \times wash$
$6\_omelette \times 5\_omelette \times 0\_omelette$.
We used here notations of logic programming (Brewka 2004). For example $5\_omelette \leftarrow throw\_away$ *is interpreted as: if the egg is thrown away then the agent will get an omelette with* 5 *eggs. The desire* $6\_omelette \times 5\_omelette \times 0\_omelette$ *is interpreted as: preferably* $6\_omelette$*, if not then* $5\_omelette$ *and if neither* $6\_omelette$ *nor* $5\_omelette$ *then* $0\_omelette$.
Possible solutions are:
$S_1 = \{6\_omelette, \neg wash, fresh, in\_omelette\}$,
$S_2 = \{0\_omelette, \neg wash, rotten, in\_omelette\}$,
$S_3 = \{6\_omelette, wash, fresh, in\_cup\}$,
$S_4 = \{5\_omelette, wash, rotten, in\_cup\}$,
$S_5 = \{5\_omelette, \neg wash, fresh, throw\_away\}$,
$S_6 = \{5\_omelette, \neg wash, rotten, throw\_away\}$.
Each solution is composed of an instantiation of decision variables and the satisfied desires.

Let us run this example following Brewka's approach (Brewka 2004).

Example 10 *(Continued) Brewka generates a preference order on the solutions (called answer sets in his framework) following agent's desires. Indeed* S_1 *is the single preferred solution.* S_5 *and* S_6 *are equally preferred. They are preferred to* S_2 *and* S_4 *but incomparable to* S_3*.* S_3 *is preferred to* S_4 *and incomparable to* S_5, S_6 *and* S_2*. Lastly* S_2 *and* S_4 *are incomparable.*

In our approach, controllable and uncontrollable variables are dealt with separately, respecting their distinct nature in decision theory. Our approach uses also various kinds of preferences, and non-monotonic reasoning (based on specificity algorithms) to deal with under-specification.

Example 10 *(Continued) Let us consider the following preferences on controllable and uncontrollable variables:*

$$\mathcal{O} = \begin{cases} fresh \rightarrow in\_omelette > in\_cup \\ fresh \rightarrow in\_cup > throw\_away \\ rotten \rightarrow throw\_away > in\_cup \\ rotten \rightarrow in\_cup > in\_omelette \end{cases}$$

$$\mathcal{P} = \begin{cases} in\_omelette \rightarrow fresh > rotten \\ in\_cup \rightarrow fresh > rotten \\ throw\_away \rightarrow rotten > fresh \end{cases}$$

The set of possible alternatives is $W = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6\}$ *where*
$\omega_1 = fresh \wedge in\_omelette$,
$\omega_2 = rotten \wedge in\_omelette$,
$\omega_3 = fresh \wedge in\_cup$,
$\omega_4 = rotten \wedge in\_cup$,
$\omega_5 = fresh \wedge throw\_away$ *and*
$\omega_6 = rotten \wedge throw\_away$.
We apply Algorithm 1 on the set $\mathcal{O}$ *of optimistic preferences, we get* $(\{\omega_1, \omega_6\}, \{\omega_3, \omega_4\}, \{\omega_2, \omega_5\})$.
We apply Algorithm 2 on the set $\mathcal{P}$ *of pessimistic preferences, we get* $(\{\omega_1, \omega_3, \omega_6\}, \{\omega_2, \omega_4, \omega_5\})$.
We merge the two preorders using the symmetric merger, we get $(\{\omega_1, \omega_6\}, \{\omega_3\}, \{\omega_4\}, \{\omega_2, \omega_5\})$.
Now agent's desires may be used to discriminate ω_1 *and* ω_6*. Both satisfy* $\neg wash$ *however* ω_1 *satisfies* $6\_omelette$ *while* ω_6 *satisfies* $5\_omelette$ *so* ω_1 *is preferred to* ω_6*.*
Concerning ω_2 *and* ω_5*,* ω_5 *is preferred to* ω_2*. Indeed solutions of the previous example are ordered as follows in our framework:* $S_1 \succ S_6 \succ S_3 \succ S_4 \succ S_5 \succ S_2$.

Our approach may be viewed as an extension of Brewka's approach where preferences among alternatives are used in addition to preferences among desires.

Concluding remarks

The distinction between controllable and uncontrollable propositions is fundamental in decision and control theory, and in various agent theories. Moreover, various kinds of optimistic and pessimistic reasoning are also present in many decision theories, for example in the maximin and minimax decision rules. However, their role seems to have attracted less attention in the non-monotonic logic of preference (Boella & van der Torre 2005; Dastani *et al.* 2005; Kaci & van der Torre 2005a; Lang 2004), despite the recent interest in this area, and the recent recognition that preference logic plays a key role in many knowledge representation and reasoning tasks, including decision making.

In this paper we study non-monotonic preference logic extended with the distinction between controllable and uncontrollable propositions. We illustrate how the logic can be used in decision making where preferences on controllables and preferences on uncontrollables have to be merged.

Our approach may also be used in more complex merging tasks such as social and group decision making. For example, one such extension are preferences on controllable variables conditional on preferences on uncontrollable variables, i.e. $(q \rhd_p r) \rightarrow (x \rhd_o y)$, or conversely, i.e. $(x \rhd_o y) \rightarrow (q \rhd_p r)$. This extension can be used for social decision making where an agent states its preferences given the preferences of another agent.

The following example illustrates how such social preferences can be used. Roughly, for a conditional optimistic preference $(q \rhd_p r) \rightarrow (x \rhd_o y)$, we first apply the pessimistic ordering on uncontrollables and then use the result to incorporate preferences on controllables, combining the two using the maximin merger.

Example 11 *Carl and his girlfriend Sandra go the restaurant. Menus are composed of meat or fish, wine or jus and dessert or cheese. Sandra is careful about her fitness so each menu without cake is preferred for her to all menus with cake. Even if Carl likes dessert, he does want to attempt Sandra by choosing a menu composed of a cake so, to compensate, he states that there is at least one menu composed of wine and cheese which is preferred to all menus composed of neither cake nor wine. Let* $W = \{\omega_0 : \neg d \neg w \neg m, \omega_1 : \neg d \neg w m, \omega_2 : \neg d w \neg m, \omega_3 : \neg d w m, \omega_4 : d \neg w \neg m, \omega_5 : d \neg w m, \omega_6 : d \neg w \neg m, \omega_0 : dwm\}$ *be the set*

of possible menus where m, w *and* d *stand for* meat, wine *and* dessert *respectively.* $\neg m$, $\neg w$ *and* $\neg d$ *stand for* fish, jus *and* cheese *respectively.*
Sandra's preferences give the following preorder $\succeq = (\{\omega_0, \omega_1, \omega_2, \omega_3\}, \{\omega_4, \omega_5, \omega_6, \omega_7\})$ *and Carl's preferences give the following preorder* $\succeq' = (\{\omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7\}, \{\omega_0, \omega_1\})$. *We use the maximin merger and get:* $(\{\omega_2, \omega_3\}, \{\omega_0, \omega_1\}, \{\omega_4, \omega_5, \omega_6, \omega_7\})$.

Given a set of preferences of the form $\{q_j \rhd_p r_j \rightarrow x_i \rhd_o y_i\}$, one may be tried to compute the preorders associated to $\{q_j \rhd_p r_j\}$ and $\{x_i \rhd_o y_i\}$ and then to merge them. However this way is misleading since each set of preferences may be inconsistent. The correct way would be to compute the preorder associated to each rule $q_j \rhd_p r_j \rightarrow x_i \rhd_o y_i$ as explained above and then to merge the different preorders using the symmetric merger since there is no reason to give priority to any preorder. The investigation of this idea is left to a further research.

Other topics for further research are preference specifications in which strong preferences ${}^{p}{>}^{o}$ are defined on both controllables and uncontrollables to define a stronger notion than weak satisfiability of a preference specification, the extension with beliefs, and ceteris paribus preferences (see (Kaci & van der Torre 2005b)).

References

Benferhat, S.; Dubois, D.; Prade, H.; and Williams, M. 1999. A practical approach to fusing and revising prioritized belief bases. In *Proceedings of EPIA 99, LNAI* n^o *1695, Springer Verlag*, 222–236.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2002. Possibilistic merging and distance-based fusion of propositional information. In *Annals of Mathematics and Artificial Intelligence*, volume 34(1-3), 217–252.

Boella, G., and van der Torre, L. 2005. A nonmonotonic logic for specifying and querying preferences. In *Proceedings of IJCAI'05*.

Boutilier, C. 1994. Toward a logic for qualitative decision theory. In *Proceedings KR94*, 75–86.

Brewka, G. 2004. Answer sets and qualitative decision making. In *Synthese*.

Dastani, M.; Governatori, G.; Rotolo, A.; and van der Torre, L. 2005. Preferences of agents in defeasible logic. In *Proceedings AI'05*. Springer.

Doyle, J., and Thomason, R. 1999. Background to qualitative decision theory. *AI Magazine* 20(2):55–68.

Kaci, S., and van der Torre, L. 2005a. Algorithms for a nonmonotonic logic of preferences. In *Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'05)*, 281–292.

Kaci, S., and van der Torre, L. 2005b. Non-monotonic reasoning with various kinds of preferences. In *IJCAI'05 Multidisciplinary Workshop on Advances in Preference Handling*.

Kaci, S., and van der Torre, L. 2006. Merging Optimistic and Pessimistic Preferences. In *C.R.I.L., Technical report*.

Konieczny, S., and Pérez, R. P. 1998. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento*, 488–498.

Lang, J. 2004. A preference-based interpretation of other agents' actions. In *Proceedings of KR04*, 644–653.

Lin, J., and Mendelzon, A. 1998. Merging databases under constraints. *International Journal of Cooperative Information Systems* 7(1):55–76.

Lin, J. 1996. Integration of weighted knowledge bases. *Artificial Intelligence* 83:363–378.

Revesz, P. Z. 1993. On the semantics of theory change: arbitration between old and new information. In 12^{th} *ACM SIGACT-SIGMOD-SIGART symposium on Principles of Databases*, 71–92.

Revesz, P. Z. 1997. On the semantics of arbitration. *International Journal of Algebra and Computation* 7(2):133–160.

Savage, L. 1954. The foundations of Statistics. *Dover, New York*.

Wright, G. V. 1963. *The Logic of Preference*. Edinburg. University Press.

2.5 Distance-Based Semantics for Multiple-Valued Logics

Distance-Based Semantics for Multiple-Valued Logics

Ofer Arieli
Department of Computer Science,
The Academic College of Tel-Aviv
4 Antokolski street, Tel-Aviv 61161, Israel.
oarieli@mta.ac.il

Abstract

We show that the incorporation of distance-based semantics in the context of multiple-valued consequence relations yields a general, simple, and intuitively appealing framework for reasoning with incomplete and inconsistent information.

Introduction

Reasoning with distance functions is a common way of giving semantics to formalisms that are non-monotonic in nature. The basic intuition behind this approach is that, given a set of possible worlds (alternatively, interpretations) that represent the reasoner's epistemic states or the information content of different data sources, the similarity between those worlds can be expressed quantitatively (that is, in terms of distance measurements), and thus can be evaluated by corresponding distance operators. In this respect, there is no wonder that distance semantics has played a prominent role in different paradigms for (non-monotonic) information processing. Two remarkable examples for this are the following:

- Formalisms for modeling belief revision, in which distance minimization corresponds to the idea that the difference between the reasoner's new states of belief and the old one should be kept as minimal as possible, that is, restricted only to what is really implied by the new information (see, e.g., (Lehmann, Magidor, & Schlechta 2001; Peppas, Chopra, & Foo 2004; Delgrande 2004)).
- Database integration systems (Arenas, Bertossi, & Chomicki 1999; 2003; Lin & Mendelzon 1999) and merging operators for independent data-sources (Konieczny, Lang, & Marquis 2002; Konieczny & Pino Pérez 2002), where the basic idea is that the amalgamated information should be kept coherent and at the same time as close as possible to the collective information as it is depicted by the distributed sources.

The goal of this paper is to introduce similar distance considerations in the context of *paraconsistent logics*, that is: formalisms that tolerate inconsistency and do not become trivial in the presence of contradictions (see (da Costa 1974) and (Priest 2002); some collections of papers on this topic appear, e.g., in (Batens *et al.* 2000; Carnielli, Coniglio, & Dóttaviano 2002)). One could identify at least four parties with different philosophical attitudes to such logics: the *traditionalists* defend classical logics and deny any need of paraconsistent logics. On the other extreme, the *dialetheists* contend that the world is fundamentally inconsistent and hence the true logic should be paraconsistent. The *pluralists* view inconsistent structures as fundamental but provisional, and favour their replacement, at least in empirical domains, by consistent counterparts. Finally, the *reformists* defend consistency in ontological matters, but argue that human knowledge and thinking necessarily requires inconsistency, and hence that classical logic should be replaced by a paraconsistent counterpart. The underlying theme here, following the reformists, is that conflicting data is unavoidable in practice, but it corresponds to inadequate information about the real world, and therefore it should be minimized. As we show below, this intuition is nicely and easily expressed in terms of distance semantics. Indeed, the incorporation of distance-based semantics in the context of multiple-valued consequence relations yields a framework in which a variety of paraconsistent multiple-valued logics are definable. These logics are naturally applied in many situations where uncertainty is involved.

The principle of uncertainty minimization by distance semantics is in fact a preference criterion among different interpretations of the premises. In this respect, the formalisms that are defined here may be considered as a certain kind of *preferential logics* (Shoham 1987; 1988; Makinson 1994). In particular, the intuition and the motivation behind this work is closely related to other extensions to multiple-valued semantics of the theory of preferential reasoning (see for instance (Arieli & Avron 1998; 2000; Konieczny & Marquis 2002; Arieli & Denecker 2003; Ben Naim 2005; Arieli 2004; 2006)).

The rest of this paper is organized as follows: in the next section we set up the framework; we consider basic multiple-valued entailments and define their distance-based variants. Then we consider different distance metrics and investigate some of the properties of the induced consequence relations. Finally, we discuss a generalization of the distance-based entailments to prioritized theories and show its usefulness for modeling belief revision and for consistent query answering in database systems. In the last section we conclude.

The Framework

Basic Multiple-Valued Entailments

Definition 1 Let $\mathcal{L}$ be an arbitrary propositional language. A *multiple-valued structure* for $\mathcal{L}$ is a triple $\langle \mathcal{V}, \mathcal{O}, \mathcal{D} \rangle$, where $\mathcal{V}$ is set of elements ("truth values"), $\mathcal{O}$ is a set of operations on $\mathcal{V}$ that correspond to the connectives in $\mathcal{L}$, and $\mathcal{D}$ is a nonempty proper subset of $\mathcal{V}$.

The set $\mathcal{D}$ consists of the *designated* values of $\mathcal{V}$, i.e., those that represent true assertions. In what follows we shall assume that $\mathcal{V}$ contains at least the classical values true, false, and that $\mathsf{true} \in \mathcal{D}$, $\mathsf{false} \notin \mathcal{D}$.

Definition 2 Let $\mathcal{S} = \langle \mathcal{V}, \mathcal{O}, \mathcal{D} \rangle$ be a multiple-valued structure for a propositional language $\mathcal{L}$.

a) A (multiple-valued) *valuation* ν is a function that assigns an element of $\mathcal{V}$ to each atomic formula in $\mathcal{L}$. Extensions to complex formulae are done as usual. In what follows we shall sometimes write $\nu = \{p_1 : x_1, \ldots, p_n : x_n\}$ to denote that $\nu(p_i) = x_i$ for $i = 1, \ldots, n$. The set of valuations on $\mathcal{V}$ is denoted by $\Lambda^{\mathcal{V}}$.

b) A valuation ν *satisfies* a formula ψ if $\nu(\psi) \in \mathcal{D}$.

c) A valuation ν is a *model* of a set Γ of formulae in $\mathcal{L}$, if ν satisfies every formula in Γ. The set of the models of Γ is denoted by $mod^{\mathcal{S}}(\Gamma)$.

Definition 3 Let $\mathcal{S} = \langle \mathcal{V}, \mathcal{O}, \mathcal{D} \rangle$ be a multiple-valued structure for a language $\mathcal{L}$. A *basic $\mathcal{S}$-entailment* is a relation $\models^{\mathcal{S}}$ between sets of formulae in $\mathcal{L}$ and formulae in $\mathcal{L}$, defined as follows: $\Gamma \models^{\mathcal{S}} \psi$ if every model of Γ satisfies ψ.

Example 4 In many cases the underlying semantical structure of a multiple-valued logic is a lattice, and so it is usual to include in $\mathcal{O}$ (at least) the basic lattice operations. In such cases a conjunction in $\mathcal{L}$ is associated with the join, the disjunction corresponds to the meet, and if the lattice has a negation operator, it is associated with the negation of the language. In what follows we use these definitions for the operators in $\mathcal{O}$. Now, the two-valued structure TWO is defined by the two-valued lattice, and is obtained by taking $\mathcal{V} = \{\mathsf{true}, \mathsf{false}\}$ and $\mathcal{D} = \{\mathsf{true}\}$. The corresponding entailment is denoted $\models^2$. For three-valued structures we take $\mathcal{V} = \{\mathsf{true}, \mathsf{false}, \mathsf{middle}\}$, the lattice operators in $\mathcal{O}$ are defined with respect to the total order $\mathsf{false} < \mathsf{middle} < \mathsf{true}$, and $\mathcal{D}$ is either $\{\mathsf{true}\}$ or $\{\mathsf{true}, \mathsf{middle}\}$. The structure with $\mathcal{D} = \{\mathsf{true}\}$ is denoted here by $\mathsf{THREE}_{\perp}$. The associated entailment, $\models^{3_\perp}$, corresponds to Kleene's three-valued logic (Kleene 1950). The other three-valued structure, $\mathsf{THREE}_{\top}$, corresponds to Priest's logic LP (Priest 1989; 1991).[1] Note that by different choices of the operators in $\mathcal{O}$ other three-valued logics are obtained, line weak Kleene logic, strong Kleene logic, and Łukasiewicz's logic (see, e.g., (Fitting 1990; Avron 1991)). In the four-valued case there are usually two middle elements, denoted here by both and $\mathsf{neither}$.[2] In this context it is usual to take true and both as the designated values. The corresponding structure is known as Belnap's bilattice (see (Belnap 1977a; 1977b) as well as (Arieli & Avron 1998)), and it is denoted here by FOUR. Its entailment is denoted by $\models^4$. Entailments in which $\mathcal{V}$ is the unit interval and $\mathcal{D} = \{1\}$ are common in the context of fuzzy logic (see, e.g., (Hájek 1998)). In this context it is usual to consider different kinds of operations on the unit interval (T-norms, T-conorms, residual implications, etc.), and this is naturally supported in our framework as well. The simplest case is obtained by associating $\wedge$ and $\vee$ with the meet and the join operators on the unit interval, which in this case are the same as the minimum and the maximum functions (respectively), and relating negation to the involutive operator $\neg$, defined for every $0 \leq x \leq 1$ by $\neg x = 1 - x$. In what follows we denote the corresponding structure ($\mathcal{S}$) by $[0, 1]$.

[1] Also known as J_3, RM_3, and PAC (see (D'ottaviano 1985; Rozoner 1989; Avron 1991) and chapter IX of (Epstein 1990)).

[2] The names of the middle elements correspond to their intuitive meaning as representing conflicts ('both true and false') and incomplete information ('neither true nor false').

Distance-Based Entailments

By their definition, basic $\mathcal{S}$-entailments are monotonic. In addition, some of them are trivial in the presence of contradictions (e.g., $p, \neg p \models^2 q$ and $p, \neg p \models^{3_\perp} q$), or exclude classically valid rules (e.g., $p, \neg p \vee q \not\models^{3_\top} q$ and $p, \neg p \vee q \not\models^4 q$). Common-sense reasoning, on the other hand, is frequently non-monotonic and tolerant to inconsistency. For assuring such properties we consider in what follows distance-based derivatives of the basic entailments. In the sequel, unless otherwise stated, we shall consider *finite* sets of premises in the classical propositional language $\mathcal{L} = \{\neg, \wedge, \vee, \rightarrow\}$, the operators of which correspond, respectively, to a negation, meet, join, and the material implication on the underlying lattice.

Definition 5 A total function $d : U \times U \rightarrow \mathbb{R}^+$ is called *pseudo distance* on U if it is symmetric (that is, $\forall u, v \in U \; d(u, v) = d(v, u)$) and preserves identity ($\forall u, v \in U \; d(u, v) = 0$ iff $u = v$). A *distance function* on U is a pseudo distance on U that satisfies the triangular inequality $(\forall u, v, w \in U \; d(u, v) \leq d(u, w) + d(w, v))$.

Definition 6 An *aggregation function* f is a total function that accepts arbitrarily many real numbers[3] and returns a real number. In addition, the following conditions should be satisfied: (a) f is non-decreasing in each of its arguments, (b) $f(x_1, \ldots, x_n) = 0$ if $x_1 = \ldots = x_n = 0$, and (c) $\forall x \in \mathbb{R}, f(x) = x$.

Definition 7 An *$\mathcal{S}$-distance metric* is a quadruple $\mathfrak{D} = \langle \mathcal{S}, d, f, g \rangle$, where $\mathcal{S} = \langle \mathcal{V}, \mathcal{O}, \mathcal{D} \rangle$ is a multiple-valued structure, d is a pseudo distance on the space of the $\mathcal{V}$-valued interpretations $\Lambda^{\mathcal{V}}$, and f and g are aggregation functions.

Definition 8 Given a theory $\Gamma = \{\psi_1, \ldots, \psi_n\}$, a $\mathcal{V}$-valued interpretation ν, and an $\mathcal{S}$-distance metric $\mathfrak{D} = \langle \mathcal{S}, d, f, g \rangle$, define:

- $d_f(\nu, \psi_i) = f_{\mu \in mod^{\mathcal{S}}(\psi_i)} \, d(\mu, \nu)$
- $d_g(\nu, \Gamma) = g(d_f(\nu, \psi_1), \ldots, d_f(\nu, \psi_n))$

[3] This can be formally handled by associating f with the set $\{f_n : \mathbb{R}^n \rightarrow \mathbb{R} \mid n \in \mathbb{N}\}$ of n-ary functions.

It is common to define f as the minimum function, so that a distance between an interpretation ν to a formula ψ is the minimal distance between ν and some model of ψ. Frequent choices of g are the summation function (over the distances to the formulae in Γ) and the maximal value (among those distances).

Note 9 Let $\mathfrak{D} = \langle \mathcal{S}, d, f, g\rangle$ be an $\mathcal{S}$-distance metric. As distances are non-negative numbers, by conditions (a) and (b) in Definition 6, d_f is a non-negative function for every choice of an aggregation function f. This implies that d_g is obtained by applying an aggregation function g on non-negative numbers, and so d_g is non-negative as well.

Definition 10 An $\mathcal{S}$-distance metric $\mathfrak{D} = \langle \mathcal{S}, d, f, g\rangle$ is called *normal*, if: (a) $d_f(\nu, \psi) = 0$ for every $\nu \in mod^{\mathcal{S}}(\psi)$, and (b) $g(x_1, \ldots, x_n) = 0$ only if $x_1 = \ldots = x_n = 0$.

As easily verified, the standard choices of f and g mentioned above preserve the conditions in Definition 10. Thus, for instance, for every multi-valued structure $\mathcal{S}$ and a pseudo distance d, $\mathfrak{D} = \langle \mathcal{S}, d, \min, g\rangle$ is a normal metric for each $g \in \{\Sigma, \max, \mathrm{avg}, \mathrm{median}\}$.[4]

Definition 11 Given a finite theory Γ and an $\mathcal{S}$-distance metric $\mathfrak{D} = \langle \mathcal{S}, d, f, g\rangle$, define:

$$\Delta^{\mathfrak{D}}(\Gamma) = \big\{\nu \in \Lambda^{\mathcal{V}} \mid \forall \mu \in \Lambda^{\mathcal{V}}\ d_g(\nu, \Gamma) \leq d_g(\mu, \Gamma)\big\}.$$

Proposition 12 *Let $\mathfrak{D} = \langle \mathcal{S}, d, f, g\rangle$ be a normal metric. If $mod^{\mathcal{S}}(\Gamma) \neq \emptyset$ then $\Delta^{\mathfrak{D}}(\Gamma) = mod^{\mathcal{S}}(\Gamma)$.*

Proof. If ν is a model of $\{\psi_1, \ldots, \psi_n\}$, then as $\mathfrak{D}$ is normal, $d_f(\nu, \psi_i) = 0$ for every $1 \leq i \leq n$. Thus, as g is an aggregation function, by condition (b) in Definition 6, $d_g(\nu, \Gamma) = 0$. Since $d_g(\mu, \Gamma) \geq 0$ for every $\mu \in \Lambda^{\mathcal{V}}$ (Note 9), it follows that $\nu \in \Delta^{\mathfrak{D}}(\Gamma)$.

For the converse, consider the following lemma:

Lemma 13 *In every normal metric $\langle \mathcal{S}, d, f, g\rangle$ the function g is strictly positive whenever it has at least one strictly positive argument and the rest of its arguments are non-negative.*

Lemma 13 follows from the fact that $g(x_1, \ldots, x_n) = 0$ iff $x_1 = \ldots = x_n = 0$ (by conditions (b) in Definitions 6 and 10) together with the requirements that g is non-decreasing in each of its arguments (condition (a) in Definitions 6).

To complete the proof of Proposition 12, suppose then that ν is not a model of $\{\psi_1, \ldots, \psi_n\}$. As such, it does not satisfy ψ_k for some $1 \leq k \leq n$, and so $d_f(\nu, \psi_k) > 0$. By Lemma 13, $d_g(\nu, \Gamma) > 0$ as well. On the other hand, we have shown that $d_g(\mu, \Gamma) = 0$ for every $\mu \in mod^{\mathcal{S}}(\Gamma)$, thus $\nu \notin \Delta^{\mathfrak{D}}(\Gamma)$. □

Now we are ready to define distance-based entailments:

Definition 14 For a metric $\mathfrak{D}$, define $\Gamma \models^{\mathfrak{D}} \psi$ if every valuation in $\Delta^{\mathfrak{D}}(\Gamma)$ is a model of ψ.

[4]Note that the arguments of g are non-negative numbers, and so letting g be the summation, average, or median of such numbers preserves condition (b) in Definition 10.

Example 15 Consider $\Gamma = \{p,\ \neg q,\ r,\ p \rightarrow q\}$, and let $\mathfrak{D}_2 = \langle \mathsf{TWO}, d_H, \min, \Sigma\rangle$ be a (normal) distance metric, where d_H is the Hamming distance between two-valued valuations[5]. The distances between the relevant two-valued valuations and Γ are given in the following table:

model	p	q	r	d_Σ
ν_1	true	true	true	1
ν_2	true	true	false	2
ν_3	true	false	true	1
ν_4	true	false	false	2
ν_5	false	true	true	2
ν_6	false	true	false	3
ν_7	false	false	true	1
ν_8	false	false	false	2

Thus, $\Delta^{\mathfrak{D}_2}(\Gamma) = \{\nu_1, \nu_3, \nu_7\}$, and so, for instance, $\Gamma \models^{\mathfrak{D}_2} r$, while $\Gamma \not\models^{\mathfrak{D}_2} p$ and $\Gamma \not\models^{\mathfrak{D}_2} q$. This can be intuitively explained by the fact that, unlike p and q, the atomic formula r is not related to the contradictory fragment of Γ, thus it is a reliable information that can be safely deduced from Γ.

Proposition 16 *Let $\mathfrak{D}$ be a normal $\mathcal{S}$-distance metric, and let Γ be a set of formulas in $\mathcal{L}$ such that $mod^{\mathcal{S}}(\Gamma) \neq \emptyset$. Then for every formula ψ in $\mathcal{L}$, $\Gamma \models^{\mathcal{S}} \psi$ iff $\Gamma \models^{\mathfrak{D}} \psi$.*

Proof. Immediately follows from Proposition 12. □

Some important particular cases of Proposition 16 are the following:

Corollary 17 *Let $\mathfrak{D}$ be a normal distance metric in* TWO. *For every classically consistent set of formulas Γ and for every formula ψ, $\Gamma \models^2 \psi$ iff $\Gamma \models^{\mathfrak{D}} \psi$.*

Proof. By Proposition 16, since every classically consistent theory has a model. □

Corollary 18 *Let $\mathfrak{D}$ be a normal $\mathcal{S}$-distance metric.*
a) If $\mathcal{S} =$ $\mathsf{THREE}_\top$ then $\Gamma \models^{3_\top} \psi$ iff $\Gamma \models^{\mathfrak{D}} \psi$.
b) If $\mathcal{S} =$ FOUR then $\Gamma \models^4 \psi$ iff $\Gamma \models^{\mathfrak{D}} \psi$.

Proof. By Proposition 16, since in $\mathsf{THREE}_\top$ and in FOUR, a valuation that assigns the designated middle element to every atom is a model of every theory in the classical propositional language. □

Example 19 Consider again the distance metric $\mathfrak{D}_2$ of Example 15. By Corollary 17, $\models^{\mathfrak{D}_2}$ is the same as $\models^2$ with respect to classically consistent sets of premises, but unlike the basic two-valued entailment, it does not become trivial in the presence of contradictions. On the contrary, as Example 15 shows, $\models^{\mathfrak{D}_2}$ allows to draw conclusion from inconsistent theories in a non-trivial way, and so $\models^{\mathfrak{D}_2}$ (as well as many other distance-based relations that are induced by Definition 14; see Proposition 22 below) is a *paraconsistent* consequence relation.

Consider now $\mathfrak{D}^{3_\bot} = \langle \mathsf{THREE}_\bot, d_H, \min, \Sigma\rangle$. The induced entailment, $\models^{\mathfrak{D}^{3_\bot}}$, is again paraconsistent, and with respect to consistent set of premises it coincides with Kleene's logic, $\models^{3_\bot}$ (note that the latter relation is *not* paraconsistent, so in general $\models^{3_\bot}$ and $\models^{\mathfrak{D}^{3_\bot}}$ are *not* the same). By

[5]I.e., $d_H(\nu, \mu)$ is the number of atomic formulas p such that $\nu(p) \neq \mu(p)$; see also the next section.

Corollary 18, the three-valued entailment, $\models^{\mathfrak{D}^{3\top}}$, induced by $\mathfrak{D}^{3\top} = \langle \mathsf{THREE}_\top, d_H, \min, \Sigma\rangle$, and the four-valued entailment $\models^{\mathfrak{D}_4}$, induced by $\mathfrak{D}_4 = \langle \mathsf{FOUR}, d_H, \min, \Sigma\rangle$, are paraconsistent consequence relations that coincide with the consequence relation of Priest's logic LP and with the consequence relation of Belnap's four-valued logic, respectively.

Note that the above observations still hold when the summation function in the metrics is replaced, e.g., by maximum, average, or the median function.

Reasoning with Distance-based Semantics

Distance Functions

A major consideration in the definition of the entailment relations considered in the previous section is the choice of the distance functions. In this section we consider some useful definitions of distances in the context of multiple-valued semantics. For this, we need the following notation.

Notation 20 Denote by Atoms the set of atomic formulas of the language $\mathcal{L}$ and by $\mathsf{Atoms}(\Gamma)$ the set of the atomic formulae that appear in some formula of Γ.

Many distance definitions have been considered in the literature as quantitative measurements of the level of similarity between given interpretations. For instance, the *drastic distance*, considered in (Konieczny, Lang, & Marquis 2002), is defined by

$$d_D(\nu,\mu) = \begin{cases} 0 & \text{if } \nu = \mu, \\ 1 & \text{otherwise.} \end{cases}$$

Another common measurement of the distance between two-valued interpretations is given by the *Hamming distance* that counts the number of atomic formulae that are assigned different truth values by these interpretations (see also (Dalal 1988)):

$$d_H(\nu,\mu) = |\{p \in \mathsf{Atoms} \mid \nu(p) \neq \mu(p)\}|.$$

For three-valued logics (such as Kleene's and Priest's logics considered above) it is possible to apply the same distance measurements, or to use a natural extension of the Hamming distance that considers the distance between the extreme elements true and false as strictly bigger than the distances between each one of them and the middle element. In this case, true is associated with the value 1, false is associated with 0, and the middle element corresponds to $\frac{1}{2}$. The generalized Hamming distance is then defined as follows:

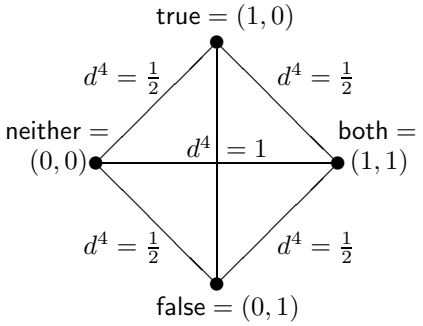
$$d^3_H(\nu,\mu) = \sum_{p\in\mathsf{Atoms}} |\nu(p) - \mu(p)|.$$

This function is used, e.g., in (de Amo, Carnielli, & Marcos 2002) as part of the semantics behind (three-valued) database integration systems.

For four-valued interpretations there is also a natural generalization of the Hamming distance. The idea here is that each one of the four truth values is associated with a pair of two-valued components as follows: $\mathsf{true} = (1,0)$, $\mathsf{false} = (0,1)$, $\mathsf{neither} = (0,0)$, $\mathsf{both} = (1,1)$. This pairwise representation preserves Belnap's original four-valued structure (see (Arieli & Denecker 2003; Arieli 2004; 2006)), and so it is a valid rewriting of the truth values. Now, the distance between two values $x = (x_1, x_2)$, $y = (y_1, y_2)$ in this pairwise representation is given by

$$d^4(x,y) = \frac{|x_1 - y_1| + |x_2 - y_2|}{2},$$

so the graphic representation of d^4 on the four-valued structure is the following:



Now, the generalized Hamming distance between two four-valued interpretations ν, μ is defined by:

$$d^4_H(\nu,\mu) = \sum_{p\in\mathsf{Atoms}} d^4\big(\nu(p), \mu(p)\big).$$

Clearly, this definition may be applied on any lattice whose elements have a pairwise representation (see (Arieli 2004; 2006)).

It is not difficult to verify that all the functions defined above satisfy the conditions in Definition 5. Below are some further observations on these distance functions:

1. Given two interpretations ν, μ into $\{\mathsf{true}, \mathsf{false}\}$, it holds that $d^4_H(\nu,\mu) = d^3_H(\nu,\mu) = d_H(\nu,\mu)$, thus d^4_H and d^3_H indeed generalize the standard Hamming distance.
2. As the following example shows, the choice of the distance function (as well as the choice of the other components of a distance metric) has a great impact on the induced entailment.

   Example 21 Consider the following two metrics:

   $$\mathfrak{D}' = \langle \mathsf{THREE}_\bot, d_H, \min, \Sigma\rangle,$$
   $$\mathfrak{D}'' = \langle \mathsf{THREE}_\bot, d^3_H, \min, \Sigma\rangle.$$

   For $\Gamma = \{p, \neg p\}$, we have

   $$\Delta^{\mathfrak{D}'}(\Gamma) = \big\{\{p{:}\mathsf{true}\}, \{p{:}\mathsf{false}\}\big\},$$
   $$\Delta^{\mathfrak{D}''}(\Gamma) = \big\{\{p{:}\mathsf{true}\}, \{p{:}\mathsf{false}\}, \{p{:}\mathsf{middle}\}\big\}.$$

   Thus, for instance, $\Gamma \models^{\mathfrak{D}'} p \vee \neg p$, while $\Gamma \not\models^{\mathfrak{D}''} p \vee \neg p$.[6]
3. In (Konieczny, Lang, & Marquis 2002) it is shown that the choice of the distance function has also a major affect on the computational complexity of the underlying formalism. See Section 4 of that paper for some complexity results of distance-based operators when $\mathcal{S} = \mathsf{TWO}$.

[6] This is so, since $\nu(p \vee \neg p) = \mathsf{middle}$ when $\nu(p) = \mathsf{middle}$, and in $\mathsf{THREE}_\bot$ the middle element is not designated.

Basic Properties of $\models^{\mathfrak{D}}$

Paraconsistency. In what follows we consider some characteristic properties of the distance-based entailments. We begin with the ability to reason with inconsistent theories in a non-trivial way. The following proposition shows that this property is common to many distance-based logics that are definable within our framework.

Proposition 22 *The consequence relations $\models^{\mathfrak{D}}$, induced by the following metrics, are all paraconsistent:*

a) $\mathfrak{D} = \langle \mathsf{TWO}, d, \min, g\rangle$, *where d is the drastic distance (d_D) or the Hamming distance (d_H) and g is either a summation or a maximum function.*

b) $\mathfrak{D} = \langle \mathsf{THREE}_{\bot}, d, \min, g\rangle$, *where $d \in \{d_D, d_H, d_H^3\}$ and g is either a summation or a maximum function.*

c) $\mathfrak{D} = \langle \mathsf{THREE}_{\top}, d, \min, g\rangle$, *where $d \in \{d_D, d_H, d_H^3\}$ and g is either a summation or a maximum function.*

d) $\mathfrak{D} = \langle \mathsf{FOUR}, d, \min, g\rangle$, *where d is any distance function of those considered in the previous section and g is either a summation or a maximum function.*

e) $\mathfrak{D} = \langle [0,1], d, \min, g\rangle$, *where d is the drastic distance or the Hamming distance and g is either a summation or a maximum function.*

Proof. For any of the items above we shall show that $p, \neg p \not\models^{\mathfrak{D}} q$, and so it is *not* the case that any formula follows from an inconsistent theory. Indeed, in item (a) we have that $\{p : \mathsf{true},\ q : \mathsf{false}\}$ (as well as $\{p : \mathsf{false},\ q : \mathsf{false}\}$) is in $\Delta^{\mathfrak{D}}(\{p, \neg p\})$, thus q does not follow from $\{p, \neg p\}$. For item (b) note that although different distance functions induce different sets of preferred models of $\{p, \neg p\}$ (see Example 21), it is easy to verify that whenever g is the summation function then $\{p : \mathsf{true},\ q : \mathsf{false}\}$ is, e.g., an element of $\Delta^{\mathfrak{D}}(\{p, \neg p\})$, and whenever g is the maximum function $\{p : \mathsf{middle},\ q : \mathsf{false}\}$ is an element of $\Delta^{\mathfrak{D}}(\{p, \neg p\})$. Thus, in both cases, q does not follow from $\{p, \neg p\}$. Part (c) holds since by Proposition 12 we have that $\Delta^{\mathfrak{D}}(\{p, \neg p\}) = mod^{3\top}(\{p, \neg p\})$, and so $\{p : \mathsf{middle}, q : \mathsf{false}\}$ is an element in $\Delta^{\mathfrak{D}}(\{p, \neg p\})$ (recall that in $\mathsf{THREE}_{\top}$ the middle element is designated, and so $\{p : \mathsf{middle}\}$ is a model of $\{p, \neg p\}$). We therefore again have that $p, \neg p \not\models^{\mathfrak{D}} q$. The proof of part (d) is similar to that of part (c) with the obvious adjustments to the four-valued case. Part (e) is similar to part (a) replacing, respectively, true and false by 1 and 0. □

Monotonicity. Next we consider monotonicity, that is: whether the set of $\models^{\mathfrak{D}}$-conclusions is non-decreasing in terms of the size of the premises. As the next two propositions show, this property is determined by the multi-valued structure and the distance metric at hand:

Proposition 23 *Let $\mathfrak{D}$ be a normal distance metric for* FOUR. *Then the corresponding distance-based entailment, $\models^{\mathfrak{D}}$, is monotonic.*

Proof. By Corollary 18(b), $\models^{\mathfrak{D}}$ is the same as the basic four-valued entailment $\models^{4}$ of Belnap's logic. The proposition now follows from the monotonicity of the latter (see (Arieli & Avron 1996, Theorem 3.10) and (Arieli & Avron 1998, Proposition 19)). □

Proposition 24 *Let $\mathfrak{D} = \langle \mathsf{TWO}, d, \min, g\rangle$ be a normal distance metric such that $g(x_1, \ldots, x_n) \leq g(y_1, \ldots, y_m)$ if $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$.*[7] *Then the corresponding distance-based entailment, $\models^{\mathfrak{D}}$, is non-monotonic.*

Proof. Consider, e.g., $\Gamma = \{p, \neg p \vee q\}$. By Corollary 17, $\Gamma \models^{\mathfrak{D}} q$. On the other hand, consider $\Gamma' = \Gamma \cup \{\neg p\}$, and let ν_t and ν_f be two-valued valuations that respectively assign true and false to p. By the assumption on g we have that

$$\begin{aligned} d_g(\nu_t, \Gamma') &= g\big(d_{\min}(\nu_t, \neg p), d_{\min}(\nu_t, \neg p \vee q)\big) \\ &\geq g\big(d_{\min}(\nu_t, \neg p)\big) \\ &= d_{\min}(\nu_t, \neg p) \\ &= d_{\min}(\nu_f, p) \\ &= g\big(d_{\min}(\nu_f, p)\big) \\ &= d_g(\nu_f, \Gamma'). \end{aligned}$$

It follows, then, that every two-valued valuation ν_f that assigns false to p is in $\Delta^{\mathfrak{D}}(\Gamma')$, no matter what value it assigns to q (as $d_g(\nu_f, \Gamma')$ is not affected by $\nu_f(q)$). In particular, $\Delta^{\mathfrak{D}}(\Gamma')$ contains valuations that assign false to q, and so $\Gamma' \not\models^{\mathfrak{D}} q$. □

Rationality. In (Lehmann & Magidor 1992), Lehmann and Magidor consider some properties that a "rational" non-monotonic consequence relation should satisfy. One property that is considered as particularly important assures that a reasoner will not have to retract any previous conclusion when learning about a new fact that has no influence on the existing set of premises. Consequence relations that satisfy this property are called *rational*. Next we show that many distant-based entailments are indeed "rational".

Notation 25 An aggregation function f is called *hereditary*, if $f(x_1, \ldots, x_n, z_1, \ldots, z_m) < f(y_1, \ldots, y_n, z_1, \ldots, z_m)$ whenever $f(x_1, \ldots, x_n) < f(y_1, \ldots, y_n)$.[8]

Proposition 26 Let $\mathfrak{D} = \langle \mathcal{S}, d, f, g\rangle$ be an $\mathcal{S}$-distance metric with a hereditary function g. If $\Gamma \models^{\mathfrak{D}} \psi$ then $\Gamma, \phi \models^{\mathfrak{D}} \psi$ for every ϕ such that $\mathsf{Atoms}(\Gamma \cup \{\psi\}) \cap \mathsf{Atoms}(\phi) = \emptyset$.

Intuitively, the condition on ϕ in Proposition 26 guarantees that ϕ is 'irrelevant' for Γ and ψ. The intuitive meaning of Proposition 26 is, therefore, that the reasoner does not have to retract ψ when learning that ϕ holds.

Proof of Proposition 26. Let $\mu \in \Lambda^{\mathcal{V}}$ be a valuation that does not satisfy ψ. As $\Gamma \models^{\mathfrak{D}} \psi$ while $\mu(\psi) \notin \mathcal{D}$, necessarily μ is not in $\Delta^{\mathfrak{D}}(\Gamma)$, and so there is a valuation ν in $\Delta^{\mathfrak{D}}(\Gamma)$, for which $d_g(\nu, \Gamma) < d_g(\mu, \Gamma)$. Again, since $\Gamma \models^{\mathfrak{D}} \psi$, $\nu(\psi) \in \mathcal{D}$. Assuming that $\Gamma = \{\psi_1, \ldots, \psi_n\}$, we have that

$$g(d_f(\nu, \psi_1), \ldots, d_f(\nu, \psi_n)) < g(d_f(\mu, \psi_1), \ldots, d_f(\mu, \psi_n)).$$

Now, consider a valuation σ, defined for every atom p as follows:

$$\sigma(p) = \begin{cases} \nu(p) & \text{if } p \in \mathsf{Atoms}(\Gamma \cup \psi) \\ \mu(p) & \text{otherwise} \end{cases}$$

[7] As the arguments of g are non-negative, summation, maximum, and many other aggregation functions satisfy this property.

[8] Note that heredity, unlike monotonicity, is defined by strict inequalities. Thus, for instance, the summation is hereditary, while the maximum function is not.

Note that $\sigma(p) = \nu(p)$ for every $p \in \mathsf{Atoms}(\psi)$, and so $\sigma(\psi) \in \mathcal{D}$ as well. As $\mathsf{Atoms}(\Gamma \cup \{\psi\}) \cap \mathsf{Atoms}(\phi) = \emptyset$ and since g is hereditary, we have that

$$\begin{aligned} d_g(\sigma, \Gamma \cup \{\phi\}) &= g(d_f(\sigma,\psi_1), \ldots, d_f(\sigma,\psi_n), d_f(\sigma,\phi)) \\ &= g(d_f(\nu,\psi_1), \ldots, d_f(\nu,\psi_n), d_f(\mu,\phi)) \\ &< g(d_f(\mu,\psi_1), \ldots, d_f(\mu,\psi_n), d_f(\mu,\phi)) \\ &= d_g(\mu, \Gamma \cup \{\phi\}). \end{aligned}$$

Thus, for every valuation μ such that $\mu(\psi) \notin \mathcal{D}$ there is a valuation σ such that $\sigma(\psi) \in \mathcal{D}$ and $d_g(\sigma, \Gamma \cup \{\phi\}) < d_g(\mu, \Gamma \cup \{\phi\})$. It follows that the elements of $\Delta^{\mathfrak{D}}(\Gamma \cup \{\phi\})$ must satisfy ψ, and so $\Gamma, \phi \models^{\mathfrak{D}} \psi$. □

Adaptivity. The ability to handle theories with contradictions in a nontrivial way and at the same time to presuppose a consistency of all sentences 'unless and until proven otherwise', is called *adaptivity* (Batens 1989; 1998). Consequence relations with this property *adapt* to the *specific* inconsistencies that occur in the theories. For instance, a plausible inference mechanism should *not* apply the Disjunctive Syllogism for concluding that q follows from $\{p, \neg p, \neg p \vee q\}$. On the other hand, in the case of $\{p, \neg p, r, \neg r \vee q\}$, applying the Disjunctive Syllogism to r and $\neg r \vee q$ may be justified by the fact that the subset of formulae to which the Disjunctive Syllogism is applied should not be affected by the inconsistency of the whole theory, therefore inference rules that are classically valid can be applied to it.

The following proposition shows that in many cases distance-based entailments are adaptive. If a given theory can be split up to a consistent and an inconsistent parts, then every assertion that is not related to the inconsistent part, and which classically follows from the consistent part, must be entailed by the whole theory.

Proposition 27 Let $\mathfrak{D} = \langle \mathcal{S}, d, f, g \rangle$ be a normal $\mathcal{S}$-distance metric with a hereditary function g. Suppose that Γ is a theory that can be represented as $\Gamma' \cup \Gamma''$, where $mod^{\mathcal{S}}(\Gamma') \neq \emptyset$ and $\mathsf{Atoms}(\Gamma') \cap \mathsf{Atoms}(\Gamma'') = \emptyset$. Then for every formula ψ such that $\mathsf{Atoms}(\psi) \cap \mathsf{Atoms}(\Gamma'') = \emptyset$, it holds that if $\Gamma' \models^{\mathcal{S}} \psi$ then $\Gamma \models^{\mathfrak{D}} \psi$.

Proof. If $\Gamma' \models^{\mathcal{S}} \psi$, then by Proposition 16, $\Gamma' \models^{\mathfrak{D}} \psi$. Now, as $\mathsf{Atoms}(\Gamma' \cup \{\psi\}) \cap \mathsf{Atoms}(\Gamma'') = \emptyset$, we have, by Proposition 26, that $\Gamma \models^{\mathfrak{D}} \psi$. □

Distance-based Entailments for Prioritized Theories

$\models^{\mathfrak{D}}$, Generalized

We now extend the distance-based semantics of the previous section to *prioritized theories*. An n-prioritized theory is a theory $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$, where the sets Γ_i $(1 \leq i \leq n)$ are pairwise disjoint. Intuitively, when $i < j$ the formulas in Γ_i are preferred than those in Γ_j. A common situation in which theories are prioritized is, e.g., when data-sources are augmented with integrity constraints. In such cases the corresponding theory has two priority levels, as the integrity constraints must always be satisfied, while the data facts may be revised in case of conflicts.

To formalize the existence of different levels of priority in prioritized theories, we consider the following sequence of sets: for a metric $\mathfrak{D} = \langle \mathcal{S}, d, f, g \rangle$ and an n-prioritized theory $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$, define:

- $\Delta_1^{\mathfrak{D}}(\Gamma) = \{\nu \in \Lambda^{\mathcal{V}} \mid \forall \mu \in \Lambda^{\mathcal{V}}\ d_g(\nu, \Gamma_1) \leq d_g(\mu, \Gamma_1)\}$
- for every $1 < i \leq n$, let
  $\Delta_i^{\mathfrak{D}}(\Gamma) = \{\nu \in \Delta_{i-1}^{\mathfrak{D}}(\Gamma) \mid$
  $\forall \mu \in \Delta_{i-1}^{\mathfrak{D}}(\Gamma)\ d_g(\nu, \Gamma_i) \leq d_g(\mu, \Gamma_i)\}$

Definition 28 Given an $\mathcal{S}$-distance metric $\mathfrak{D}$, define for every n-prioritized theory Γ and formula ψ, $\Gamma \models^{\mathfrak{D}} \psi$ if every valuation in $\Delta_n^{\mathfrak{D}}(\Gamma)$ satisfies ψ.

Note that the last definition is a conservative extension of Definition 14, since for non-prioritized theories (i.e., when $n = 1$) the two definitions coincide.

Example 29 Consider the following puzzle, known as the Tweety dilemma:

$$\Gamma = \left\{ \begin{array}{l} \mathtt{bird(x)} \rightarrow \mathtt{fly(x)}, \\ \mathtt{penguin(x)} \rightarrow \mathtt{bird(x)}, \\ \mathtt{penguin(x)} \rightarrow \neg\mathtt{fly(x)}, \\ \mathtt{bird(Tweety)}, \\ \mathtt{penguin(Tweety)} \end{array} \right\}$$

As this theory is not consistent, everything classically follows from it, including, e.g., $\mathtt{fly(Tweety)}$, which seems a counter-intuitive conclusion in this case, as penguins should not fly, although they are birds. The reason for this anomaly is that all the formulas above have the same importance, in contrast to the intuitive understanding of this case. Indeed,

1. The confidence level of strict facts ($\mathtt{bird(Tweety)}$ and $\mathtt{penguin(Tweety)}$ in our case) is usually at least as high as the confidence level of general rules (implications).
2. As penguins *never* fly, and this is a characteristic feature of penguins (without exceptions), one would probably like to attach to the assertion $\mathtt{penguin(x)} \rightarrow \neg\mathtt{fly(x)}$ a higher priority than that of $\mathtt{bird(x)} \rightarrow \mathtt{fly(x)}$, which states only a default property of birds.[9]

Consider now the metric $\mathfrak{D} = \langle \mathsf{TWO}, d_H, \min, \Sigma \rangle$ and regard Γ as a prioritized theory in which the two considerations above are satisfied. It is easy to verify that the unique valuation in $\Delta_n^{\mathfrak{D}}(\Gamma)$ (where $n > 1$ is the number of priority levels in Γ) assigns true to $\mathtt{bird(Tweety)}$, true to $\mathtt{penguin(Tweety)}$, and false to $\mathtt{fly(Tweety)}$. Thus, e.g., $\Gamma \models^{\mathfrak{D}} \neg\mathtt{fly(Tweety)}$, as intuitively expected.

Applications

In this section we show how the generalized distance-based semantics for prioritized theories, introduced in the previous section, can be naturally applied in related areas. Below we consider two representative examples: database query systems and belief revision theory.

[9] The third assertion, $\mathtt{penguin(x)} \rightarrow \mathtt{bird(x)}$, could have an intermediate priority, as again there are no exceptions to the fact that every penguin is a bird, but still penguins are not *typical* birds, thus they shouldn't inherit all the properties we expect birds to have.

A. Consistent Query Answering in Database Systems
A particularly important context in which reasoning with prioritized theories naturally emerges is consistency handling in database systems. In such systems, it is of practical importance to enforce the validity of the data facts by a set of integrity constraints. In case of any violation of some integrity constraint, the set of data-facts is supposed to be modified in order to restore the database consistency. It follows, then, that integrity constraints are superior than the facts themselves, and so the underlying theory is a prioritized one. This also implies that consistent query answering from possibly inconsistent databases (Arenas, Bertossi, & Chomicki 1999; 2003; Greco & Zumpano 2000; Bravo & Bertossi 2003; Eiter 2005) or constraint data-sources (Konieczny, Lang, & Marquis 2002; Konieczny & Pino Pérez 2002) may be defined in terms of distance-based entailments on prioritized theories. Moreover, as our framework is tolerant to different semantics, such methods of query answering, which are traditionally two-valued ones, may be related to other formalisms that are based on many-valued semantics like those considered in (Subrahmanian 1994) and (de Amo, Carnielli, & Marcos 2002).

Let $\mathcal{L}$ be a propositional language with Atoms its underlying set of atomic propositions. A (propositional) *database instance* $\mathcal{I}$ is a finite subset of Atoms. The semantics of a database instance is given by the conjunction of the atoms in $\mathcal{I}$, augmented with the *closed world assumption* ($\mathsf{CWA}(\mathcal{I})$) (Reiter 1978) that assures that each atom which is not explicitly mentioned in $\mathcal{I}$ is false.

Definition 30 A *database* is a pair $(\mathcal{I}, \mathcal{C})$, where $\mathcal{I}$ is a database instance, and $\mathcal{C}$ — the set of *integrity constraints* — is a finite and consistent set of formulae in $\mathcal{L}$. A database $\mathcal{DB} = (\mathcal{I}, \mathcal{C})$ is *consistent* if every formula in $\mathcal{C}$ follows from $\mathcal{I}$, that is, there is no integrity constraint that is violated in $\mathcal{I}$.

Given a database $\mathcal{DB} = (\mathcal{I}, \mathcal{C})$, the theory $\Gamma_{\mathcal{DB}}$ that is associated with it contains the components of $\mathcal{DB}$ and imposes the closed word assumption on $\mathcal{I}$. In addition, this theory should reflect the fact that the integrity constraints in $\mathcal{C}$ are of higher priority than the rest of the data. That is, $\Gamma_{\mathcal{DB}}$ should be a two-leveled theory, in which $\Gamma_1 = \mathcal{C}$ and $\Gamma_2 = \mathcal{I} \cup \mathsf{CWA}(\mathcal{I})$. Now, query answering with respect to $\mathcal{DB}$ may be defined in terms of a distance-based entailment on $\Gamma_{\mathcal{DB}}$.

Suppose, then, that $\mathfrak{D}$ is a normal $\mathcal{S}$-distance metric for some multiple-valued structure $\mathcal{S}$, and let $\mathcal{DB} = (\mathcal{I}, \mathcal{C})$ be a (possibly inconsistent) database. Its prioritized theory is

$$\Gamma_{\mathcal{DB}} = \Gamma_1 \cup \Gamma_2 = \mathcal{C} \cup (\mathcal{I} \cup \mathsf{CWA}(\mathcal{I})),$$

and $\mathcal{Q}$ is a consistent query answer if $\Gamma_{\mathcal{DB}} \models^{\mathfrak{D}} \mathcal{Q}$. Now, as $\mathcal{C}$ is classically consistent, by Proposition 12, $\Delta_1^{\mathfrak{D}}(\Gamma_{\mathcal{DB}}) = mod(\mathcal{C})$. It follows, therefore, that $\mathcal{Q}$ is a consistent query answer of $\mathcal{DB}$ if it is satisfied by every model of $\mathcal{C}$ with minimal distance (in terms of d_g) from $\mathcal{I} \cup \mathsf{CWA}(\mathcal{I})$.

Example 31 Let $\mathcal{DB} = (\{p, r\}, \{p \to q\})$. Here,

$$\mathcal{I} \cup \mathsf{CWA}(\mathcal{I}) = \mathcal{I} \cup \{\neg x \mid x \notin \mathcal{I}\} = \{p, \neg q, r\},$$

so the associated theory is

$$\Gamma_{\mathcal{DB}} = \{p \to q\} \cup \{p, \neg q, r\}.$$

This theory is the same as the one considered in Example 15, but with one major difference: now $p \to q$ is preferred over the other formulas, thus only its models are taken into account. Consider the same metric as that of Example 15. As valuations ν_3, ν_4 in the table of that example do not satisfy $\mathcal{C}$, they are excluded. Among the remaining valuations, ν_1 and ν_7 are the closest to $\mathcal{I} \cup \mathsf{CWA}(\mathcal{I})$, and so the consistent query answers of $(\mathcal{I}, \mathcal{C})$ are the formulas that are satisfied by both ν_1 and ν_7.

Note 32 Example 31 shows, in particular, that $\models^{\mathfrak{D}}$ is *not* reflexive, since for instance $\Gamma_{\mathcal{DB}} \not\models^{\mathfrak{D}} p$ although $p \in \Gamma_{\mathcal{DB}}$. This can be justified by the fact that one way of restoring the consistency of $\mathcal{DB}$ is by removing p from $\mathcal{I}$ (ν_7 corresponds to this situation), and so p does not hold in all the consistency 'repairs' of $\Gamma_{\mathcal{DB}}$.[10] Similarly, the fact that $\Gamma_{\mathcal{DB}} \not\models^{\mathfrak{D}} \neg q$ although $\neg q \in \Gamma_{\mathcal{DB}}$ may be justified by the alternative way of restoring the consistency of $\mathcal{DB}$, in which q is added to $\mathcal{I}$ (ν_1 corresponds to this situation). Note also that there is no reason to remove r from $\mathcal{I}$, as this will not contribute to the consistency restoration of $\mathcal{DB}$. This intuitively justifies the fact that for r (unlike the other atomic formulae in $\Gamma_{\mathcal{DB}}$), we do have that $\Gamma_{\mathcal{DB}} \models^{\mathfrak{D}} r$ (cf. Example 15). This is also to the intuition behind the query answering formalisms for inconsistent databases, considered e.g. in (Arenas, Bertossi, & Chomicki 1999; 2003; Greco & Zumpano 2000; Bravo & Bertossi 2003; Eiter *et al.* 2003; Arieli *et al.* 2004; 2006).

B. Modeling of Belief Revision
A belief revision theory describes how a belief state is obtained by the revision of a belief state $\mathcal{B}$ by some new information, ψ. A belief revision operator $\circ$ describes the kind of information change that should be made in face of the new (possibly contradicting) information depicted by ψ. The new belief state, denoted $\mathcal{B} \circ \psi$, is usually characterized by the closest worlds to $\mathcal{B}$ in which ψ holds. This criterion, often called *the principle of minimal change*, is one of the most widely advocated postulates of belief revision theory. Clearly, it is derived by distance considerations, so it is not surprising that this consideration can be expressed in our framework. Indeed, the intended meaning of the revision operator is to describe 'how to revise $\mathcal{B}$ in order to be consistent with ψ'. In our context the revised belief state corresponds to the (coherent) set of conclusions that can be derived from the prioritized theory $\{\psi\} \cup \mathcal{B}$, in which ψ is superior than $\mathcal{B}$. Indeed, suppose again that $\mathfrak{D}$ is a normal $\mathcal{S}$-distance metric for some multiple-valued structure $\mathcal{S}$, and consider $\Gamma = \Gamma_1 \cup \Gamma_2 = \{\psi\} \cup \mathcal{B}$. Again, by Proposition 12, $\Delta_1^{\mathfrak{D}}(\Gamma) = mod(\psi)$, and so the new belief state consists of the formulas that are satisfied by every model of ψ and that are minimally distant (in terms of d_g) from $\mathcal{B}$. In other words,

$$\mathcal{B} \circ \psi \;=\; \Delta_2^{\mathfrak{D}}(\Gamma), \qquad (1)$$

[10] Or, equivalently, p is involved in contradictions in $\Gamma_{\mathcal{DB}}$; see also the discussion in Example 15 above.

where $\Gamma = \Gamma_1 \cup \Gamma_2$, $\Gamma_1 = \{\psi\}$, and $\Gamma_2 = \mathcal{B}$.

Example 33 For $\mathfrak{D}_2 = \langle \mathsf{TWO}, d_H, \min, \Sigma \rangle$ define a belief revision operator $\circ$ by Equation (1) above. The revision operator that is obtained is the same as the one considered in (Dalal 1988). It is well-known that this operator satisfies the AGM postulates (Alchourrón, Gärdenfors, & Makinson 1985).

Conclusion

In this paper we have introduced a family of multiple-valued entailments, the underlying semantics of which is based on distance considerations. It is shown that such entailments can be incorporated in a variety of deductive systems, mediators of distributed databases, consistent query answering engines, and formalisms for belief revision.

A characteristic property of the entailments considered here is that, although being paraconsistent in nature, to a large extent they retain consistency. For instance, the entailments that are defined by normal distance metrics in a two-valued (respectively, $\mathcal{S}$-valued) semantics, are identical to classical two-valued entailment (respectively, are identical to the corresponding basic $\mathcal{S}$-entailment), as long as the set of premises is kept consistent. Moreover, even when the set of premises becomes inconsistent, the conclusions that are obtained from the fragment of the theory that is not related to the 'core' of the inconsistency, are the same as those obtained by the classical two-valued (respectively, the basic $\mathcal{S}$-valued) entailment, when only the consistent fragment is taken into account. In contrast to the classical entailment, however, our formalisms are not degenerated in the presence of contradictions, so the set of conclusions is not 'exploded' is such cases.

References

Alchourrón, C. E.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision function. *Journal of Symbolic Logic* 50:510–530.

Arenas, M.; Bertossi, L.; and Chomicki, J. 1999. Consistent query answers in inconsistent databases. In *Proc. 18th Symp. on Principles of Database Systems* (PODS'99), 68–79.

Arenas, M.; Bertossi, L.; and Chomicki, J. 2003. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming* 3(4–5):393–424.

Arieli, O., and Avron, A. 1996. Reasoning with logical bilattices. *Journal of Logic, Language, and Information* 5(1):25–63.

Arieli, O., and Avron, A. 1998. The value of the four values. *Artificial Intelligence* 102(1):97–141.

Arieli, O., and Avron, A. 2000. Bilattices and paraconsistency. In Batens, D.; Mortenson, C.; Priest, G.; and Van Bendegem, J., eds., *Frontiers of Paraconsistent Logic*, volume 8 of *Studies in Logic and Computation*. Research Studies Press. 11–27.

Arieli, O., and Denecker, M. 2003. Reducing preferential paraconsistent reasoning to classical entailment. *Logic and Computation* 13(4):557–580.

Arieli, O.; Denecker, M.; Van Nuffelen, B.; and Bruynooghe, M. 2004. Database repair by signed formulae. In Seipel, D., and Turull Torres, J. M., eds., *Proc. 3rd Symp. on Foundations of Information and Knowledge Systems* (FoIKS'02), number 2942 in LNCS, 14–30. Springer.

Arieli, O.; Denecker, M.; Van Nuffelen, B.; and Bruynooghe, M. 2006. Computational methods for database repair by signed formulae. *Annals of Mathematics and Artificial Intelligence* 46(1–2):4–37.

Arieli, O. 2004. Paraconsistent preferential reasoning by signed quantified Boolean formulae. In de Mántaras, R., and Saitta, L., eds., *Proc. 16th European Conference on Artificial Intelligence* (ECAI'04), 773–777. IOS Press.

Arieli, O. 2006. Paraconsistent reasoning and preferential entailments by signed quantified Boolean formulae. *ACM Transactions on Computational Logic*. Accepted.

Avron, A. 1991. Natural 3-valued logics: Characterization and proof theory. *Journal of Symbolic Logic* 56(1):276–294.

Batens, D.; Mortenson, C.; Priest, G.; and Bendegem, J. V., eds. 2000. *Frontiers of Paraconsistent Logic*. Research Studies Press.

Batens, D. 1989. Dynamic dialectical logics. In Priest, G.; Routely, R.; and Norman, J., eds., *Paraconsistent Logic. Essay on the Inconsistent*. Philosophia Verlag. 187–217.

Batens, D. 1998. Inconsistency-adaptive logics. In Orlowska, E., ed., *Logic at Work*. Physica Verlag. 445–472.

Belnap, N. D. 1977a. How a computer should think. In Ryle, G., ed., *Contemporary Aspects of Philosophy*. Oriel Press. 30–56.

Belnap, N. D. 1977b. A useful four-valued logic. In Dunn, J. M., and Epstein, G., eds., *Modern Uses of Multiple-Valued Logics*. Reidel Publishing Company. 7–37.

Ben Naim, J. 2005. Preferential and preferential-discriminative consequence relations. *Logic and Computation* 15(3):263–294.

Bravo, L., and Bertossi, L. 2003. Logic programming for consistently querying data integration systems. In Gottlob, G., and Walsh, T., eds., *Proc. 18th Int. Joint Conference on Artificial Intelligence* (IJCAI'03), 10–15.

Carnielli, W. A.; Coniglio, M. E.; and Dóttaviano, I., eds. 2002. *Paraconsistency: The Logical Way to the Inconsistent*, volume 228 of *Lecture Notes in Pure and Applied Mathematics*. Marcel Dekker.

da Costa, N. C. A. 1974. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic* 15:497–510.

Dalal, M. 1988. Investigations into a theory of knowledge base revision. In *Proc. National Conference on Artificial Intelligence* (AAAI'98), 475–479. AAAI Press.

de Amo, S.; Carnielli, W. A.; and Marcos, J. 2002. A logical framework for integrating inconsistent information

in multiple databases. In *Proc. 2nd Int. Symp. on Foundations of Information and Knowledge Systems* (FoIKS'02), number 2284 in LNCS, 67–84. Springer.

Delgrande, J. 2004. Preliminary considerations on the modelling of belief change operators by metric spaces. In *Proc. Int. Workshop on Non-Monotonic Reasoning* (NMR'04), 118–125.

D'ottaviano, I. 1985. The completeness and compactness of a three-valued first-order logic. *Revista Colombiana de Matematicas* XIX(1–2):31–42.

Eiter, T.; Fink, M.; Greco, G.; and Lembo, D. 2003. Efficient evaluation of logic programs for querying data integration systems. In *Proc. 19th Int. Conf. on Logic Programming* (ICLP'03), number 2916 in LNCS, 163–177. Springer.

Eiter, T. 2005. Data integration and answer set programming. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Proc. 8th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (LPNMR'05), number 3662 in LNCS, 13–25. Springer.

Epstein, R. L. 1990. *The semantic foundation of logic. Vol.I: propositional logics*. Kluwer.

Fitting, M. 1990. Kleene's logic, generalized. *Logic and Computation* 1:797–810.

Greco, S., and Zumpano, E. 2000. Querying inconsistent databases. In *Proc. Int. Conf. on Logic Programming and Automated Reasoning* (LPAR'2000), number 1955 in LNAI, 308–325. Springer.

Hájek, P. 1998. *Metamatematics of Fuzzy Logic*. Kluwer.

Kleene, S. C. 1950. *Introduction to Metamathematics*. Van Nostrand.

Konieczny, S., and Marquis, P. 2002. Three-valued logics for inconsistency handling. In Flesca, S.; Greco, S.; Leone, N.; and Ianni, G., eds., *Proc. 8th European Conference on Logics in Artificial Intelligence* (JELIA'02), number 2424 in LNAI, 332–344. Springer.

Konieczny, S., and Pino Pérez, R. 2002. Merging information under constraints: a logical framework. *Logic and Computation* 12(5):773–808.

Konieczny, S.; Lang, J.; and Marquis, P. 2002. Distance-based merging: A general framework and some complexity results. In *Proc 8th Int. Conf. on Principles of Knowledge Representation and Reasoning* (KR'02), 97–108.

Lehmann, D., and Magidor, M. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55:1–60.

Lehmann, D.; Magidor, M.; and Schlechta, K. 2001. Distance semantics for belief revision. *Journal of Symbolic Logic* 66(1):295–317.

Lin, J., and Mendelzon, A. O. 1999. Knowledge base merging by majority. In *Dynamic Worlds: From the Frame Problem to Knowledge Management*. Kluwer.

Makinson, D. 1994. General patterns in nonmonotonic reasoning. In Gabbay, D.; Hogger, C.; and Robinson, J., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3. Oxford Science Publications. 35–110.

Peppas, P.; Chopra, S.; and Foo, N. 2004. Distance semantics for relevance-sensitive belief revision. In *Proc 9th Int. Conf. on Principles of Knowledge Representation and Reasoning* (KR'04), 319–328. AAAI Press.

Priest, G. 1989. Reasoning about truth. *Artificial Intelligence* 39:231–244.

Priest, G. 1991. Minimally inconsistent LP. *Studia Logica* 50:321–331.

Priest, G. 2002. Paraconsistent logic. In Gabbay, D., and Guenthner, F., eds., *Handbook of Philosophical Logic*, volume 6. Kluwer. 287–393.

Reiter, R. 1978. On closed world databases. In *Logic and Databases*. Plenum Press. 55–76.

Rozoner, L. I. 1989. On interpretation of inconsistent theories. *Information Sciences* 47:243–266.

Shoham, Y. 1987. A semantical approach to nonmonotonic logics. In Ginsberg, M. L., ed., *Readings in Non-Monotonic Reasoning*. Morgan Kaufmann Publishers. 227–249.

Shoham, Y. 1988. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press.

Subrahmanian, V. S. 1994. Amalgamating knowledge bases. *ACM Transactions on Database Systems* 19(2):291–331.

2.6 On Compatibility and Forward Chaining Normality

On Compatibility and Forward Chaining Normality*

Mingyi Zhang[1,2] and **Ying Zhang**[1]
[1]Guizhou Academy of Sciences, Guiyang, P. R. China.
[2]School of Information Engineering,
Guizhou University, P. R. China.
zhangmingyi045@yahoo.com.cn

Yisong Wang[3]
[3]School of Computer Science & Engineering,
Guizhou University, Guiyang, P. R. China.
ys_wang168@yahoo.com.cn

Abstract

In Reiter's default logic, the class of normal default theories is an important subclass of default theories. All defaults in this subclass have a simple syntactic criterion and the subclass has a number of nice properties, which makes it a desirable context for belief revision (Reiter 1980). But this simple syntactic criterion has a side effect — interacting defaults would lead to anomalous conclusions (Reiter & Criscuolo 1981). Auto-compatible default theories (Mingyi 1992; 1993), which is obtained by introducing the notions of (strongly) compatibility and auto-compatibility of defaults, and Forward Chaining normal (FC-normal) default theories (Marek, Nerode, & Remmel 1994) which employed the notion of normal default theories with respect to a consistency property are two larger subclasses than normal default theories, and both enjoy all the desirable properties of normal default theories.

In this paper we extend the class of auto-compatible default theories to weakly auto-compatible default theories, present a sound and complete algorithm to compute all of its extensions, and show that weakly auto-compatible theories have the same nice properties as the auto-compatible default theories. We argue that every FC-normal default theory is weakly auto-compatible. Moreover, we also show that this class properly contains FC-normal default theories, i.e., there are some weakly auto-compatible default theories which are not FC-normal. We also point out that it is easy to apply the notions of (weakly) auto-compatibility to general logic programs and truth maintenance systems as well.

1 Introduction

Default logic (DL) (Reiter 1980) is one of the best known and most widely studied formalizations of non-monotonic reasoning due to its very expressive and lucid language (Marek & Truszczynski 1993; Makinson 2005). However, the existence of extensions for a default theory (DT) is not guaranteed and the construction of extensions is quite complex. So many researchers had proposed several variants of default logic (Lukaszewicz 1985; Brewka 1991; Delgrande, Schaub, & Jackson 1994; Mikitiuk & Truszczynski 1993; Przymusinska & Przymusinski 1994; Giordano & Martelli 1994; Brewka & Gottlob 1997). Many of these variants put forward the formal property of semi-monotonicity because it guarantees the existence of extensions and it allows for incremental construction. And the other variants address the expressive power of default logic, which was diminished by semi-monotonicity. An important class of default theories enjoying semi-monotonicity is normal default theories in Reiter's framework. One particular feature of the class is that a simple syntactic criterion which enables us to determine easily whether a given default theory is normal or not. The class of normal default theories has the following remarkable properties: the existence of extensions, semi-monotonicity and a default proof theory. But it has a side effect — interacting defaults would lead to a counterintuitive conclusion. It is natural to ask that whether one can extend normal default theories to a larger subclass of default theories, which have all the above desirable properties of normal default theories.

By introducing the notions of (strongly) compatibility and auto-compatibility of defaults, we proposed a larger class of default theories, so-called auto-compatible default theories (Mingyi 1992). We proved that an extension of any auto-compatible default theory always exists and the class of auto-compatible default theories strictly contains all of normal default theories. We also pointed out that auto-compatible default theories have all the desirable properties of normal default theories (Mingyi 1992; 1994). Then we detailed some important properties of auto-compatible default theories (Mingyi 1996), which are possessed by normal default theories. Marek *et al* extended the notion of normal default theories with respect to a consistency property and also got a larger subclass of default theories, so-called Forward Chaining normal (FC-normal) default theories (Marek, Nerode, & Remmel 1994), which has the same nice properties as normal default theories. In order to study the connection between the two notions: auto-compatibility and FC-normality, we extend auto-compatible default theories to weakly auto-compatible default theories (WACDT) and discover that this subclass has all desired nice properties of normal default theories. We also explore some new features of FC-normal non-monotonic rule systems (FC-normal NRS) and we show that every FC-normal default theory is weakly auto-compatible, but not vice versa. It will not only theoretically fertilizes the two notions of auto-compatibility and FC-normality but also, or more importantly, the notions

*The work was partially supported by the Natural Science Foundation under grant NSF 60573009 and the stadholder foundation of Guizhou Province under grant 2005(212).

can be easily applied to general logic programs and truth maintenance systems and then it will surely benefit both of them.

The outline of this paper is as follows. In section 2 we briefly recall some notations of Reiter's DL, Lukaszewic's modified extension and some key properties of auto-compatible default theories. Then in section 3 we explore some new features of FC-normal nonm-onotonic rule systems. In section 4, we establish the relationship between the class of FC-normal default theories and that of weakly auto-compatible default theories. For the sake of space, we leave the proofs of some main propositions at the end of the paper as appendix.

2 Preliminaries

Following the notations in (Reiter 1980) with slight difference, a default is a rule of the form

$$\alpha : \beta_1, \ldots, \beta_n/\gamma \tag{1}$$

where $\alpha, \beta_i (1 \leq i \leq n)$ and γ are wffs in a underlying propositional language $\mathcal{L}$. If $\alpha = true$, it is called prerequisite-free and it is usually written as : $\beta_1, \ldots, \beta_n/\gamma$. If n=0, it is called justification-free and is written as $\alpha : /\gamma$. A default theory (DT) is a pair (D, W), where D is a set of defaults and W a set of formulas. A default is *normal* if it is of the form $\alpha : \beta/\beta$. A default is *semi-normal* if it is of the form $\alpha : \beta \wedge \gamma/\gamma$. A DT (D, W) is normal (semi-normal) if every default $d \in D$ is normal (semi-normal). Reiter gave the fixed-point definition and quasi-inductive characterization of extensions for a DT. These definitions are based on an infinite and deductively closed set of formulas.

As we know, there are some default theories which have no extension. To avoid this, (Lukaszewicz 1988) defined a new version of application for defaults by employing two operators, whose roles are to keep tracks of the consequents and the consistent conditions of being applied defaults respectively. That is, let (E, F) and (E', F') be pairs of sets of formulas. A default $d = \alpha : \beta_1, \ldots, \beta_n/\gamma$ is *applicable* to (E', F') w.r.t. (E, F), denoted $d^{\nabla}_{(E,F)}(E', F')$, whenever if $\alpha \in E'$ and $E \cup \{\gamma\} \models \neg\beta$ for no $\beta \in F \cup \{\beta_1, \ldots, \beta_n\}$, then $\gamma \in E'$ and $\{\beta_1, \ldots, \beta_n\} \subseteq F'$. Let $\Delta = (D, W)$ be a DT, E and F sets of formulae. Define $\Lambda^1_\Delta(E, F)$ and $\Lambda^2_\Delta(E, F)$ to be the smallest set of formulae such that $\Lambda^1_\Delta(E, F)$ is deductively closed, $W \subseteq \Lambda^1_\Delta(E, F)$, and if $d \in D$ then $d^{\nabla}_{(E,F)}(\Lambda^1_\Delta(E, F), \Lambda^2_\Delta(E, F))$. Then a set of formulae is a *modified extension* of Δ iff there exists a set F of formulae such that (E, F) is a fixed point of a certain operator Λ^{∇}.

We gave a characterization of extensions of a DT, which is based only on the DT itself. To do this, we introduced the notions of (joint) compatibility for a set of defaults and the operator Λ, which characterize the conditions of applicability of defaults. In the same way, we also got the finite characterizations of extensions for DL's variants. Now we recall some notations and results in (Mingyi 1992; 1992; 1994).

2.1 Reiter's DL and Lukaszewicz's extensions

Definition 1 *Let D be a set of defaults. We use the following notations:*
Pre(D)=$\{\alpha | \alpha : \beta_1, \ldots, \beta_n/\gamma \in D\}$,
Ccs(D)=$\{\beta_i | \alpha : \beta_1, \ldots, \beta_n/\gamma \in D, 1 \leq i \leq n\}$ *and,*
Cns(D)=$\{\gamma | \alpha : \beta_1, \ldots, \beta_n/\gamma \in D\}$.

Here, to avoid confusion with the notation *Con* of consistency property in (Marek, Nerode, & Remmel 1994), we replace the notation *Con* in (Mingyi 1992) with *Cns*.

Different from our previous work, we allow a default to be justification-free in this paper. It is easy to see that all of the results previously obtained still are true. A few results different from our one before will be reiterated and some special results about DT with justification-free will be given also.

A default d, as a inference rule, is monotonic if $Ccs(d) = \emptyset$; otherwise, it is non-monotonic. For any set D of defaults we denote $D_M = \{d \in D | Ccs(d) = \emptyset\}$, $D_{NM} = \{d \in D | Ccs(d) \neq \emptyset\}$.

To capture the consistency condition of generating an extension for a default theory, we introduced the notions of compatibility and auto-compatibility (Mingyi 1992), which are still well defined when we allow a default to be justification-free.

Definition 2 *Let $\Delta = (D, W)$ be a DT. Any $D' \subseteq D$ is said to be compatible with respect to (w.r.t.) Δ if $W \cup Cns(D') \nvdash \neg\beta$ for each $\beta \in Ccs(D')$. D' is maximally compatible if it is compatible and there is no compatible subset D'' of D which properly contains D'.*

Note that the empty set $\emptyset$ is compatible w.r.t. any default theory. Further, D_M is compatible for any DT $\Delta = (D, W)$.

Definition 3 *Let $\Delta = (D, W)$ be a DT and D' a compatible subset of D. A default $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma)$ is auto-incompatible w.r.t. D' if*

(1) $W \cup Cns(D') \nvdash \neg\beta_i$ for any $1 \leq i \leq n$ and,
(2) $W \cup Cns(D' \cup \{d\}) \vdash \neg\beta$ for some $\beta \in Ccs(D' \cup \{d\})$ (i.e. $D' \cup \{d\}$ is incompatible).

d is auto-incompatible w.r.t. Δ if there is a compatible subset D' of D such that d is auto-incompatible w.r.t. D'. It is auto-compatible w.r.t. a compatible subset D' of D if it is not auto-incompatible w.r.t. D', and it is auto-compatible w.r.t. Δ if it is auto-compatible w.r.t. any compatible subset D' of D. Δ is auto-compatible if every default of D is auto-compatible w.r.t. Δ.

Clearly, for a DT $\Delta = (D, W)$, if D is compatible then Δ is auto-compatible. But the inverse is not true. For example, the DT $(\{: A/B; \quad : \neg B/C\}, \emptyset)$ is auto-compatible but D is not compatible.

The following operator is to characterize the derivability of premises of defaults generating an extension.

Definition 4 *Let $\Delta = (D, W)$ be a DT. The operator $\Lambda : 2^D \to 2^D$ (the power set of D) is defined as: for any $D' \subseteq D$, $\Lambda(D', \Delta) = \bigcup_{\eta \in \mu} D'_\eta(\Delta)$, where μ is the ordinal of D (Assume that the ordering among ordinals is given by $\in$) and*

- $D'_0(\Delta) = \{d \in D' | W \vdash Pre(\{d\})\}$;

- $D'_{\eta+1}(\Delta) = \{d \in D' | W \cup Cns(D'_\eta(\Delta)) \vdash Pre(\{d\})\}$ *if η is a successive ordinal;*
- $D'_\eta(\Delta) = \bigcup_{\kappa\in\eta} D'_\kappa(\Delta)$, *if η is a limit ordinal.*

Essentially, an extension of a default theory is determined by its applicable defaults, which is called the set of generating defaults (Reiter 1980).

Definition 5 *Let $\Delta = (D, W)$ be a DT and E an extension of Δ. The set of generating defaults of E, $GD(E, \Delta)$, is the set $\{\alpha : \beta_1, \ldots, \beta_n/\gamma \in D | \alpha \in E, \neg\beta_1, \ldots \neg\beta_n \notin E\}$.*

Obviously, if *E* is an extension of a DT $\Delta = (D, W)$ then $E = Th(W \cup Cns(GD(E, \Delta))$, where Th is the deductive closure operator in classical logic.

Definition 6 *Let $\Delta = (D, W)$ be a DT, D' a subset of D. D' is strongly compatible w.r.t. Δ if D' is compatible and $\Lambda(D', \Delta) = D'$.*

In the following sections we will omit "w.r.t. $D'(\Delta)$" whenever it is not confused from the context. we presented the important features of compatibility concept and the operator Λ (Mingyi 1992).

Theorem 1 *Let $\Delta = (D, W)$ be a DT. For any $D' \subseteq D'' \subseteq D$,*
(1) if D'' is compatible then also is D'.
(2) if $\Lambda(D', \Delta) = D'$ then for any $d \in D$, $\Lambda(D' \cup \{d\}, \Delta) = D' \cup \{d\}$ iff $W \cup Cns(D') \vdash Pre(\{d\})$.

We say that a strongly compatible subset D' of *D* is maximal if there is no strongly compatible subset D'' of *D* such that $D' \subset D''$ (here $\subset$ implies $\subseteq$ and $\neq$). Clearly, for any DT =(D,W), $\emptyset$ is strongly compatible. Let $SC(\Delta) = \{D' | D' \subseteq D$ and D' is strongly compatible$\}$ and $MSC(\Delta) = \{D' | D' \subseteq D$ and D' is maximally strongly compatible$\}$. $SC(\Delta)$ is not empty since $\emptyset$ is strongly compatible. By Zorn's lemma, we have

Corollary 2 *Each default theory has a maximally strongly compatible set of defaults.*

Theorem 3 *If a DT $\Delta = (D, W)$ has an extension E then $GD(E, \Delta)$ is maximally strongly compatible.*

2.2 Finite Characterization of extensions

Based on the notions of compatibility and the operator Λ we presented a finite characterization of extensions, which enables us to determine whether a default theory has an extension by checking the default theory itself and to compute one extension if it exits.

Theorem 4 *(Finite characterization of DL extensions)*
A DT $\Delta = (D, W)$ has an extension iff there exists a compatible subset D' of D such that
P_1. $\Lambda(D', \Delta) = D'$;
P_2. For any $\alpha : \beta_1, \ldots, \beta_n/\gamma \in D - D'$, either $W \cup Cns(D') \nvdash \alpha$ or $W \cup Cns(D') \vdash \neg\beta_i$ for some $1 \leq i \leq n$.

In other words, a DT $\Delta = (D, W)$ has an extension iff there exists a maximally strongly compatible subset D' of D such that d is auto-compatible w.r.t. D' for any $d \in D - D'$.

From the above theorem it is immediate that the set of generating defaults of an extension for Δ is a maximally strongly compatible subset of defaults.

It is worthy of noting that, Marek and Truszczynski (Marek & Truszczynski 1993) independently obtained a characterization similar to the above theorem, which is slightly different from ours. First, the notion of compatibility carries more information about the existence of extensions than *S*-provability. For example, consider the default theory $(\{: A/\neg A\}, \emptyset)$. Let $S = \emptyset$. $: A/\neg A$ is *S*-applicable but not compatible. This can be also seen from the development of the class of auto-compatible default theories and the application to characterize Lukaszewicz's modified extensions. Next, the "overlap" caused by the operator R^{D_s} in (Marek & Truszczynski 1993) is global, while the "overlap" carried by the operator Λ is local. This property of "localization" of a "overlap" makes it simple to find extensions, i.e. verifying a candidate extension by our algorithm, in general, needs much less logical inference tests than by their algorithm.

From the above characterization we can get some sufficient conditions for the existence of extension, for instance,

Theorem 5 *Let $\Delta = (D, W)$ be a DT. If D is compatible (further if $\Lambda(D, \Delta)$ is compatible) then Δ has exactly one extension $E = Th(W \cup Cns(\Lambda(D, \Delta)))$.*

Theorem 6 *If a DT $\Delta = (D, W)$ is auto-compatible then it has an extension.*

The notion of strongly compatibility can also depict the characterization of modified extension.

Theorem 7 *Any default theory $\Delta = (D, W)$ has an modified extension which is generated by a maximally strong compatible subset of D.*

It is worth to see that Reiter's conclusion on inconsistent extension does not hold for a DT with justification-free defaults. In fact we have

Theorem 8 *Any DT $\Delta = (D, W)$ has an inconsistent extension iff $W \cup Cns(\Lambda(D_M, \Delta))$ is inconsistent.*

Proof: It is clear from the fact that $\Lambda(D_M, \Delta)$ is the unique subset of D satisfying the conditions in Theorem 4. ■

Corollary 9 *If a DT $\Delta = (D, W)$ has an inconsistent extension E then this is its only extension and $GD(E, \Delta) = \Lambda(D_M, \Delta)$.*

Without loss of generality, we always assume that $W \cup Cns(\Lambda(D_M, \Delta))$ is consistent in the rest of the paper unless otherwise stated.

3 Weakly Auto-Compatible Default Theory

3.1 Weak auto-compatibility

From the finite characterization of extensions, Theorem 4, it is easy to see that violating the condition P_2 (i.e. there is an auto-incompatible default) might lead to nonexistence of extension for a default theory. Using the notion of auto-compatibility is an approach to avoiding this problem. Extending the notion of auto-compatibility, in this section, we present a larger class than auto-compatible DTs. This class, which we call Weakly Auto-Compatible Default Theory, enjoys many nice properties as the class of

auto-compatible default theories. All proofs in the section are easily done in a way similar to those in (Mingyi 1992; 1994).

Definition 7 *Let $\Delta = (D, W)$ be a DT, D' a strongly compatible subset of D and let $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma)$ be a default in D. d is weakly auto-compatible w.r.t. D' if $\Lambda(D' \cup \{d\}, \Delta) = D' \cup \{d\}$ implies that d is auto-compatible w.r.t. D'. d is weakly auto-compatible w.r.t. Δ if d is weakly auto-compatible w.r.t. any strongly compatible subset of D. Δ is weakly auto-compatible if each of its defaults is weakly auto-compatible w.r.t. Δ.*

From the definition and Definition 3 it is clear that the class of auto-compatible default theories is a subclass of weakly auto-compatible theories. That is

Corollary 10 *Each auto-compatible default theory is also weakly auto-compatible.*

The following example shows that the inverse of the above result is false and then auto-compatible default theory class is a proper subclass of the weakly auto-compatible default theory.

Example 1 The default theory $\Delta = (\{: A/B;\ C : D/\neg A\}, \emptyset)$ is not auto-compatible but weakly auto-compatible since the default $: A/B$ is auto-incompatible w.r.t. the compatible set of defaults $\{C : D/\neg A\}$, which is not strongly compatible.

So, we have

Normal default theories $\subset$ auto-compatible default theories $\subset$ weakly auto-compatible default theories.

Noting that weak auto-compatibility just actually prevents generation of an extension from violating condition P_2 in Theorem 4. It is obvious that, for a weakly auto-compatible default theory $\Delta = (D, W)$, each maximally strongly compatible subset of D is just the set of generating defaults of an extension for Δ. In fact, for any maximally strongly compatible subset D' of D (its existence is guaranteed by Corollary 2), $E' = Th(W \cup Cns(D'))$ is just one extension of $\Delta' = (D', W)$ by Theorem 5. Further, $GD(E', \Delta') = D'$. Clearly, D' satisfies the conditions in Theorem 4 for Δ. So, D' is the set of generating defaults of an extension for Δ and E' is an extension of Δ. This shows that existence of extensions for a weakly auto-compatible DT is guaranteed. That is,

Theorem 11 *A weakly auto-compatible DT $\Delta = (D, W)$ has at least one extension and each $D' \in MSC(\Delta)$ corresponds to a set of generating defaults of an extension E of Δ.*

The above analysis on the existence of extensions does also hold for any strongly compatible subset D' of D. The only modification is extending D' to a maximally strongly compatible subset and generating an extension E for Δ such that $E' \subseteq E$, where E' is an extension for (D', W). This shows that weak auto-compatibility implies *semi-monotonicity* in a sense. Here, a default theory (D, W) enjoys semi-monotonicity if it satisfies the following condition: for any D' and D'' with $D' \subseteq D'' \subseteq D$, (D'', W) has an extension E'' such that $E' \subseteq E''$ whenever (D', W) has an extension E'. Now we state the result, whose proof is simple by Theorem 5.

Theorem 12 *(Semi-monotonicity) Suppose that $\Delta = (D, W)$ is a weakly auto-compatible default theory and that D' is any subset of D. If $\Delta' = (D', W)$ has an extension E', then Δ has an extension E such that $E' \subseteq E$ and $GD(E', \Delta') \subseteq GD(E, \Delta)$.*

Proof: Clearly, from Theorem 11, Δ' is weakly auto-compatible and then $GD(E', \Delta') \in MSC(\Delta')$. Further there exists $D'' \subseteq D$ such that $D'' \in MSC(\Delta)$ and $GD(E', \Delta') \subseteq D''$. Consequently, $E = Th(W \cup Cns(D''))$ is one extension of Δ and obviously, $E' \subseteq E$ and $GD(E', \Delta') \subseteq GD(E, \Delta)$. ■

For a prerequisite-free DT $\Delta = (D, W)$, the inverse of the above theorem is also true. In fact, we pointed out that semi-monotonicity is an essential characterization for a prerequisite-free default theory (Mingyi 1996). Note that weakly auto-compatibility is equivalent to auto-compatibility for any prerequisite-free default theory. So, we have

Theorem 13 *A prerequisite-free DT is weakly auto-compatible iff it enjoys semi-monotonicity.*

As a matter of fact, it is not difficult to prove that default theory is weakly auto-compatible whenever it is semi-monotonic.

Lemma 1 *Let $\Delta = (D, W)$ is a weakly auto-compatible default theory. Then $\Delta' = (D', W)$ is also weakly auto-compatible, where $D' \subseteq D$.*

Theorem 14 *(Underlying Characterization of Weakly Auto-compatibility DT) A default theory Δ is weakly auto-compatible if and only if it is semi-monotonic.*

Proof: "only if " It is clear from Theorem 12.

"if" Suppose that Δ is not weakly auto-compatible, then we have $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma) \in D$ and a strongly compatible subset $D' \subseteq D$ such that $W \cup Cns(D') \vdash \alpha$, $W \cup Cns(D') \nvdash \neg\beta_i$ for any $i : 1 \leq i \leq n$ and $D' \cup \{d\}$ is not compatible. Notice that (D', W) has a unique extension $E' = Th(W \cup Cns(D'))$ and $GD(E', \Delta') = D'$. By the semi-monotonicity of Δ, it follows that Δ has one extension E'' such that $E' \subseteq E''$ and $D' \subseteq GD(E'', \Delta)$. Clearly, $d \notin D \backslash GD(E'', \Delta)$ since $D' \cup \{d\}$ is not compatible by the assumption. However, by Theorem 3, it conflicts with the assumption that $W \cup Cns(D') \vdash \alpha$ and $W \cup Cns(D') \nvdash \neg\beta_i$ for any $i : 1 \leq i \leq n$. ■

To check strongly compatibility of a subset of *D* for a given weakly auto-compatible DT, starting form the empty set of defaults, we need the following lemma, whose proof is easy from the definition of strongly compatibility and Theorem 1.

Lemma 2 *Suppose that $\Delta = (D, W)$ is weakly auto-compatible. For any strongly compatible $D' \subseteq D$ and any $d \in D$, if $W \cup Cns(D') \vdash Pre(\{d\})$ and $W \cup Cns(D') \nvdash \beta$ for each $\beta \in Ccs(\{d\})$, then $D' \cup \{d\}$ is strongly compatible.*

The above result shows that we can construct any maximally strongly compatible subset of *D* starting from the empty set of defaults. Together with Theorem 11, this lemma also implies the algorithm to compute all of extensions for a given finite weakly auto-compatible default theories — all of its maximally strongly compatible defaults are enough.

Algorithm 1 *Given a finite default theory $\Delta = (D, W)$ and $D' \subseteq D$, compute the $\Lambda(D', \Delta)$.*
function *LAMBDA*(D, W, D')
begin
  $result := \emptyset$
  repeat
    $new := \emptyset$
    for each $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma) \in D' - result$ ***do***
      if $W \cup Cns(result) \vdash Pre(d)$ ***then***
        $new := new \cup \{d\}$
    $result := result \cup new$
  until $new = \emptyset$
  return*(result)*
end

It is obvious that *LAMBDA*(D, W, D') will correctly compute $\Lambda(D', \Delta)$ in $O(|D'|^2)$ costs. Please note that, hereafter, we regards $\vdash$ as one unit time cost.

Algorithm 2 *Given a finite default theory $\Delta = (D, W)$, determine whether $D' \subseteq D$ is maximally strongly compatible.*
boolean function *isMSC*(D, W, D')
begin
  for each $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma) \in D'$ ***do***
    if $W \cup Cns(D') \vdash \neg\beta_i$ *for some* $1 \leq i \leq n$ ***then***
      return*(false)*
  if *LAMBDA*$(D, W, D') \neq D'$ ***then***
    return*(false)*
  for each $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma) \in D - D'$ ***do***
    if $W \cup Cns(D') \vdash Pre(d)$ ***and***
      $W \cup Cns(D') \nvdash \neg\beta_i$ *for any* $i (1 \leq i \leq n)$ ***then***
      return*(false)*
  return*(true)*
end

This algorithm is to check whether D' is maximally strongly compatible. And it can be implied in $O(|D|^2)$.

Algorithm 3 *Given a finite WAC default theory $\Delta = (D, W)$, compute the maximally strongly compatible sets of defaults.*
function *allMSC*(D, W)
begin
  $result := \emptyset$
  for each $D' \subseteq D$ ***do***
    if *isMSC*(D, W, D') ***then*** $new := new \cup \{D'\}$
  $result := result \cup new$
  return*(result)*
end

Undoubtedly, by Theorem 11, for a given WAC default theory, we can compute all of its extensions by just find all of its maximally strongly compatible sets of defaults. That is what the above algorithm does. The soundness and completeness of this algorithm are clear, and then we ignore its proof.

In order to compute one extension of a finite WAC default theory $\Delta = (D, W)$, we just need to compute one maximally strongly compatible set of defaults. By Lemma 2, this can be achieved by giving a well-ordering $<$ over D and starting from $\emptyset$ as the next algorithm.

Algorithm 4 *Given a finite WAC default theory $\Delta = (D, W)$, to compute one extension for Δ.*
function *one-extension*$(D, W, <)$
begin
  $E^< := Th(W)$; $GD^< := \emptyset$
  repeat
  $new := \emptyset$
    if *there exists the least k such that*
      $d_k^< \in \{d_m^< | m \leq |D|\} - GD^<$, *where*
      $Pre(d_k^<) \in E^<$ *and* $\neg\beta \notin E^<$ *for any*
      $\beta \in Ccs(d_k^<)$ ***then***
      begin
        new :=new $\cup \{d_k\}$
        $E^< := Th(E^< \cup Cns(d_k^<))$
        $GD^< := GD^< \cup \{d_k^<\}$
      end
  until $new = \emptyset$
  return$(E^<)$
end

Theorem 15 *(Soundness and Completeness) Given a finite WAC default theory $\Delta = (D, W)$, we have*
(1) For any $D' \in$ allMSC(D, W), $Th(W \cup Cns(D'))$ is an extension of Δ.
(2) For each extension E of Δ, $GD(E, \Delta)$ must be in allMSC(D, W).

However, notice that, for FC-normal default theory, the Forward Chaining Construction algorithm is very similar to our algorithm and then it is also exhaustive when comes to computing all extensions although it is quite easy to just compute one extension. In the next section, we show that FC-normal default theories are weakly auto-compatible, and then this algorithm suits for FC-normal default theory without a given consistency property beforehand like FC-normal construction.

Also note that, the complexity of checking whether a default theory is WACDT is still open up to now like FC-normal default theory. We strongly conjecture that it is same to the complexity of computing all extensions of any default theory. So, it is interesting to find some subclasses of WACDT, which have simpler criterion which enables us estimating weak auto-compatibility.

Based on Theorem 11 we can get the following nice properties of weakly auto-compatible default theories, which are similar to that of normal default theories.

P_1. Updating a weakly auto-compatible default theory with new defaults cannot effect old belief provided the resulted default theory is still weakly auto-compatible.

P_2. For a given weakly auto-compatible default theory there is a proof procedure, which is local w.r.t. the defaults

so that the proofs can be constructed by ignoring some of defaults.

4 Weak Auto-Compatibility and FC-normality

In this section we establish a closed relationship between auto-compatibility and FC-normality. By analyzing the proofs of the main results on normal default theories, Marek *et al* revealed that the proof does not rely on the particular syntactic form of the rules but rather on the fact that all rules have a certain consistency property (Marek, Nerode, & Remmel 1994). They extended the notion of normal default theories and proposed so-called Forward Chaining (FC) normal default theories. They proved FC non-monotonic rule systems (NRS) have many desirable properties of normal default theories and a number of other important properties as well. It is worth to note that the notion and desired properties of FC-normal default theories are very similar to the ones of (weakly) auto-compatible default theories. This lead us to explore the relationship between these two classes of default theories. At first we recall some notations of FC-normality.

4.1 FC-normality

Definition 8 *A non-monotonic rule of inference r is an expression of the form*

$$\alpha_1, \ldots, \alpha_m : \beta_1, \ldots, \beta_n/\gamma. \quad (2)$$

For such r we denote the premises of r by $Pre(r) = \{\alpha_1, \ldots, \alpha_m\}$*, the constraints of r by* $Cons(r) = \{\beta_1, \ldots, \beta_n\}$*, and the conclusion of r by* $Cns(r) = \{\gamma\}$ *respectively. Either* $Pre(r)$*, or* $Ccs(r)$*, or both may be empty. If* $Pre(r) = Ccs(r) = \emptyset$*, then the rule r is called an axiom. A non-monotonic formal system is a pair* (U, N)*, where U is a nonempty set and N is a set of non-monotonic rules such that Pre(r), Cons(r) and Cns(r) are subset of U for all* $r \in N$*. A subset* $S \subseteq U$ *is called deductively closed if for all* $r \in N$*, whenever all the premises of r are in S and all the constraints of r are not in S, then the conclusion of r belongs to S.*

Given $S \subseteq U$ and $I \subseteq U$, an *S-deduction* of γ from *I* in *(U,N)* is a finite sequence $< \gamma_1, \ldots, \gamma_k >$ such that $\gamma_k = \gamma$ and for all $i \leq k$, each γ_i is in *I*, or is an axiom, or is the conclusion of a rule $r \in N$ such that $Pre(r) \subseteq \{\gamma_1, \ldots, \gamma_{i-1}\}$ and $Cons(r) \subseteq U - S$. An *S-consequence* of *I* is an element of *U* occurring in some *S*-deduction from *I*. Let $C_S(I)$ be the set of all *S*-consequences of *I* in *(U,N)*.

We say that $S \subseteq U$ is *ground* in *I* if $S \subseteq C_S(I)$. We say that $S \subseteq U$ is an *extension* of *I* if $C_S(I) = S$. *T* is an *extension* of $\Gamma = (U, N)$ if *T* is an extension of $\emptyset$ in *(U,N)*. Let $NG(S, \Gamma) = \{r \in N | Pre(r) \subseteq S$ and $Cons(r) \cap S = \emptyset\}$. Then $NG(T, \Gamma)$ is the set of generating non-monotonic rules of *T* if *T* is an extension of *(U,N)*.

Note that a monotonic rule system (MRS) is a special case of NRS, where every non-monotonic rule has no constraints. So, the above notions can be easily translated into the case that *(U,N)* (usually, written *(U,M)*) is a MRS.

A *proof scheme* of *r* is a finite sequence, $p =<< \gamma_0, r_0, G_0 >, ..., < \gamma_m, r_m, G_m >>$ such that
(1) If m = 0 then (a) γ_0 is a conclusion of an axiom *r*, $r_0 = r$ and $G_0 = \emptyset$, or (b) $\{\gamma_0\} = Cns(r)$, $r_0 = r$ and $G_0 = Cons(r)$, where $r = (: \beta_1, \ldots, \beta_n/\gamma) \in N$.
(2) If $m > 0$, $<< \gamma_0, r_0, G_0 >, \ldots, < \gamma_{m-1}, r_{m-1}, G_{m-1} >>$ is a proof scheme of length *m* and γ_m is a conclusion of *r*, where $Pre(r) \subseteq \{\gamma_0, \ldots, \gamma_{m-1}\}$, $r_m = r$ and $G_m = G_{m-1} \cup Cons(r)$. The formula γ_m is called the conclusion of *p* and is written $clm(p)$.

For a non-monotonic rule system $\Gamma = (U, N)$, let $mon(\Gamma) = \{r \in N | Cons(r) = \emptyset\}$, $nmon(\Gamma) = N - mon(\Gamma)$. We say a set $S \subseteq U$ is *monotonically closed* if whenever $r = \alpha_1, \ldots, \alpha_m : /\gamma \in mon(\Gamma)$ and $\alpha_1, \ldots, \alpha_m \in S$, then $\gamma \in S$. Given any set $A \subseteq U$, the monotonic-closure of A, written $cl_{mon}(A)$, is defined to be the intersection of all monotonically closed sets containing A.

Definition 9 *Let* $\Gamma = (U, N)$ *be a non-monotonic rule system. We say that* $Con \subseteq 2^U$ *is a consistency property over* Γ *if*
1. $\emptyset \in Con$*;*
2. for all $A, B \subseteq U (A \subseteq B \& B \in Con \Rightarrow A \in Con)$*;*
3. for all $A \in U (A \in Con \Rightarrow cl_{mon}(A) \in Con)$*;*
4. whenever $\Omega \subseteq Con$ *has the property that* $A, B \in \Omega \Rightarrow \exists C \in \Omega (A \subseteq C \& B \subseteq C)$ *then* $\bigcup \Omega \in Con$*.*

Note that the conditions 1,2 and 4 are Scott's conditions for information systems. Let $\Gamma = (U, N)$, A rule $r = (\alpha_1, \ldots, \alpha_m : \beta_1, \ldots, \beta_n/\gamma) \in nmon(\Gamma)$ is *FC-normal* w.r.t. a consistency property *Con* over $\Gamma = (U, N)$ if $V \cup \{\gamma\} \in Con$ and $V \cup \{\gamma, \beta_i\} \notin Con$ for all $i : 1 \leq i \leq n$ whenever $V \subseteq U$ is such that $V \in Con$, $cl_{mon}(V) = V$, $\alpha_1, \ldots, \alpha_m \in V$, and $\beta_1, \ldots, \beta_n, \gamma \notin V$. We say that Γ is *FC-normal* w.r.t. *Con* if, for every $r \in nmon(\Gamma)$, r is FC-normal w.r.t. *Con*. Finally Γ is *FC-normal* if it is FC-normal w.r.t. some consistency property $Con \subseteq 2^U$.

Marek *et al* proved that any FC-normal NRS has an extension and gave a uniform construction of extensions (Marek, Nerode, & Remmel 1994). Assume $<$ is a well-ordering of $nmon(\Gamma)$, which determines some listing of the rules of $nmon(\Gamma)$, $\{r_\alpha | \alpha \in \kappa\}$, where κ is some ordinal. Let Z_α be the least cardinal such that $\kappa < Z_\alpha$. They defined the forward chaining construction (FC construction) of extensions.

Example 2 (Marek, Nerode, & Remmel 1994) Let $U = \{a, b, c, d, e, f\}$. Consider the set of rules $N = \{: /a; \quad c : /b; \quad b : /c; \quad a : d/c; \quad c : f/e\}$. Then for the NRS *(U,N)*, subsets $2^{\{a,b,c,e\}}$ and $2^{\{a,b,c,e\}} \cup 2^{\{a,b,c,f\}}$ of 2^U are consistency properties over *(U,N)*. It is easy to check that *(U, N)* is FC-normal w.r.t. each of these subsets respectively. Clearly $\{a, b, c, e\}$ is the unique extension. Let *Con* be defined by the condition: $A \notin Con$ iff either $\{c, d\} \subseteq A$ or $\{e, f\} \subseteq A$. Obviously, *Con* is not a consistency property since $\{a, b, d, e\}, \{a, b, d, f\} \in Con$ but we have that $cl_{mon}(\{a, b, d, e\}) = \{a, b, c, d, e\} \notin Con$, $cl_{mon}(\{a, b, d, f\}) = \{a, b, c, d, f\} \notin Con$.

4.2 Properties of FC-normal NRS

Now we explore some new features of FC-normal non-monotonic rule systems. From the definitions of a consis-

tency property and FC-normality the following lemma is clear.

Lemma 3 *Let (U, N) be a NRS.*
(1) If Con is a consistency property over (U, N), then $Con = 2^U$ *iff* $U \in Con$.
(2) 2^U *(*$2^{cl_{mon}(\emptyset)}$*, resp.) is the maximal (minimal, resp.) consistency properties over (U, N), i.e.* $2^{cl_{mon}(\emptyset)} \subseteq Con \subseteq 2^U$ *for any consistency Con over (U, N).*
(3) Given any consistency properties Con_1 *and* Con_2 *over (U, N),* $Con_1 \cap Con_2$ *is also a consistency property over (U, N). Further, if (U,N) is FC-normal w.r.t.* Con_1 *and* Con_2 *respectively then it is also FC-normal w.r.t.* $Con_1 \cap Con_2$.
(4) If (U,N) is FC-normal w.r.t. a consistency property Con then there is a minimal consistency property Con^* *such that* $Con^* \subseteq Con$ *and (U,N) is FC-normal w.r.t.* Con^*.

Theorem 16 *If* $\Gamma = (U, N)$ *is FC-normal w.r.t. a consistency property Con, then* $Con^* = \bigcup\{2^E |$ *E is an extension of* $\Gamma\} \subseteq Con$.

Proof: By the completeness of FC-construction, every extension E of Γ is of the form $E^< = \bigcup\{E^<_\alpha | \alpha \in Z_\alpha\}$ for a suitably chosen well-ordering $<$ over $nmon(\Gamma)$ and $E^< \in Con$. So, $Con^* \subseteq Con$. ■

It is clear that, for a given default theory $\Delta = (D, W)$, it is easy to translate Δ into a non-monotonic rule system (U, N) and vice versa. That is, a default rule $\alpha : \beta_1, \ldots, \beta_n/\gamma$ and an element w of W are transformed into $\alpha : \neg\beta_1, \ldots, \neg\beta_n/\gamma$ and $: /w$ respectively, and U be the set of all formulas of the underlying language. Then the notion of forward chaining normal is applicable for default theory in the sense that $cl_{mon}(A) = Th(W \cup Cns(\Lambda(D_M, \Delta)) \cup A)$ where A is a set of formulas of the underlying language.

Definition 10 *(Marek, Nerode, & Remmel 1994) Given a consistency property Con and a default theory* $\Delta = (D, W)$*, we say that a default rule* $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma)$ *in* D_{NM} *is FC-normal w.r.t. Con if* $T \cup \{\gamma\} \in Con$ *and* $T \cup \{\gamma, \neg\beta_i\} \notin Con$ *for any* $i : 1 \leq i \leq n$ *whenever T is a theory such that* $T \in Con$, $cl_{mon}(T) = T$, $\alpha \in T$ *and* $\neg\beta_1, \ldots, \neg\beta_n, \gamma \notin T$. Δ *is FC-normal w.r.t. Con if each default in* D_{NM} *is FC-normal w.r.t. Con.* Δ *is FC-normal if it is FC-normal w.r.t. some consistency property over* (D, W).

From Lemma 3, given an FC-normal default theory $\Delta = (D, W)$, we asserts that there is a minimal consistency property Con^* such that Δ is FC-normal w.r.t Con^*. However, to our best knowledge, neither does there exist feasible algorithm to compute the minimal consistency property even thought we know it is FC-normal, nor does there exist feasible algorithm to decide whether a given default theory is FC-normal or not. Because only from the definition of FC-normality, there is no suggestive approach to exhaust all possible consistency properties over the given default theory. The next corollary asserts that, for a candidate minimal consistency property over a given default theory, it is not enough to just consider its extensions.

Corollary 17 *There is an FC-normal DT such that* $Con^* \subset Con$.

Example 3 Consider the default theory $\Delta = (\{: A/B; \quad : C/F \wedge G; \quad : \neg F/\neg C\}, \emptyset)$, where *A, B, C, F* and *G* are atoms. Δ is weakly auto-compatible and has two extensions $Th(\{B, F, G\})$ and $Th(\{B, \neg C\})$. If Δ is FC-normal w.r.t. a consistency property *Con*, then $\{B, G, \neg C\} \in Con$ (In fact, $Th(\{B, G\}) \in Con$ and $\neg C, F \notin Th(\{B, G\})$. It is immediate from the FC-normality of the default $: \neg F/\neg C$). Clearly, there is no any compatible subset D' of *D* such that $\{B, G, \neg C\} \subseteq Th(Cns(D'))$. By FC-normality, $\{T \subseteq E |$ *E* is an extension of $\Delta\} \subseteq Con$ for any consistency property *Con*. This example shows that $Con^* \subset Con$ is possible, where $Con^* = \{T \subseteq E |$ *E* is an extension of $\Delta\}$. However, we can check that Δ is FC-normal w.r.t. $Con = 2^{Th(\{B,F,G\})} \cup 2^{Th(\{B,G,\neg C\})}$ though the procedure is a bit complex.

4.3 Features of consistency property

Marek *et al* showed that their notion of FC-normal default theories actually extend Reiter's original notion of normal default theories and got nice properties for FC-normal default theories (Marek, Nerode, & Remmel 1994), which are similar to those of normal default theories, such as the existence of extensions, semi-normality and a proof theory *etc*. After we establish the relationship between the notions of (weak) auto-compatibility and FC-normality, these become clear from our results on weakly auto-compatible default theory. To do this, we give the following results.

Theorem 18 *Given a default theory (D,W), let Con be a consistency property for (D,W).*
(1) T is consistent for all $T \in Con$ *iff* $U \notin Con$ *iff* $Con \subset 2^U$.
(2) If W is inconsistent, then $Con = 2^U$.
(3) If $Con = 2^U$ *then any default* $d = (\alpha : \beta_1, \ldots, \beta_n/\gamma) \in D$ *is not FC-normal w.r.t. Con unless* $\neg\beta_1 = \ldots \neg\beta_n = \gamma = \alpha$.

Proof: (1) It is clear.
(2) If *W* is inconsistent, then $Th(W) = U \in Con$ since $\emptyset \in Con$. This implies that $Con = 2^U$
(3) If $Con = 2^U$, then $\{\alpha\} \in Con$ and $\neg\beta_1, \ldots, \neg\beta_n, \gamma \notin \{\alpha\}$ implies that $Th(W \cup \{\gamma \cup \neg\beta_i\}) \notin Con$ for all $i : 1 \leq i \leq n$, which contradicts $Con = 2^U$. ■

Theorem 19 *Let (D, W) be a DT, where W is consistent. The collection of all consistent sets,* $\{A \subseteq U |$ *A is consistent*$\}$ *(denoted it by* $Con^\#$*), is a consistency property for (D, W). Further,* $Con \subseteq Con^\#$*, whenever Con is a consistency property for (D, W) such that* $Con \subset 2^U$.

Proof: It is sufficient to show that $Con^\#$ is closed under the union of directed family (condition 4 of Definition 9). Assume that $\Omega \subseteq Con^\#$ has the property that $A, B \in \Omega \Rightarrow \exists C \in \Omega(A \subseteq C \& B \subseteq C)$. If $\bigcup\Omega \notin Con^\#$, then $\bigcup\Omega \vdash \alpha \wedge \neg\alpha$ for some formula α. By the compactness of positional logic, there is a finite subset Ψ of $\bigcup\Omega$ such that $\Psi \vdash \alpha \wedge \neg\alpha$. So, there is $\Phi \in Con^\#$ such that $\Psi \subseteq \Phi$. Hence $\Phi \vdash \alpha \wedge \neg\alpha$, which implies that Φ is inconsistent. This contradiction shows that $\bigcup\Omega \in Con^\#$. Hence $Con^\#$ is a consistent property for (D, W). If $Con \subset 2^U$ is a

consistency property for (D, W), then it is consistent by (1) of Theorem 18, which shows $Con \subseteq Con^{\#}$. ■

Example 4 (Continue of Example 3) Clearly Δ is not FC-normal w.r.t. $Con^{\#}$. In fact, $\{\neg B\} \in Con^{\#}$ and $\neg A, B \notin Th(\{\neg B\})$. If Δ is FC-normal w.r.t. $Con^{\#}$, then $Th(\{\neg B, B\}) \in Con^{\#}$, a contradiction.

In the rest of this paper, we always suppose that $Con \subset 2^U$ for any consistency property *Con* over a DT *(D, W)* unless explicitly stated.

4.4 The Connection of FC-normality and Weakly auto-compatibility

Given a consistency property *Con* for a default theory *(D, W)*, for any increasing chain $\{T_i\}$ in *Con*, $\bigcup\{T_i\} \in Con$, since *Con* is closed under the union of directed family. By the Kuratowski-Zorn Lemma, $\{Con, \subseteq\}$ has a maximal element. Let $MCon = \{T | T$ is a maximal element of *Con* under the set inclusion$\}$. Now we establish one of the main results in the paper, that is an FC-normal default theory is weakly auto-compatible. To do this we need the following lemmas.

Lemma 4 *$MCon \neq \emptyset$ and $MCon \subseteq Con$. For any $T \in MCon$, $Th(W \cup T) = T$.*

Lemma 5 *Let $\Delta = (D, W)$ be an FC-normal default theory w.r.t. Con. For any $T \in MCon$, let $D_T = \{\alpha : \beta_1, \ldots, \beta_n/\gamma \in D | \alpha \in T, \neg\beta_1, \ldots, \neg\beta_n \notin T\}$. Then $Th(W \cup Cns(D_T)) \subseteq T$ and D_T is compatible. Further, if $Th(W \cup Cns(D_T)) = T$ then D_T is strongly compatible.*

Proof: For any $\alpha : \beta_1, \ldots, \beta_n/\gamma \in D_T$, if $\gamma \notin T$, then $Th(T \cup \{\gamma\}) \in Con$ by the FC-normality of (D,W). So, $\gamma \in T$ by maximality of *T*. This implies $Cns(D_T) \subseteq T$ and $Th(W \cup Cns(D_T)) \subseteq T$. Compatibility of D_T is obvious from $Th(W \cup Cns(D_T)) \subseteq T$. If $Th(W \cup Cns(D_T)) = T$ then it is easy to prove that $\Lambda(D_T, \Delta) = D_T$ by Definition 4. ■

Lemma 6 *Let $\Delta = (D, W)$ be an FC-normal default theory w.r.t. Con. For any $D' \subseteq D$, if D' is strongly compatible, then $W \cup Cns(D') \in Con$.*

Proof: By the strongly compatibility of D' we have that $\Lambda(D', \Delta) = D'$ and D' is compatible. So $D'_\eta(\Delta)$ is compatible for any $\eta \in \mu$, where $\Lambda(D', \Delta) = \bigcup_{\eta\in\mu} D'_\eta(\Delta)$ and μ is the ordinal of D. By (transfinite) induction we show that $W \cup Cns(D'_\eta(\Delta)) \in Con$.

BASE: Consider the case $\eta = 0$. Suppose $D'_0(\Delta) = \{d_\kappa | \kappa \in \sigma\}$, where σ is the ordinal of $D'_0(\Delta)$. Now we inductively prove $W \cup Cns(\{d_\kappa | \kappa \in \rho\}) \in Con$ for any $\rho \in \sigma$.

Sub-base. $\rho = 0$. Since $\emptyset \in Con$, then $Th(W) \in Con$ since $cl_{mon}(\emptyset) = Th(W \cup \Lambda(D_M, \Delta) \cup \emptyset) \in Con$. By Definition 4, $Pre(D'_0(\Delta)) \subseteq Th(W)$. For $d_0 = \alpha : \beta_1, \ldots, \beta_n/\gamma \in D'_0(\Delta)$, we have that $\alpha \in Th(W)$ and $\neg\beta_1, \ldots, \neg\beta_n \notin Th(W)$ since D' is compatible. If $\gamma \in Th(W)$ then $W \cup \{\gamma\} \in Con$ since $W \cup \{\gamma\} \subseteq Th(W)$. Otherwise, by the FC-normality of (D,W) we have that $W \cup \{\gamma\} \in Con$ and $W \cup \{\gamma, \neg\beta\} \notin Con$ for any $\beta \in Ccs(d_0)$. This shows that $W \cup Cns(\{d_0\}) \in Con$.

Sub-step. $\rho \in \sigma$ is a successor ordinal. Assume that $W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}) \in Con$ and $d_\rho = (\alpha : \beta_1, \ldots, \beta_n/\gamma)$. Clearly, $\alpha \in Th(W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}))$ and $\neg\beta_1, \ldots, \neg\beta_n \notin Th(W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}))$ since $Pre(D'_0(\Delta)) \subseteq Th(W)$ and D' is compatible. If $\gamma \in Th(W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}))$ then $W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}) \cup \{\gamma\} \in Con$. Otherwise, by the FC-normality of (D,W) we have that $W \cup Cns(\{d_\kappa | \kappa \in \rho - 1\}) \cup \{\gamma\} \in Con$ and $W \cup \{\gamma, \neg\beta\} \notin Con$ for any $\beta \in Ccs(d_\rho)$.

ρ is a limit ordinal. Then $W \cup Cns(\{d_\kappa | \kappa \in \rho\}) = \bigcup_{\lambda\in\rho}\{W \cup Cns(\{d_\kappa | \kappa \in \lambda\})\}$ and $W \cup Cns(\{d_\kappa | \kappa \in \lambda\}) \in Con$ for any $\lambda \in \rho$. So, $W \cup Cns(\{d_\kappa | \kappa \in \lambda\}) \in Con$ by the consistency property *Con* that *Con* is closed under the union of directed family.

STEP: It is similar to that in BASE.

Finally, note that $D' = \Lambda(D', \Delta) = \bigcup_{\eta\in\mu} D'_\eta(\Delta)$ and $D'_\eta(\Delta) \subseteq D'_{\eta+1}(\Delta)$ for any $\eta \in \mu$. We have that $W \cup Cns(D') \in Con$ since Con is closed under the union of a directed family of Con. ■

The following lemma shows that an extension of an FC-normal default theory $\Delta = (D, W)$ can be gotten by enlarging a strongly compatible subset D' of *D*.

Lemma 7 *Let $\Delta = (D, W)$ be an FC-normal default theory. For any strongly compatible subset D' of D, there is an extension E of Δ such that $D' \subseteq GD(E, \Delta)$.*

Proof: Let ρ be the ordinal type of *D*. We fix some well-order $<$, which determines some listing of the elements of *D*, $\{d_\rho | \rho \in \sigma\}$, such that $\{d_\rho | \rho \in \Psi\}$ is a listing of the elements of D', where $\Psi \in \sigma$. By the FC-construction, we get an extension $E^<$ of Δ such that $E^<_\Psi \subseteq E^<$, where $E^<_\Psi = Th(W \cup Cns(D'))$, since D' is strongly compatible. Hence, $D' \subseteq GD(E^<, \Delta)$. ■

It is easy to see that Lemma 4 is a corollary of the above lemma since $W \cup Cns(GD(E, \Delta)) \in Con$, that is we get another proof of Lemma 4. The following theorem shows that each extension of an FC-normal default theory $\Delta = (D, W)$ is just generated by a maximally strongly compatible subset D' of D.

Theorem 20 *Let $\Delta = (D, W)$ be an FC-normal default theory. E is an extension of Δ iff there is a maximally strongly compatible subset D' of D such that $E = Th(W \cup Cns(D'))$.*

Proof: "Only If" It is clear since $GD(E, \Delta)$ is maximally strongly compatible by Theorem 3.

"If" From Lemma 5, we get $D' \subseteq GD(E, \Delta)$, where *E* is an extension of Δ. Since $GD(E, \Delta)$ is strongly compatible, then $D' = GD(E, \Delta)$ by the maximality of D'. So, $E = Th(W \cup Cns(D'))$. ■

From the semi-monotonicity of FC-normal default theory (Marek, Nerode, & Remmel 1994) and the essential characterization of weakly auto-compatible default theory (Theorem 14), we have the following important conclusion:

Theorem 21 *If $\Delta = (D, W)$ is an FC-normal default theory then it is weakly auto-compatible.*

However, we find a default theory Δ which is weakly auto-compatible but not FC-normal.

Example 5 Consider the default theory $\Delta = (D, W)$, where $D = \{d_1 =: \neg B/A;\quad d_2 = A : /C;\quad d_3 = C : \neg A/B;\quad d_4 = B \wedge C : /A\}$ and $W = \emptyset$. Note that, the strongly compatible sets of defaults are $S_1 = \emptyset$, $S_2 = \{d_1\}$ and $S_3 = \{d_1, d_2\}$. It is easy to see that every default $d \in D$ is weakly auto-compatible Δ and it has a unique extension $E = Th(\{A, C\})$. As a matter of fact, it is not FC-normal. Otherwise, if it is FC-normal w.r.t. a consistency Con then we have
$cl_{mon}(\emptyset) = Th(\emptyset) \in Con$ (since $\emptyset \in Con$)
$\Rightarrow \{A\} \cup Th(\emptyset) \in Con$ (since $cl_{mon}(Th(\emptyset)) = Th(\emptyset)$, $d_1 \in D_{NM}$, $Pre(d_1) \subseteq Th(\emptyset)$, $B \notin Th(\emptyset)$ and $A \notin Th(\emptyset)$)
$\Rightarrow cl_{mon}(\{A\} \cup Th(\emptyset)) = Th(\{A \cup C\}) \in Con$
$\Rightarrow Th(\{C\}) \in Con$ (since $Th(\{C\}) \subseteq Th(\{A, C\})$)
$\Rightarrow \{B\} \cup Th(\{C\}) \in Con$ (since $cl_{mon}(Th(\{C\})) = Th(\{C\})$, $d_3 \in D_{NM}$, $Pre(d_3) \subseteq Th(\{C\})$, $A \notin Th(\{C\})$ and $B \notin Th(\{C\})$)
$\Rightarrow \{A, B\} \cup Th(\{C\}) \notin Con$.
And note that $cl_{mon}(\{B\} \cup Th(\{C\})) = Th(\{A, B, C\}) \in Con$ and then $\{A, B\} \cup Th(\{C\}) \in Con$ since $\{A, B\} \cup Th(\{C\}) \subseteq Th(\{A, B, C\})$. It is a paradox.

This example together with Theorem 21 implies that weakly auto-compatible default theory properly contains FC-normal default theories. Consequently, we bridge the two classes of FC-normal default theories and weakly auto-compatible default theories.

As we pointed out, all extensions of a weakly auto-compatible default theory are just generated by maximally strongly compatible subsets of *D* and vice versa. The set of generating defaults of any such extension can be gotten, starting from the empty set, by the facts that if $\Lambda(D', \Delta) = D'$ then for any $d \in D$, $\Lambda(D' \cup \{d\}, \Delta) = D' \cup \{d\}$ if $W \cup Cns(D') \vdash Pre(\{d\})$. This is just FC-construction.

As for the application to logic programs and truth maintenance systems, it is not difficult by the connection between default logic and them. We will detail this in the future.

5 Conclusion and future work

In the paper, we present a properly larger subclass of default theory than the class of FC-normal default theory which we called weakly auto-compatible default theory. Both of them enjoy many desirable properties of normal default theory. Different from FC-normal default theory, weakly auto-compatible theory just depends on the default theory itself without a given consistency property over the default theory. And then it easier to be check than FC-normal default theory. We also present a sound and complete algorithm to compute all of its extension of weakly auto-compatible default theory. Although, to our best knowledge, both of their complexities are still open up to now. It is worthy of exploring.

Linke and Schaub presented a new approach to reason with default logic (default reasoning via blocking sets) by shifting the emphasis from application of individual defaults to that of the joint application of a default rule together with the rules supporting this application (Linke & Schaub 2000). Their compromising approach allows for reasoning (compositional) incremental construction and ensuring the existence of extensions without semi-monotonicity . In the following paper, we will explore the relationship between Linke and Schaub' s approach and our characterization.

References

Brewka, G., and Gottlob, G. 1997. Well-founded semantics for default logic. *Fundam. Inform.* 31(3/4):221–236.

Brewka, G. 1991. Cumulative default logic: In defense of nonmonotonic inference rules. *Artif. Intell.* 50(2):183–205.

Delgrande, J. P.; Schaub, T.; and Jackson, W. K. 1994. Alternative approaches to default logic. *Artif. Intell.* 70(1-2):167–237.

Giordano, L., and Martelli, A. 1994. On cumulative default logics. *Artif. Intell.* 66(1):161–179.

Linke, T., and Schaub, T. 2000. Alternative foundations for reiter's default logic. *Artif. Intell.* 124(1):31–86.

Lukaszewicz, W. 1985. Two results on default logic. In *IJCAI*, 459–461.

Lukaszewicz, W. 1988. Considerations on default logic: an alternative approach. *Computational Intelligence* 4:1–16.

Makinson, D. 2005. *Bridge from Classical to Nonmonotonic Logic*. King's College.

Marek, V. W., and Truszczynski, M. 1993. *Non-monotonic logic: context-dependent reasoning*. Springer.

Marek, V.; Nerode, A.; and Remmel, J. 1994. A context for belief revision: forward chaining-normal nonmonotonic rules systems. *Annals of Pure and Applied Logic* 67:259–323.

Mikitiuk, A., and Truszczynski, M. 1993. Rational default logic and disjunctive logic programming. In *LPNMR*, 283–299.

Mingyi, Z. 1992. A characterization of extension of general default theories. In *proceeding of 9th Canadian Conference on Artificial Intelligence*, 134–139.

Mingyi, Z. 1993. On extension of general default theories. *Sci China Ser. A,* 16(10):1273–1280.

Mingyi, Z. 1994. Some results on default logic. *J. Comput. Sci. Technol.* 9(3):267–274.

Mingyi, Z. 1996. A new research into default logic. *Inf. Comput.* 129(2):73–85.

Przymusinska, H., and Przymusinski, T. C. 1994. Stationary default extensions. *Fundam. Inform.* 21(1/2):67–87.

Reiter, R., and Criscuolo, G. 1981. On interacting defaults. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 270–276.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

2.7 Incomplete knowledge in hybrid probabilistic logic programs

Incomplete Knowledge in Hybrid Probabilistic Logic Programs

Emad Saad
College of Computer Science and Information Technology
Abu Dhabi University
Abu Dhabi, UAE
emad.saad@adu.ac.ae

Abstract

Although negative conclusions are presented implicitly in Normal Hybrid Probabilistic Programs (NHPP) (Saad & Pontelli 2005a) through the closed world assumption, representing and reasoning with explicit negation is needed in NHPP to allow the ability to reasoning with incomplete knowledge. In this paper we extend the language of NHPP to explicitly encode classical negation in addition to non-monotonic negation. The semantics of the extended language is based on the answer set semantics of traditional logic programming (Gelfond & Lifschitz 1991). We show that the proposed semantics is a natural extension to the answer set semantics of traditional logic programming (Gelfond & Lifschitz 1991). In addition, the proposed semantics is reduced to the stable probabilistic model semantics of NHPP (Saad & Pontelli 2005a). The importance of that is computational methods developed for NHPP can be applied to the proposed language. Furthermore, we show that some commonsense probabilistic knowledge can be easily represented in the proposed language.

Introduction

Hybrid Probabilistic Programs (HPP) in (Saad & Pontelli 2005b) is a probabilistic logic programming framework that modifies the original Hybrid Probabilistic Programming framework of (Dekhtyar & Subrahmanian 2000). HPP (Saad & Pontelli 2005b) enables the user to *explicitly* encode his/her knowledge about the type of dependencies existing between the probabilistic events being described by the programs. In addition, it allows the ability to encode the user's knowledge about how to combine the probabilities of the same event derived from different rules. Moreover, it subsumes Lakshmanan and Sadri's (Lakshmanan & Sadri 2001) probabilistic implication-based framework as well as it is a natural extension of traditional logic programming. As a step towards enhancing its reasoning capabilities, HPP (Saad & Pontelli 2005b) was extended to cope with non-monotonic negation (Saad & Pontelli 2005a) by introducing the notion of Normal Hybrid Probabilistic Programs (NHPP) and providing two different semantics namely; stable probabilistic model semantics and well-founded probabilistic model semantics. It was shown in (Saad & Pontelli 2005a) that the relationship between the stable probabilistic model semantics and the well-founded probabilistic model semantics *preserves* the relationship between the stable model semantics and the well-founded semantics for normal logic programs (Gelder, Ross, & Schlipf 1991). More importantly, the stable probabilistic models semantics and the well-founded probabilistic semantics *naturally extend* the stable model semantics (Gelfond & Lifschitz 1988) and the well-founded semantics (Gelder, Ross, & Schlipf 1991) of normal logic programs.

An important limitation of the language of NHPP (Saad & Pontelli 2005a) compared to traditional logic programming (Gelfond & Lifschitz 1991) is its inability to represent and reason directly in the presence of classical negation to cope with incomplete knowledge. This is because HPP (Saad & Pontelli 2005b) and NHPP (Saad & Pontelli 2005a) allow *closed world assumption* in defining their semantics. Therefore, for any event represented by a program in either HPP or NHPP there is an associated probability interval (probability interval represents the bounds on the degree of belief a rational agent has about the truth of an event.) Any event that cannot be derived from a program is assigned the probability $[0, 0]$, by default. But, an event that can be derived from the program is assigned a probability $[a, b] \neq [0, 0]$. However, a third possibility, which is *unknown* or *undecidable*, is possible which represents information incompleteness. This is because assuming that non-derivable events have the probability interval $[0, 0]$ could lead to serious implications.

Consider a medical doctor who treats his/her patient from a certain disease (di) by taking specific medication (med) for that disease. The doctor knows that if the patient took this medication he will be recovered. But the doctor also knows that the patient is suffering from a heart disease and taking that medication could affect the function of his heart and lead to death, although the medication is very effective. Therefore, the doctor can give the medication to the patient with probability $[0.87, 0.95]$ if there are no side effects of the medication on the heart with probability $[0.85, 0.85]$. This situation can be represented as an NHPP program as follow:

$$give(di, med) : [0.87, 0.95] \leftarrow not(eff(med, hrt) : [0.15, 0.15]).$$

If our knowledge regarding the side effects of the medication on the heart is incomplete because they might have not been yet clinically proven, then the medication should not be given to this specific patient, otherwise, he would probably die. The current semantics of NHPP allows us to assume that probability interval of the side effects of the medication on the heart is $[0, 0]$, which is strictly less than $[0.15, 0.15]$, and hence, medication is given to the patient, although, the program has no enough knowledge to assume the contrary.

We propose to overcome this limitation by extending the language of NHPP to explicitly allow classical negation as well as non-monotonic negation *not*, by introducing the notion of *Extended Hybrid Probabilistic Programs (EHPP)*. The semantics of EHPP is based on the answer set semantics of traditional logic programming (Gelfond & Lifschitz 1991) and employs the *Open World Assumption*. We show that some commonsense probabilistic knowledge can be easily represented in the proposed language. We present that the proposed semantics is a natural extension to the answer set semantics (Gelfond & Lifschitz 1991). Moreover, we show that the proposed semantics is reduced to stable probabilistic model semantics of NHPP (Saad & Pontelli 2005a). The importance of that is computational methods developed for NHPP can be applied to the language of EHPP.

Syntax

In the following two subsections we review the basic notions associated with EHPP (Dekhtyar & Subrahmanian 2000; Saad & Pontelli 2005a). Let $C[0,1]$ denotes the set of all closed intervals in $[0,1]$. In the context of EHPP, probabilities are assigned to primitive events (atoms) and compound events (conjunctions or disjunctions of atoms) as intervals in $C[0,1]$. Let $[\alpha_1,\beta_1],[\alpha_2,\beta_2] \in C[0,1]$. Then the *truth order* asserts that $[\alpha_1,\beta_1] \leq_t [\alpha_2,\beta_2]$ iff $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$. The set $C[0,1]$ and the relation $\leq_t$ form a complete lattice. In particular, the join ($\oplus_t$) operation is defined as $[\alpha_1,\beta_1] \oplus_t [\alpha_2,\beta_2] = [max\{\alpha_1,\alpha_2\}, max\{\beta_1,\beta_2\}]$ and the meet ($\otimes_t$) is defined as $[\alpha_1,\beta_1] \otimes_t [\alpha_2,\beta_2] = [min\{\alpha_1,\alpha_2\}, min\{\beta_1,\beta_2\}]$ w.r.t. $\leq_t$. The type of dependency among the primitive events within a compound event is described by *probabilistic strategies*, which are explicitly selected by the user. We call ρ, a pair of functions $\langle c, md\rangle$, a probabilistic strategy (p-strategy), where $c : C[0,1] \times C[0,1] \rightarrow C[0,1]$, the *probabilistic composition function*, which is *commutative*, *associative*, *monotonic* w.r.t. $\leq_t$, and meets the following *separation* criteria: there are two functions c_1, c_2 such that $c([\alpha_1,\beta_1],[\alpha_2,\beta_2]) = [c_1(\alpha_1,\alpha_2), c_2(\beta_1,\beta_2)]$. Whereas, $md : C[0,1] \rightarrow C[0,1]$ is the *maximal interval function*. The maximal interval function md of a certain p-strategy returns an estimate of the probability range of a primitive event, e, from the probability range of a compound event that contains e. The composition function c returns the probability range of a conjunction (disjunction) of two events given the ranges of its constituents. For convenience, given a multiset of probability intervals $M = \{\!\{[\alpha_1,\beta_1],\ldots,[\alpha_n,\beta_n]\}\!\}$, we use cM to denote $c([\alpha_1,\beta_1], c([\alpha_2,\beta_2],\ldots,c([\alpha_{n-1},\beta_{n-1}],[\alpha_n,\beta_n]))\ldots)$. According to the type of combination among events, p-strategies are classified into *conjunctive* p-strategies and *disjunctive* p-strategies. Conjunctive (disjunctive) p-strategies are employed to compose events belonging to a conjunctive (disjunctive) formula (please see (Dekhtyar & Subrahmanian 2000; Saad & Pontelli 2005b) for the formal definitions).

Language Syntax

In this subsection, we describe the syntax of EHPP. Let L be an arbitrary first-order language with finitely many predicate symbols, constants, and infinitely many variables. Function symbols are disallowed. In addition, let $S = S_{conj} \cup S_{disj}$ be an arbitrary set of p-strategies, where S_{conj} (S_{disj}) is the set of all conjunctive (disjunctive) p-strategies in S. The Herbrand base of L is denoted by B_L. A literal is either an atom a or the negation of an atom $\neg a$, where $\neg$ is the classical negation. We denote the set of all literals in L by Lit. More formally, $Lit = \{a | a \in B_L\} \cup \{\neg a | a \in B_L\}$. An *annotation* denotes a probability interval and it is represented by $[\alpha_1,\alpha_2]$, where α_1, α_2 are called annotation items. An *annotation item* is either a constant in $[0,1]$, a variable (*annotation variable*) ranging over $[0,1]$, or $f(\alpha_1,\ldots,\alpha_n)$ (called *annotation function*) where f is a representation of a total function $f : ([0,1])^n \rightarrow [0,1]$ and $\alpha_1,\ldots,\alpha_n$ are annotation items. The building blocks of the language of EHPP are *hybrid basic formulae*. Let us consider a set of literals $l_1,\ldots,l_n$, a conjunctive p-strategy ρ, and a disjunctive p-strategy ρ'. Then $l_1 \wedge_\rho \ldots \wedge_\rho l_n$ and $l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n$ are called *hybrid basic formulae*. A *hybrid literal* is a hybrid basic formula $l_1 \wedge_\rho \ldots \wedge_\rho l_n (l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n)$ or the negation of hybrid basic formula $\neg(l_1 \wedge_\rho \ldots \wedge_\rho l_n)(\neg(l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n))$. $bf_S(Lit)$ is the set of all ground hybrid literals formed using distinct literals from Lit and p-strategies from S. Note that any hybrid basic formula F can be represented in terms of another hybrid basic formula G such that $F = \neg G$, since $\neg\neg a = a$, $(a_1 \wedge_\rho a_2) = \neg(\neg a_1 \vee_\rho \neg a_2)$ and $(a_1 \vee_{\rho'} a_2) = \neg(\neg a_1 \wedge_{\rho'} \neg a_2)$ and $\wedge_\rho, \vee_\rho, \vee_{\rho'}$, and $\wedge_{\rho'}$ are associative and commutative. An annotated hybrid basic formula is an expression of the form $F : \mu$ where F is a hybrid basic formula and μ is an annotation. An *annotated hybrid literal* is an annotated *positive* hybrid basic formula $F : \mu$ or an annotated *negative* hybrid basic formula $(\neg F) : \mu$.

Definition 1 (E-rules) *An extended hybrid probabilistic rule (E-rule) is an expression of the form*

$$l : \mu \leftarrow L_1 : \mu_1,\ldots,L_m : \mu_m, \; not\ (L_{m+1} : \mu_{m+1}),\ldots,not\ (L_n : \mu_n)$$

where l is a literal, L_i ($1 \leq i \leq n$) are hybrid literals, and μ, μ_i ($1 \leq i \leq n$) are annotations.

The intuitive meaning of an E-rule is that, if for each $L_i : \mu_i$ ($1 \leq i \leq m$), L_i is true with probability interval at least μ_i and for each $not\ (L_j : \mu_j)$ ($m+1 \leq j \leq n$), it is not *known* that L_j is true with probability interval at least μ_j, then l is true with probability interval at least μ.

Definition 2 (E-programs) *An extended hybrid probabilistic program over S (*E-program*) is a pair $P = \langle R, \tau\rangle$, where R is a finite set of E-rules with p-strategies from S, and τ is a mapping $\tau : Lit \rightarrow S_{disj}$.*

The mapping τ in the above definition associates to each literal l a disjunctive p-strategy that will be employed to combine the probability intervals obtained from different E-rules having l in their heads. An E-program is ground if no variables appear in any of its rules.

Satisfaction and Models

In this section, we define the declarative semantics of EHPP. We define the notions of interpretations, models, and satisfaction of E-programs. The notion of a probabilistic model (p-model) is based on *hybrid formula function*.

Definition 3 *A hybrid formula function is a mapping* $h : bf_S(Lit) \rightarrow C[0,1]$ *that satisfies the following conditions:*

- *Commutativity:* $h(L_1 *_\rho L_2) = h(L_2 *_\rho L_1)$, $* \in \{\wedge, \vee\}$, $\rho \in S$
- *Composition:* $c_\rho(h(L_1), h(L_2)) \leq_t h(L_1 *_\rho L_2)$, $* \in \{\wedge, \vee\}$, $\rho \in S$
- *Decomposition. For any hybrid basic formula* L, $\rho \in S$, *and* $M \in bf_S(Lit)$: $md_\rho(h(L *_\rho M)) \leq_t h(L)$.

If the probability of an event e is $pr(e)$, then the probability of $\neg e$ is $pr(\neg e) = 1 - pr(e)$. This can be generalized to probability intervals as follows. Given $pr(e) = [\alpha_1, \alpha_2]$ is the probability interval of an event e then the probability interval of the event $\neg e$ is given by $pr(\neg e) = [1,1] - pr(e) = [1-\alpha_2, 1-\alpha_1]$. Note that Definition 3 does not restrict the assignment of probability intervals to formulae in hybrid formula functions. However, since we allow both an event and its negation to be defined in hybrid formula functions, more conditions need to be imposed on hybrid formula functions to ensure their consistency. This can be characterized by the following definition.

Definition 4 *A total (partial) hybrid formula function* h *is inconsistent if there exists* $F, \neg F \in bf_S(Lit)$ *(*$F, \neg F \in dom(h)$*) such that* $h(\neg F) \neq [1,1] - h(F)$.

Definition 4 states that a hybrid formula function h is consistent if for any $F, \neg F \in dom(h)$ we have $h(\neg F) = [1,1] - h(F)$.

Definition 5 *We say a set* C, *a subset of* Lit, *is a set of consistent literals if there is no pair of complementary literals* a *and* $\neg a$ *belonging to* C. *Similarly, a consistent set of hybrid literals* C^* *is a subset of* $bf_S(Lit)$ *such that there is no pair of complementary hybrid literals* F *and* $\neg F$ *belonging to* C^*.

Definition 6 *A consistent hybrid formula function* h *is either not inconsistent or maps a consistent set of hybrid literals* C^* *to* $C[0,1]$.

A consistent hybrid formula function is a partial or total hybrid formula function. Furthermore, given a consistent partial hybrid formula function h, *complementing* h with $\forall\ F \in dom(h), h(\neg F) = [1,1] - h(F)$ and with $\forall\ \neg G \in dom(h), h(G) = [1,1] - h(\neg G)$, still keeps h a consistent partial (or become total) hybrid formula function. We denote complementing h by $compl(h)$. For a consistent partial hybrid formula function h, we use $compl(L)$, for some $L \in dom(h)$, to denote defining $\neg L$ in h by $h(\neg L) = [1,1] - h(L)$. The notion of truth order can be employed to hybrid formula functions (partial or total). Given hybrid formula functions h_1 and h_2, we say

$$(h_1 \leq_o h_2) \Longrightarrow (dom(h_1) \subseteq dom(h_2)\ and\ \forall L \in dom(h_1)\ h_1(L) \leq_t h_2(L)).$$

The set of all hybrid formula functions, HFF, and the order $\leq_o$ form a complete lattice. The meet $\otimes_o$ and the join $\oplus_o$ operations are defined respectively as follows.

Definition 7 *Let* h_1 *and* h_2 *be two hybrid formula functions. The meet* $\otimes_o$ *and join* $\oplus_o$ *operations corresponding to the partial order* $\leq_o$ *are defined respectively as:*

- $(h_1 \otimes_o h_2)(F) = h_1(F) \otimes_t h_2(F)$ $\forall F \in (dom(h_1) \cap dom(h_2))$, *otherwise, undefined.*
- $(h_1 \oplus_o h_2)(F) = h_1(F) \oplus_t h_2(F)$ $\forall F \in (dom(h_1) \cap dom(h_2))$, $(h_1 \oplus_o h_2)(F) = h_1(F)$ $\forall F \in (dom(h_1) \setminus dom(h_2))$, *and* $(h_1 \oplus_o h_2)(F) = h_2(F)$ $\forall F \in (dom(h_2) \setminus dom(h_1))$, *otherwise, undefined.*

Definition 8 *A probabilistic interpretation (p-interpretation) of an E-program* P *is a (partial or total) hybrid formula function.*

The satisfiability of an E-program is based on the satisfaction of its E-rules.

Definition 9 (Probabilistic Satisfaction) *Let* $P = \langle R, \tau \rangle$ *be a ground E-program,* h *be a p-interpretation, and*

$$r \equiv l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n).$$

Then

- h *satisfies* $L_i : \mu_i$ *(denoted by* $h \models L_i : \mu_i$*) iff* $L_i \in dom(h)$ *and* $\mu_i \leq_t h(L_i)$.
- h *satisfies* $not\ (L_j : \mu_j)$ *(denoted by* $h \models not\ (L_j : \mu_j)$*) iff* $L_j \in dom(h)$ *and* $\mu_j \not\leq_t h(L_j)$ *or* $L_j \notin dom(h)$.
- h *satisfies*

$$Body \equiv L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n)$$

(denoted by $h \models Body$*) iff* $\forall (1 \leq i \leq m), h \models L_i : \mu_i$ *and* $\forall (m+1 \leq j \leq n), h \models not\ (L_j : \mu_j)$.

- h *satisfies* $l : \mu \leftarrow Body$ *iff* $h \models l : \mu$ *or* h *does not satisfy* $Body$.
- h *satisfies* P *iff* h *satisfies every E-rule in* R *and for every literal* $l \in dom(h)$, $c_{\tau(l)}\{\!\{\mu | l : \mu \leftarrow Body \in R\ and\ h \models Body\}\!\} \leq_t h(l)$.

Definition 10 (Models) *Let* P *be an E-program. A probabilistic model (*p-model*) of* P *is a p-interpretation* h *of* P *that satisfies* P.

We say that h is a minimal p-model of P if there is no p-model h' of P such that $h' <_o h$. An E-program without non-monotonic negation is simpler and has exactly one minimal p-model. The following results allow us to characterize the minimal (least) p-model (we call this least p-model *probabilistic answer set*) of an E-program without non-monotonic negation.

Proposition 1 *Let* $P = \langle R, \tau \rangle$ *be a ground E-program without non-monotonic negation, i.e.* $n = m$ *for each E-rule* $r \in R$, *and* h_1, h_2 *be two p-models of* P. *Then* $h_1 \otimes_o h_2$ *is also a p-model of* P.

Corollary 1 *Let P be a ground E-program without non-monotonic negation and let H_P be the set of all p-models of P. Then, $h_P = \otimes_o\{h|h \in H_P\}$ is the probabilistic answer set of P.*

However, it is possible to get the probabilistic answer set of an E-program P without non-monotonic negation and this probabilistic answer set is inconsistent. If this is the case, we say P is inconsistent. In other words, P is inconsistent if it has inconsistent probabilistic answer set. When P is inconsistent, LIT, where $LIT : bf_S(Lit) \rightarrow [1,1]$, is the probabilistic answer set of P. In this case every hybrid literal with probability interval $[1,1]$ follows from P. We adopt this view from the answer set semantics of traditional logic programming (Gelfond & Lifschitz 1991).

Example 1 *Consider the following E-program $P = \langle R, \tau\rangle$ without non-monotonic negation, where R is*

$$\begin{array}{lcl} c : [0.35, 0.91] & \leftarrow & a : [0, 0.11], b : [0.8, 0.99] \\ \neg c : [0, 0.21] & \leftarrow & a : [0.1, 0.13], \neg b : [0.05, 0.08] \\ d : [0.12, 0.18] & \leftarrow & c : [0.35, 0.65] \\ \neg d : [0.45, 0.55] & \leftarrow & a : [0, 0.15], \neg b : [0.02, 0.22], \\ & & \neg c : [0, 0.1] \\ \neg b : [0.15, 0.3] & \leftarrow & \\ a : [0.1, 0.2] & \leftarrow & \end{array}$$

and τ is any arbitrary assignment of disjunctive p-strategies. It is easy to verify that P has (unique) probabilistic answer set h where $h(a) = [0.1, 0.2], h(\neg b) = [0.15, 0.3], h(\neg c) = [0, 0.21], h(\neg d) = [0.45, 0.55]$.

Now, suppose P is an E-program without non-monotonic negation and h is a probabilistic answer set for P. Then, if we complement h, denoted by $compl(h)$, do we still have h as a probabilistic answer set for P? The answer to this question is *no* in general. Since after $compl(h)$, it is possible to have h as a p-model for P but not a probabilistic answer set or even not a p-model at all. We show this by following example.

Example 2 *Consider the following E-program $P = \langle R, \tau\rangle$ where R is*

$$\begin{array}{lcl} b : [0.6, 0.7] & \leftarrow & \neg a : [0.7, 0.8] \\ a : [0.5, 0.6] & \leftarrow & b : [0.55, 0.7] \\ a : [0.2, 0.3] & \leftarrow & \\ b : [0.4, 0.5] & \leftarrow & \end{array}$$

and $\tau(a) = \tau(b) = pcd$ and $\tau(\neg b) = \tau(\neg a) = \pi$ where π is any arbitrary disjunctive p-strategy. pcd denotes the disjunctive positive correlation p-strategy whose composition function is defined as: $c_{pcd}([\alpha_1, \beta_1], [\alpha_2, \beta_2]) = [\max(\alpha_1, \alpha_2), \max(\beta_1, \beta_2)]$. We can easily see that h_1 is the probabilistic answer set of P where $h_1(a) = [0.2, 0.3], h_1(b) = [0.4, 0.5]$. Now, let us $compl(h_1)$ and see if the new h_1 is a probabilistic answer set of P. After $compl(h_1)$ we get a new h_1 as $h_1(a) = [0.2, 0.3], h_1(b) = [0.4, 0.5], h_1(\neg a) = [0.7, 0.8], h_1(\neg b) = [0.5, 0.6]$, which is a consistent p-interpretation but it is not a p-model of P, and hence, it is not a probabilistic answer set of P. Note that after $comp(h_1)$, if we remove $h_1(\neg a)$ from the definition of h_1, we get $h_1(a) = [0.2, 0.3], h_1(b) = [0.4, 0.5], h_1(\neg b) = [0.5, 0.6]$, which is a p-model of P but it is not a probabilistic answer set. However, it can be a probabilistic answer set of P except that the condition $c_{\tau(\neg b)}\{\!\{\mu|\neg b : \mu \leftarrow Body \in R \text{ and } h_1 \models Body\}\!\} \leq_t h_1(\neg b)$ does not apply, because there is no E-rule in P with $\neg b$ appearing in its head. I.e., there are no E-rules in P that support the probability interval assigned to $\neg b$. The only support to the probability interval of $\neg b$ is $h_1(\neg b) = [1,1] - h_1(b)$. This can be done for any hybrid literal L defined in the probabilistic answer set h of an E-program P without non-monotonic negation, if the program P as a whole provides no means to derive more information about the probability of L. This means that, given P, our knowledge about the probability of L is complete, and hence, we can assert that the probability of $\neg L$ is $h(\neg L) = [1,1] - h(L)$. In fact, we cannot assign any $[\alpha_1, \alpha_2]$ to $\neg b$ in h_1 such that $h_1(\neg b) = [\alpha_1, \alpha_2] <_t [0.5, 0.6]$ or $[0.5, 0.6] <_t [\alpha_1, \alpha_2] = h_1(\neg b)$. This is because of the inconsistency condition, otherwise, the probabilistic answer set of P is LIT.

Definition 11 *Let P be an E-program without non-monotonic negation and h be the probabilistic answer set of P (different from LIT). The final probabilistic answer set of P is the probabilistic answer set h after $compl(L)$ w.r.t. h for some $L \in dom(h)$ such that the program P as a whole provides no means to derive more information about the probability of L assigned by the original h.*

Example 3 *Consider the following E-program $P = \langle R, \tau\rangle$ where R is*

$$\begin{array}{lcl} b : [0.3, 0.4] & \leftarrow & \neg a : [0.7, 0.8] \\ a : [0.1, 0.22] & \leftarrow & b : [0.55, 0.7] \\ a : [0.2, 0.3] & \leftarrow & \\ b : [0.4, 0.5] & \leftarrow & \end{array}$$

and $\tau(a) = \tau(b) = pcd$ and $\tau(\neg b) = \tau(\neg a) = \pi$ where π is any arbitrary disjunctive p-strategy. Then h where $h(a) = [0.2, 0.3]$ $h(b) = [0.4, 0.5]$ $h(\neg a) = [0.7, 0.8]$ $h(\neg b) = [0.5, 0.6]$ is the final probabilistic answer set of P. Since, $h(a) = [0.2, 0.3], h(b) = [0.4, 0.5]$ is the probabilistic answer set of P and the program P as a whole does not allow us to derive more information about the probability assigned to a and b by h.

In the rest of paper, we will consider probabilistic answer sets not the final probabilistic answer sets.

Proposition 2 *Every E-program P without non-monotonic negation has unique probabilistic answer set h_P.*

Associated with each E-program P without non-monotonic negation, is an operator, T_P, called the *fixpoint operator*, which maps a p-interpretation to a p-interpretation.

Definition 12 *Let $P = \langle R, \tau\rangle$ be a ground E-program without non-monotonic negation, h be a p-interpretation, and HFF be the set of all hybrid formula functions. The fixpoint operator T_P is a mapping $T_P : HFF \rightarrow HFF$ which is defined as follows:*

1. *if l is a literal, $T_P(h)(l) = c_{\tau(l)}\ M_l$ where $M_l = \{\!\{\mu|l : \mu \leftarrow Body \in R \text{ such that } h \models Body\}\!\}$.*
2. *$T_P(h)(L_1 \wedge_\rho L_2) = c_\rho(T_P(h)(L_1), T_P(h)(L_2))$ where $(L_1 \wedge_\rho L_2) \in bf_S(Lit)$ and $L_1, L_2, \in dom(T_P(h))$*

3. $T_P(h)(L_1 \vee_{\rho'} L_2) = c_{\rho'}(T_P(h)(L_1), T_P(h)(L_2))$ *where* $(L_1 \vee_{\rho'} L_2) \in bf_S(Lit)$ *and* $L_1, L_2, \in dom(T_P(h))$.

If M_l is empty—i.e., there are no E-rules in P whose heads contain l such that their bodies are satisfied by h—then no probability interval is assigned to l. This means the probability interval of l is *unknown* with respect to h. Let us now proceed in the construction of the probabilistic answer set as repeated iteration of the fixpoint operator T_P.

Definition 13 *Let P be a ground E-program without non-monotonic negation. Then*

- $T_P \uparrow 0 = \emptyset$ *where $\emptyset$ is the empty set.*
- $T_P \uparrow \alpha = T_P(T_P \uparrow (\alpha - 1))$ *where α is a successor ordinal.*
- $T_P \uparrow \lambda = \oplus_o\{T_P \uparrow \alpha | \alpha < \lambda\}$ *where λ is a limit ordinal.*

Lemma 1 *The T_P operator is monotonic.*

The properties of the T_P operator guarantee the existence of a least fixpoint and its correspondence to the probabilistic answer set of E-programs without non-monotonic negation.

Proposition 3 *Let P be an E-program without non-monotonic negation and h be a p-interpretation. Then h is a p-model of P iff $T_P(h) \leq_o h$.*

Theorem 1 *Let P be an E-program without non-monotonic negation. Then, $h_P = lfp(T_P)$.*

Example 4 *Let us reconsider the E-program P, without non-monotonic negation, described in Example 1. It is easy to see that the $lfp(T_P)$ assigns $[0.1, 0.2]$ to a, $[0.15, 0.3]$ to $\neg b$, $[0, 0.21]$ to $\neg c$, and $[0.45, 0.55]$ to $\neg d$.*

Probabilistic Answer Set Semantics for E-programs

In this section we define the *probabilistic answer sets* of E-programs (with non-monotonic negation), which extend the notion of answer sets for traditional logic programming (Gelfond & Lifschitz 1991). The semantics is defined in two steps. First, we guess a probabilistic answer set h for a certain E-program P, then we define the notion of the probabilistic reduct of P with respect to h. The probabilistic reduct is an E-program without non-monotonic negation which has a unique probabilistic answer set. Second, we determine whether h is a probabilistic answer set for P. This is verified by determining whether h is the probabilistic answer set of the probabilistic reduct of P w.r.t. h.

Definition 14 (Probabilistic Reduct) *Let $P = \langle R, \tau \rangle$ be a ground E-program and h be a p-interpretation. The probabilistic reduct P^h of P w.r.t. h is $P^h = \langle R^h, \tau \rangle$ where:*

$$R^h = \left\{ \begin{array}{l} l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m | \\ l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, \\ \quad not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n) \in R\ and \\ \forall (m+1 \leq j \leq n),\ \mu_j \not\leq_t h(L_j)\ or\ L_j \notin dom(h). \end{array} \right\}$$

The probabilistic reduct P^h is an E-program without non-monotonic negation. Therefore, its probabilistic answer set is well-defined. For any $not\ (L_j : \mu_j)$ in the body of $r \in R$ with $\mu_j \not\leq_t h(L_j)$ means that it is not known that the probability interval of L_j is at least μ_j given the available knowledge, and $not\ (L_j : \mu_j)$ is removed from the body of r. In addition, if $L_j \notin dom(h)$, i.e., L_j is undefined in h, then it is completely *not known (undecidable)* that the probability interval of L_j is at least μ_j. In this case, $not\ (L_j : \mu_j)$ is also removed from the body of r. Here we distinguish between the case where it is not known the probability of L_j is at least μ_j because we have some but incomplete knowledge about the probability of L_j (by $\mu_j \not\leq_t h(L_j)$) and the case where we completely have no knowledge about the probability interval of L_j (by $L_j \notin dom(h)$). If $\mu_j \leq_t h(L_j)$ then we know that the probability interval of L_j is at least μ_j and the body of r is not satisfied and r is trivially ignored.

Definition 15 *A p-interpretation h is a probabilistic answer set of an E-program P if h is the probabilistic answer set of P^h.*

The domain of a probabilistic answer set of an E-program represents an agent set of beliefs based on the knowledge represented by the E-program. However, the probability intervals associated to these beliefs represent the agents belief degrees on these beliefs. Intuitively, the probabilistic answer sets of an E-program are the possible sets of beliefs with associated beliefs degrees an agent might have. Note that E-programs without classical negation (normal hybrid probabilistic programs (Saad & Pontelli 2005a)), i.e., E-programs that contain no negative literals neither in head nor in the body of E-rules, have probabilistic answer sets with hybrid literals consisting of only atoms. In other words, the domain of probabilistic answer set in this case consists of positive hybrid basic formulae. Moreover, the definition of probabilistic answer sets coincides with the definition of stable probabilistic models defined in (Saad & Pontelli 2005a). This implies that the probabilistic answer sets for a normal hybrid probabilistic program are equivalent to its stable probabilistic models. This means that the application of probabilistic answer set semantics to normal hybrid probabilistic programs is reduced to the stable probabilistic model semantics for normal hybrid probabilistic programs. However, there are a couple of main differences between the two semantics. A probabilistic answer set may be a partial p-interpretation, however, a stable probabilistic model is a total p-interpretation. In addition, each hybrid basic formula F with probability interval [0,0]— i.e. there is no proof that F has probability interval different from $[0, 0]$ or F is false by default— in a stable probabilistic model of a normal hybrid probabilistic program corresponds to the fact that the probability interval of F is unknown, and hence undefined, in its equivalent probabilistic answer set—i.e., the probability of F is unknown.

Proposition 4 *Let P be an E-program without classical negation. Then h is a probabilistic answer set for P iff h' is a stable probabilistic model of P, where $h(F) = h'(F)$ for $h'(F) \neq [0, 0]$ and $h(F)$ is undefined for $h'(F) = [0, 0]$.*

Proposition 4 shows that there is a simple reduction from E-programs to normal hybrid probabilistic programs. The importance of this is that, under the consistency condition,

computational methods developed for normal hybrid probabilistic programs can be applied to extended hybrid probabilistic programs.

Example 5 *In addition to the intuitive representation, the undesirable consequences due to the use of non-monotonic negation in the hear medication example described in the introduction,*

$$give(di, med) : [0.87, 0.95] \leftarrow not(eff(med, hrt) : [0.15, 0.15]),$$

can be eliminated by using classical negation. Therefore, by using the classical negation we get

$$give(di, med) : [0.87, 0.95] \leftarrow \neg\, eff(med, hrt) : [0.85, 0.85].$$

Then, $give(di, med)$ *can be concluded with probability interval* $[0.87, 0.95]$ *if no side effects of the medication on the heart,* $\neg\, eff(med, hrt)$*, is concluded with probability interval at least* $[0.85, 0.85]$.

Example 6 *Suppose that we know a bird can fly with probability at least the probability between 70% and 85% as long as it is not known that it is incapable of flying with probability at least the probability in the range 30% to 35%. However, a bird is incapable of flying with probability at least the probability from 48% to 65% if it is wounded with probability at least the probability between 50% to 68%. Nevertheless, certainly, a bird is incapable of flying if it is a penguin. In addition, we also know that Tweety and Rocky are birds. Rocky is penguin, and there is over 70% chance that Tweety is injured. This can be represented by the following E-program* $P = \langle R, \tau \rangle$ *where* R *is*

$$\begin{array}{lcl}
fly(X) : [0.7, 0.85] & \leftarrow & bird(X) : [1, 1], \\
 & & not(\neg fly(X) : [0.3, 0.35]) \\
\neg fly(X) : [0.48, 0.65] & \leftarrow & wounded(X) : [0.5, 0.68] \\
\neg fly(X) : [1, 1] & \leftarrow & penguin(X) : [1, 1] \\
bird(tweety) : [1, 1] & \leftarrow & \\
wounded(tweety) : [0.7, 1] & \leftarrow & \\
bird(rocky) : [1, 1] & \leftarrow & \\
penguin(rocky) : [1, 1] & \leftarrow &
\end{array}$$

and τ *is any arbitrary assignment of disjunctive p-strategies.* P *has only one probabilistic answer set* h *where*

$$\begin{array}{ll}
h(bird(tweety)) & = \textit{ [1,1]} \\
h(bird(rocky)) & = \textit{ [1,1]} \\
h(wounded(tweety)) & = \textit{ [0.7,1]} \\
h(penguin(rocky)) & = \textit{ [1,1]} \\
h(\neg fly(tweety)) & = \textit{ [0.48,0.65]} \\
h(\neg fly(rocky)) & = \textit{ [1,1].}
\end{array}$$

Although completely different, the following examples are inspired by examples described in (Loyer & Straccia 2002; Dekhtyar, Dekhtyar, & Subrahmanian 1999).

Example 7 *A detective is investigating a crime by considering the persons who might be suspects among the ones who have been recalled for questions during the investigation. The detective is considering a person as a suspect with probability at least the probability between 55% to 70% if the probability that person lies is at least the probability from 47% to 60% and has a motive to commit the crime with probability at least the probability in the range 50% to 55%. A person is also a suspect with at least 60% to 90% probability if there is at least 70% to 82% chance that there is another person who is willing to witness against him. However, the detective does not consider a person as a suspect with probability at least the probability from 48% to 65% if that person does not lie with probability at least the probability between 68% and 90%. But the detective needs to collect more evidence about a person (investigate) with probability at least the probability from 87% to 93% whenever it is not known that person is suspect with at least 50% to 85% probability and is not a suspect with at least 40% to 70% probability. The detective is over 70% sure that Frank has a motive to commit the crime, but, he also knows that John is not willing to witness against Frank with probability as low as in the range 20% to 25%. This can be represented by the following E-program* $P = \langle R, \tau \rangle$ *where* R *is*

$$\begin{array}{l}
suspect(X) : [0.55, 0.7] \leftarrow lie(X) : [0.47, 0.6], \\
\qquad\qquad\qquad\qquad\quad motive(X) : [0.5, 0.55] \\
suspect(X) : [0.6, 0.9] \leftarrow witness(X, Y) : [0.7, 0.82] \\
\neg suspect(X) : [0.48, 0.65] \leftarrow \neg lie(X) : [0.68, 0.9] \\
investigate(X) : [0.87, 0.93] \leftarrow not\ (suspect(X) : [0.5, 0.85]), \\
\qquad\qquad\qquad\qquad\quad not\ (\neg suspect(X) : [0.4, 0.7]) \\
motive(frank) : [0.7, 1] \leftarrow \\
\neg witness(frank, john) : [0.2, 0.25] \leftarrow
\end{array}$$

and τ *is any arbitrary assignment of disjunctive p-strategies.* P *has only one probabilistic answer set* h *where*

$$\begin{array}{ll}
h(\neg witness(frank, john)) & = \textit{ [0.2,0.25]} \\
h(motive(frank)) & = \textit{ [0.7,1]} \\
h(investigate(frank)) & = \textit{ [0.87,0.93].}
\end{array}$$

Note that our knowledge about the probability of $\neg witness(frank, john), motive(frank)$, and $investigate(frank)$ is complete given the above E-program. Therefore, we can assert that

$$\begin{array}{ll}
h(witness(frank, john)) & = \text{ [1,1] - [0.2,0.25]} \\
 & = \text{ [0.75, 0.8]} \\
 & \\
h(\neg motive(frank)) & = \text{ [1,1] - [0.7,1]} \\
 & = \text{ [0.,0.3]} \\
h(\neg investigate(frank)) & = \text{ [1,1] - [0.87,0.93]} \\
 & = \text{ [0.07,0.13].}
\end{array}$$

Example 8 *It is probable that Frank was in the crime scene. However, it is not yet known that Frank is a suspect. Therefore, the detective has decided to question the persons who also were seen in the crime scene due to the need for more investigation around Frank. The detective questions a person with probability at least the probability between 77% and 93% if that person was seen in the crime seen with probability at least the probability from 52% to 68% where Frank was also seen with probability at least the probability from 40% to 52%. If the detective is ignorant about the relationship between a person who was seen in the crime scene and the fact that Frank was also seen in the crime scene with probability at least the probability in the range 60 % to 79%, then there is probability at least the probability between 50% and 65% that the detective questions that person. However, if there is a positive correlation between a person who was not seen in the crime scene and Frank was seen in the crime scene with probability at least the probability from 80% to 95%, then there is over 50% probability that person is questioned. But, if the detective does not know whether to question a person with probability at least 50% to 85% and not to question that person with probability at least 40% to 70%, then he considers that person is not a suspect with probability at least 87% to 93%. The detective also knows that if a person was seen in the crime scene with probability*

interval $[V_1, V_2]$, then he was not seen with probability interval $[1 - V_2, 1 - V_1]$. But if he was not seen with probability interval $[V_1, V_2]$, then he was seen with probability interval $[1 - V_2, 1 - V_1]$. Further investigations have shown that Frank was seen in the crime scene with probability 55% to 73%, however, Mark was not seen with probability 30% to 40%. This can be represented by the following E-program $P = \langle R, \tau \rangle$ where R is

$$
\begin{array}{l}
question(X) : [0.77, 0.93] \leftarrow seen(X) : [0.52, 0.68], \\
\qquad\qquad\qquad\qquad\qquad seen(frank) : [0.4, 0.52] \\
question(X) : [0.5, 0.65] \leftarrow \\
\qquad (seen(X) \wedge_{ig} seen(frank)) : [0.6, 0.79] \\
question(X) : [0.5, 1] \leftarrow \\
\qquad (\neg seen(X) \wedge_{pc} seen(frank)) : [0.8, 0.95] \\
\neg suspect(X) : [0.87, 0.93] \leftarrow not\ (question(X) : [0.5, 0.85]), \\
\qquad\qquad\qquad\qquad\qquad not\ (\neg question(X) : [0.4, 0.7]) \\
\neg seen(X) : [1 - V_2, 1 - V_1] \leftarrow seen(X) : [V_1, V_2] \\
seen(X) : [1 - V_2, 1 - V_1] \leftarrow \neg seen(X) : [V_1, V_2] \\
seen(frank) : [0.55, 0.73] \leftarrow \\
\neg seen(mark) : [0.3, 0.4] \leftarrow
\end{array}
$$

and τ is any arbitrary assignment of disjunctive p-strategies. $\wedge_{ig}$ and $\wedge_{pc}$ correspond to the conjunctive ignorance p-strategy igc—whose composition function is defined as $c_{igc}([\alpha_1, \beta_1], [\alpha_2, \beta_2]) = [\max(0, \alpha_1 + \alpha_2 - 1), \min(\beta_1, \beta_2)]$—and the conjunctive positive correlation p-strategy pcc—whose composition function is defined as $c_{pcc}([\alpha_1, \beta_1], [\alpha_2, \beta_2]) = [\min(\alpha_1, \alpha_2), \min(\beta_1, \beta_2)]$—respectively. P has only one probabilistic answer set h where

$$
\begin{array}{l}
h(seen(frank)) = [0.55, 0.73] \\
h(\neg seen(frank)) = [0.27, 0.45] \\
h(\neg seen(mark)) = [0.3, 0.4] \\
h(seen(mark)) = [0.6, 0.7] \\
h((seen(mark) \wedge_{ig} seen(frank))) = [0.14, 0.7] \\
h(question(mark)) = [0.77, 0.93] \\
h((\neg seen(mark) \wedge_{pc} seen(frank))) = [0.3, 0.4].
\end{array}
$$

In the following we define the *immediate consequence operator* of E-programs and study its relationship to the probabilistic answer sets.

Definition 16 *Let $P = \langle R, \tau \rangle$ be a ground E-program and $h \in HFF$. The* immediate consequence operator *T'_P is a mapping $T'_P : HFF \rightarrow HFF$ defined as follows:*

1. $T'_P(h)(l) = c_{\tau(l)}\ M'_l$ *where*

$$
M'_l = \left\{\!\!\left\{ \mu \;\middle|\; \begin{array}{l} l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, \\ not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n) \in R \\ and\ \forall (1 \leq i \leq m), h \models L_i : \mu_i \\ and\ \forall (m+1 \leq j \leq n), h \models not\ (L_j : \mu_j) \end{array} \right\}\!\!\right\}
$$

2. $T'_P(h)(L_1 \wedge_\rho L_2) = c_\rho(T'_P(h)(L_1), T'_P(h)(L_2))$ *where* $(L_1 \wedge_\rho L_2) \in bf_S(Lit)$ *and* $L_1, L_2 \in dom(T'_P(h))$.
3. $T'_P(h)(L_1 \vee_{\rho'} L_2) = c_{\rho'}(T'_P(h)(L_1), T'_P(h)(L_2))$ *where* $(L_1 \vee_{\rho'} L_2) \in bf_S(Lit)$ *and* $L_1, L_2 \in dom(T'_P(h))$.

It is easy to see that T'_P extends T_P to handle E-rules with non-monotonic negation, and hence, $T'_P = T_P$ for any E-program P without non-monotonic negation.

Theorem 2 *Let $P = \langle R, \tau \rangle$ be an E-program such that for every E-rule in R, $n = m$. Then $T'_P = T_P$.*

The operator T'_P is not monotonic w.r.t. $\leq_o$. This can be seen by the following result.

Proposition 5 *T'_P is not monotonic w.r.t. $\leq_o$.*

Example 9 *Consider the E-program:*

$$a : [0.2, 0.3] \leftarrow not\ (b : [0.6, 0.8]).$$

Let $h_1 = \emptyset$ be a p-interpretation. In addition, let h_2 be a p-interpretation that assigns $[0.65, 0.9]$ to b. Hence, $h_1 \leq_o h_2$. But $T'_P(h_1)$ assigns $[0.2, 0.3]$ to a and $T'_P(h_2) = \emptyset$. Thus, $T'_P(h_1) \not\leq_o T'_P(h_2)$

The following results establish the relationship between the T'_P operator and the probabilistic answer set semantics.

Lemma 2 *Let P be an E-program and h be a probabilistic answer set of P. Then $T'_P(h) = h$, i.e., h is a fixpoint of T'_P.*

Theorem 3 *Let P be an E-program and h be a probabilistic answer set of P. Then h is a minimal fixpoint of T'_P.*

It is worth noting that not every minimal fixpoint of T'_P is a probabilistic answer set for P. Consider the following E-program P.

Example 10 *Let $P = \langle R, \tau \rangle$ where τ is arbitrary and R contains*

$$
\begin{array}{lcl}
a : [0.1, 0.33] & \leftarrow & not\ (a : [0.1, 0.33]) \\
a : [0.1, 0.33] & \leftarrow & b : [1, 1]
\end{array}
$$

It is easy to verify that the p-interpretation $h(a) = [0.1, 0.33]$ and $h(b) = [1, 1]$ is a minimal fixpoint of T'_P. However, P^h consists of $a : [0.1, 0.33] \leftarrow b : [1, 1]$ where $lfp(T_{P^h}) = \emptyset$. Hence, h is not a probabilistic answer set for P.

Let us show that the probabilistic answer set semantics generalizes the answer set semantics of extended logic programs in traditional logic programming (Gelfond & Lifschitz 1991). An extended logic program P can be represented as an E-program $P' = \langle R, \tau \rangle$ where each extended rule

$$l \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \in P$$

can be encoded, in R, as an E-rule of the form

$$l : \nu \leftarrow l_1 : \nu, \ldots, l_m : \nu, not\ (l_{m+1} : \nu), \ldots, not\ (l_n : \nu)$$

belonging to R, where $\nu \equiv [1, 1]$, $l, l_1, \ldots, l_m, l_{m+1}, \ldots, l_n$ are literals, and $[1, 1]$ represents the truth value *true*. τ is any arbitrary assignment of disjunctive p-strategies. We call the class of E-programs that consists only of E-rules of the above form as $EHPP_1$. The following result shows that extended programs (Gelfond & Lifschitz 1991) are subsumed by EHPP.

Proposition 6 *Let P be an extended logic program. Then S' is a an answer set of P iff h is a probabilistic answer of $P' \in EHPP_1$ that corresponds to P where $h(l) = [1, 1]$ iff $l \in S'$ and $h(l')$ is undefined iff $l' \notin S'$.*

Related Work

The problem of extending uncertain logic programming in general and probabilistic logic programming in particular with non-monotonic negation (negation-as-failure or default negation) has been extensively studied in the literature. A survey on these various approaches can be found in (Saad & Pontelli 2005a). However, the main difference in this work is that we allow classical negation as well as non-monotonic negation to reason with incomplete knowledge, given the underlying semantics is the answer set semantics for traditional logic programming (Gelfond & Lifschitz 1991), which has not been addressed by the current work in probabilistic logic programming. The closest to our work is the work presented in (Baral, Gelfond, & Rushton 2004). In (Baral, Gelfond, & Rushton 2004), an elegant way has been presented to reason with causal Bayesian nets by considering a body of logical knowledge, by using the answer set semantics of traditional logic programming (Gelfond & Lifschitz 1991). Answer set semantics (Gelfond & Lifschitz 1991) has been used to emulate the possible world semantics. Probabilistic logic programs of (Baral, Gelfond, & Rushton 2004) is expressive and straightforward and relaxed some restrictions on the logical knowledge representation part existed in similar approaches to Bayesian reasoning, e.g., (Kersting & Raedt 2000; Muggleton 1995; Poole 1997; 2000; Vennekens, Verbaeten, & Bruynooghe 2004). Since (Ng & Subrahmanian 1992; 1993; 1994; Dekhtyar & Subrahmanian 2000) provided a different semantical characterization to probabilistic logic programming, it was not clear that how these proposals relate to (Baral, Gelfond, & Rushton 2004). However, the work presented in this paper and (Saad & Pontelli 2005a), which are modification and generalization of the work presented in (Ng & Subrahmanian 1992; 1993; 1994; Dekhtyar & Subrahmanian 2000), are closely related to (Baral, Gelfond, & Rushton 2004). The work presented in this paper strictly syntactically and semantically subsumes probabilistic logic programs of (Baral, Gelfond, & Rushton 2004). This can be easily argued by the fact that EHPP naturally extends traditional logic programming with answer set semantics (Gelfond & Lifschitz 1991), and probabilistic logic programs of (Baral, Gelfond, & Rushton 2004) mainly rely on traditional logic programming with answer set semantics (Gelfond & Lifschitz 1991) as a knowledge representation and inference mechanism for reasoning with causal Bayesian nets. This is true although EHPP does not allow disjunctions in the head of rules since it is easy to transform an extended disjunctive logic program into an equivalent extended logic program via a simple transformation (Baral 2003). In this sense, the comparisons established between (Baral, Gelfond, & Rushton 2004) and the existing probabilistic logic programming approaches such as (Kersting & Raedt 2000; Muggleton 1995; Poole 1997; 2000; Vennekens, Verbaeten, & Bruynooghe 2004; Ng & Subrahmanian 1992; 1993; 1994; Dekhtyar & Subrahmanian 2000; Lukasiewicz 1998; Dekhtyar & Dekhtyar 2004) also carry over to EHPP and these approaches. In addition, unlike (Baral, Gelfond, & Rushton 2004), EHPP does not put any restriction on the type of dependency existing among events.

Conclusions and Future Work

We presented an extension to the language of normal hybrid probabilistic programs (Saad & Pontelli 2005a), called extended hybrid probabilistic programs, to allow classical negation, in addition to, non-monotonic negation. The extension is important to provide the capability of reasoning with incomplete knowledge. We developed a semantical characterization of the extended language, which relies on a probabilistic generalization of the answer set semantics, originally developed for extended logic programs (Gelfond & Lifschitz 1991). We showed that the probabilistic answer set semantics naturally generalizes the answer set semantics for extended logic programs (Gelfond & Lifschitz 1991). Furthermore, we showed that the proposed semantics is reduced to stable probabilistic model semantics of NHPP proposed in (Saad & Pontelli 2005a). The importance of that computational methods developed for NHPP can be applied to the language of EHPP. Moreover, we showed that some commonsense probabilistic knowledge can be easily represented in the proposed language.

A topic of future research is to extend the language of extended hybrid probabilistic programs to allow disjunctions of annotated literals in the heads of rules. In addition, we intend to investigate the computational aspects of the probabilistic answer set semantics—by developing algorithms and implementations for computing the proposed semantics. The algorithms and implementations we will develop will be based on appropriate extensions of the existing technologies for computing the answer semantics for extended logic programs.

References

Baral, C.; Gelfond, M.; and Rushton, N. 2004. Probabilistic reasoning with answer sets. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer Verlag.

Baral, C. 2003. *Knowledge representation, reasoning, and declarative problem solving*. Cambridge University Press.

Dekhtyar, A., and Dekhtyar, I. 2004. Possible worlds semantics for probabilistic logic programs. In *International Conference on Logic Programming*, 137–148.

Dekhtyar, A., and Subrahmanian, V. 2000. Hybrid probabilistic program. *Journal of Logic Programming* 43(3):187–250.

Dekhtyar, M.; Dekhtyar, A.; and Subrahmanian, V. S. 1999. Hybrid probabilistic programs: Algorithms and complexity. In *Uncertainty in Artificial Intelliegence*, 160–169.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of ACM* 38(3):620–650.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICSLP*. MIT Pres.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3-4):363–385.

Kersting, K., and Raedt, L. D. 2000. Bayesian logic programs. In *Inductive Logic Programming*.

Lakshmanan, L., and Sadri, F. 2001. On a theory of probabilistic deductive databases. *Journal of Theory and Practice of Logic Programming* 1(1):5–42.

Loyer, Y., and Straccia, U. 2002. The well-founded semantics in normal logic programs with uncertainty. In *FLOPS*. Springer Verlag.

Lukasiewicz, T. 1998. Probabilistic logic programming. In *13th European Conference on Artificial Intelligence*, 388–392.

Muggleton, S. 1995. Stochastic logic programming. In *5th International Workshop on Inductive Logic Programming*.

Ng, R., and Subrahmanian, V. 1992. Probabilistic logic programming. *Information & Computation* 101(2).

Ng, R., and Subrahmanian, V. 1993. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *ARJ* 10(2).

Ng, R., and Subrahmanian, V. 1994. Stable semantics for probabilistic deductive databases. *Information & Computation* 110(1).

Poole, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94(1-2):7–56.

Poole, D. 2000. Abducing through negation as failure: stable models within the independent choice logic. *Journal of Logic Programming* 44(5-35).

Saad, E., and Pontelli, E. 2005a. Hybrid probabilistic logic programs with non-monotonic negation. In *International Conference on Logic Programming*.

Saad, E., and Pontelli, E. 2005b. Towards a more practical hybrid probabilistic logic programming framework. In *Practical Aspects of Declarative Languages*.

Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*, 431–445.

2.8 Extending the role of causality in probabilistic modeling

Extending the Role of Causality in Probabilistic Modeling

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe
{joost, marcd, maurice}@cs.kuleuven.be
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium

Abstract

Causality plays an important role in probabilistic modeling. Often, a probability distribution can be naturally described as the outcome of a causal process, in which different random variables interact through a series of non-deterministic events. However, formal tools such as Bayesian networks do not directly represent such events, but focus instead on derivate concepts such as probabilistic independencies and conditional probabilities. In this paper, we present a logic, designed from fundamental causal principles, which has a representation of such non-deterministic, probabilistic events as its basic construct. We show that Bayesian networks can be described in this language and illustrate some of its interesting properties. We then relate this logic to a certain class of probabilistic logic programming languages. We show that our logic induces a semantics for disjunctive logic programs, in which these represent non-deterministic processes. We also show that logic programs under the well-founded semantics can be seen as a language of deterministic causality, which we relate to McCain & Turner's causal theories.

Introduction

Causal information plays a crucial role in commonsense reasoning in general and probabilistic modeling in particular. The underlying assumption in work such as Pearl's influential treatment of causality (Pearl 2000) seems to be the assumption that the world can be viewed as consisting of a number of non-deterministic causal events. However, the formal tools that are typically used in probabilistic modeling—most notably Bayesian networks—do not directly represent such causal events. Instead, derivate kinds of information, such as probabilistic (in-)dependencies and conditional probability distributions, are represented.

In this paper, we propose a more direct approach, which is based on the following construct of a *conditional probabilistic event*, or *CP-event* for short: "If property φ is satisfied, then a probabilistic event will happen that causes at most one of propositions h_1, h_2, ..., h_m, where the probability of h_1 being caused is α_1, the probability of h_2 is α_2, ..., and the probability of h_m is α_m." We use the following syntax to represent a CP-event of the above form: $(h_1 : \alpha_1) \vee \cdots \vee (h_m : \alpha_m) \leftarrow \varphi$. In the next section, we will be more precise about which kinds of preconditions φ we allow. For now, we will consider the simple case of φ being a conjunction $b_1 \wedge \cdots \wedge b_n$ of propositions. The precondition φ may also be absent, in which case the event is called *unconditional*.

The language *Conditional Probabilistic Event Logic*, or *CP-logic* for short, now consists of sets of such CP-events. Such a set is called a *CP-theory*. By examining our intuitions about how different CP-events should interact with each other, the following two fundamental principles can be isolated. The first is that of *independent causation*. It states that every CP-event represents an independent causal event; in other words, the outcome of one event might affect whether or not some other event will happen, but does not have an influence on what the outcome of such an event will be, should it in fact happen. This principle is precisely what allows causality to act as the basis for a stable and modular view of the world. Moreover, it also implies that, to a certain extent, the order in which CP-events happen is irrelevant. The second principle, which we call the principle of *no deus ex machina effects*, is that nothing happens without a cause, i.e., everything remains false unless there is a cause for it to become true and something cannot cause itself. This is a fundamental principle of causal reasoning and will turn out to be especially vital in the presence of cyclic causal relations.

In the next section, we will construct a semantics for CP-logic based on these two principles. In this semantics, a CP-theory constructively defines a unique probability distribution over interpretations of the propositions. At each step, this constructive process simulates a single CP-event. Such a simulation derives proposition h_i with probability α_i, but can only be performed if all the propositions $b_1, \ldots, b_n$ have already been derived. Moreover, each event can occur at most once. This process will start from the empty set, i.e., initially nothing has been derived yet, and will end once there are no more CP-events left to simulate. The probability of an interpretation, then, is the sum of the probabilities of all possible derivations of this interpretation. It can be shown that the precise order in which CP-events are simulated does not matter, i.e., all sequences

will construct the same distribution. This follows from the principle of independent causation, together with the fact that, by only considering preconditions that are conjunctions of propositions, we have ensured the *monotonicity* of such sequences of simulations, i.e., the fact that if, at a certain time, all preconditions to a CP-event are satisfied, they will remain satisfied. Moreover, the "no deus ex machina"-principle is clearly incorporated in this semantics, because a proposition is only derived if it is caused by a CP-event with satisfied preconditions.

To sketch some of the interesting properties of this language, we consider two ways in which a person might get infected by the HIV virus: sexual intercourse with an infected partner and blood transfusion. For concreteness, assume that the probability of contracting HIV from an infected partner is 0.6 and that the probability of contracting it through a blood transfusion is 0.01. For the case of two partners a and b, of which only a has received a blood transfusion, we can model this example by the following CP-theory: $\{(hiv(a) : 0.6) \leftarrow hiv(b).\ (hiv(b) : 0.6) \leftarrow hiv(a).\ (hiv(a) : 0.01).\}$

As this example shows, the principle of independent causation makes it easy to represent the relation between an effect and a number of independent causes for this effect in a compact, clear and modular way, with each possible cause corresponding to a single CP-event. Moreover, this principle also makes the representation elaboration tolerant, in the sense that adding (or removing) an additional cause simply corresponds to adding (removing) a single rule. For instance, if b undergoes a blood transfusion as well, we only need to add a rule $(hiv(b) : 0.01)$. Because of the "no deus ex machina"-principle, the cyclic causal relation between $hiv(a)$ and $hiv(b)$ can be represented in precisely the same way as an acyclic one. Indeed, the first two rules will act as one would expect from such a causal loop: if neither a nor b have been infected by an external cause, then both are not infected, i.e., by itself such a loop does not cause anything; if precisely one of a and b has been infected by an external cause, then the probability of the other also being infected is 0.6. Another useful consequence of the "no deus ex machina"-principle is that domains can be represented in a compact way, because cases in which some proposition is not caused can simply be ignored. Indeed, we do not need to mention that without either intercourse with an infected partner or blood transfusion, an HIV infection is impossible. Further on in this paper, we will compare CP-logic to Bayesian networks and address these issues in more detail.

The contributions of this paper are the following. We introduce the concept of a CP-event and the principles of independent causation and no deus ex machina effects. We then use these to define the language of CP-logic. We show how this logic relates to Bayesian networks and illustrate some advantages of making causal events explicit. We relate our logic to logic programming based approaches to probabilistic modeling. This result provide additional motivation for these approaches and helps to clarify their knowledge representation methodology. It also allows us to consider a causal interpretation for disjunctive and normal logic programs. We compare this to McCain & Turner's causal theories.

Conditional Probabilistic Event Logic

The full syntax of CP-logic extends the one presented in the introduction in two ways. Firstly, it allows variables to be used to represent a set of CP-events by a single rule. Secondly, arbitrary first-order logic formulas can be used as preconditions to CP-events. More concretely, a CP-theory consists of a set of rules of the form: $(A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \varphi$. Here, the A_i's are atoms which may contain variables, φ is a first-order formula, and the α_i's are numbers between 0 and 1, s.t. $\sum_{i=1}^{n} \alpha_i \leq 1$. We distinguish between two different kinds of variables: those that are bound by a quantifier in φ and those that are free. The free variables $free(r)$ of a rule r are treated as place holders for ground terms. Concretely, we will consider a rule r as an abbreviation for the *grounding* (w.r.t. the Herbrand universe) of this rule, i.e., the set of all rules r' that can be derived from r by replacing all the variables $free(r)$ by ground terms. We reserve the term CP-event to refer to a rule in which no free variables appear. As such, a rule with free variables is simply a convenient way of representing a set of similarly structured CP-events. In formal discussions, we will always assume that CP-theories have already been grounded, i.e., we restrict attention to sets of rules of the form:

$$(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow \varphi. \qquad (1)$$

where the h_i's are ground atoms and φ is a formula with $free(\varphi) = \{\}$. For a rule r of form (1), we use $Body(r)$ to refer to the formula φ. We use $Body_+(r)$ to denote the set of all atoms that occur only positively (i.e., within the scope of an even number of negations) in φ and $Body_-(r)$ to denote all atoms that occur negatively (in the scope of an odd number of negations). By $body_+(r)$ and $body_-(r)$ we mean the set of all ground atoms belonging to the grounding of an atom in, respectively, $Body_+(r)$ and $Body_-(r)$. The set of pairs $\{(h_1, \alpha_1), \ldots, (h_n, \alpha_n)\}$ will be denoted by $head(r)$. By $head_{At}(r)$ we mean the set $\{h_1, \ldots, h_n\}$ of all atoms appearing in the head of r. Rules of the form $(h : 1) \leftarrow \varphi$ are called *deterministic*. We also write such a rule as $h \leftarrow \varphi$.

We now discuss how the definition of the semantics of CP-logic outlined in the introduction can be adapted to cope with arbitrary formulas as preconditions. The main difficulty here lies in the presence of negation. Indeed, this causes the previously mentioned monotonicity property to be lost, i.e., it will no longer be the case that if the preconditions to a CP-event are satisfied at a certain point in time, they are guaranteed to remain satisfied. As a consequence of this, the order in which CP-events are executed might matter. Consider, for instance, the following example: $\{p \leftarrow \neg q.\ \ q.\}$ If the first

event is executed first, then p will hold; if the second event is executed first, then p will not. So, negation introduces an ambiguity into the language and this needs to be resolved. Moreover, to avoid making the language too cumbersome, we would like to resolve it without forcing the user to explicitly specify an order in which events must happen. The most reasonable convention that can be assumed to solve this problem is that a literal $\neg r$ refers to the truth of r *after* all executable events that might cause r have been executed. As such, in our semantics, the simulation of such an event will be delayed as long as there are still executable events left that might cause r.

To ensure that this is indeed possible, we require CP-theories to be *stratified*, meaning that there has to exist a way of assigning to each ground atom p a level $\lambda(p) \in \mathbb{N}$, s.t. for each rule r and $h \in head_{At}(r)$, for all $b \in body_+(r)$, $\lambda(h) \geq \lambda(b)$, while for all $b \in body_-(r)$, $\lambda(h) > \lambda(b)$. The level of a rule r is defined as the minimum of the levels of the atoms in $head_{At}(r)$. Now, by executing CP-events with a lower level first, we can make sure that, by the time we need to decide whether a precondition $\neg r$ of some event holds, all executable events that might cause r have already been executed. From now on, we will restrict our attention to CP-theories which admit such a stratification.

We will now formally define the semantics of a CP-theory C. We use the mathematical structure of a *probabilistic transition system*. This is a tree structure $\mathcal{T}$, in which every edge is labeled with a probability. To each node c, we associate a Herbrand interpretation $I(c)$ for the alphabet of C. Formally, such an interpretation is simply a set of ground atoms. A node c *executes* a rule r of form (1) if c has as its children precisely nodes $c_0, c_1, \ldots, c_n$, where $I(c_0) = I(c)$ and, for all $i > 0$, $I(c_i) = I(c) \cup \{h_i\}$; the probability of the edge (c, c_0) is the probability with which none of the h_i's is caused, i.e., $1 - \sum_{1 \leq i \leq n} \alpha_i$, and, for $i > 0$, the probability of (c, c_i) is α_i. A rule r is *executable* in a node c if $body(r)$ holds in $I(c)$ and no ancestor of c already executes r. A probabilistic transition system $\mathcal{T}$ *runs* a CP-theory C iff:

- For the root r of $\mathcal{T}$, $I(r)$ is $\{\}$;
- For every node c of $\mathcal{T}$, either c executes an executable rule r, s.t. no executable rule r' has a lower level than that of r, or no rules are executable in c and c is a leaf;

A system $\mathcal{T}$ defines a probability distribution over its leaves: the probability of a leaf c is the product of the probabilities of all edges in the path from the root to c. From this, a probability distribution $\pi_{\mathcal{T}}$ over interpretations can be derived, by defining the probability $\pi_{\mathcal{T}}(I)$ of an interpretation I as the sum of the probabilities of all leaves c for which $I(c) = I$.

It can be shown that every system $\mathcal{T}$ that runs a CP-theory C defines the same probability distribution, i.e., for all such $\mathcal{T}$ and $\mathcal{T}'$, $\pi_{\mathcal{T}} = \pi_{\mathcal{T}'}$. While we do not have space to present a formal proof of this, the following informal argument should show that, if in a certain node n there are a number of different rules that might be executed, it does not matter which one is chosen first. If some r is executed first, then the existence of the stratification for C ensures that, for each of the children n' of n, all other rules r' that were executable in n are still executable in n', and, in fact, will remain this way until they are executed. Because every executable rule must eventually be executed, this shows that in every branch originating from n, every r' that was originally executable in n will actually be executed. Now, let us consider the sum of the probabilities of all branches going through n, in which an atom h, with $(h, \alpha) \in head(r')$, is derived as a result of executing r'. It can be checked that, because $\mathcal{T}$ is constructed in such a way that the sum of the labels of all edges that leave a node is always equal to 1, this probability will be α times the sum of the probabilities of all branches going through n, which is precisely what it would have been if we had chosen to execute r' in n. A formal proof can be found in (Vennekens *et al.* 2006).

The formal semantics of a CP-theory C is now defined as precisely this unique distribution, which we denote as π_C.

Bayesian networks in CP-logic

CP-logic can express the same kind of knowledge as expressed in a Bayesian network. The semantics of Bayesian networks (Pearl 1988) states that a probability distribution is a model of a network with graph $\langle N, E \rangle$ iff (1) the conditional probabilities it determines are the same as the appropriate entries in the various tables and (23) the value of a node n is probabilistically independent of the value of all nodes m, s.t. there is no path from n to m in E, given a value for the parents of n in E. If a distribution π satisfies condition (2) w.r.t. some binary relation E, we say that π is *Bayesian w.r.t.* E.

We now investigate whether a similar relation exists for the semantics π_C of a CP-theory C. In other words, we want to answer the question: "When does observing the truth value of a ground atom p give us direct information about the truth of some other atom q?" By the word "direct" in this sentence, we mean that this information is not mediated by some other atom r. It turns out that this happens in two cases.

- An atom p has a *direct causal influence* on an atom q if p is a precondition to a CP-event that might cause q, i.e., if $\exists r \in C, q \in head_{At}(r)$ and $p \in body(r)$. In this case, whether p holds affects whether one of the CP-events that might cause q is executed and, therefore, learning the truth of p clearly might affect the probability of q.
- Atoms p and q are *alternatives* iff p and q appear in the head of the same rule, i.e., iff $\exists r \in C : p, q \in head_{At}(r)$ and $p \neq q$. In this case, p and q are alternative outcomes of the same CP-event. As such,

learning that p holds will decrease the probability that q holds, and vice versa.

Let us now say that an atom p *directly affects* an atom q if one of these conditions holds, i.e., if $\exists r \in C : q \in head_{At}(r)$ and $p \in body(r) \cup (head_{At}(r) \setminus \{p\})$. We have shown in a recent technical report [reference omitted for anonymity] that the semantics π_C of a CP-theory is Bayesian w.r.t. this "directly affects"-relation. From this, a way of representing Bayesian networks in CP-logic can be derived. We first show how to represent the following Bayesian network:

	B,E	B,¬E	¬B,E	¬B,¬E
A	0.9	0.8	0.8	0.1

E	0.2
B	0.1

This can be modeled by the following CP-theory.

$$(bg : 0.1). \qquad (eq : 0.2).$$
$$(al : 0.9) \leftarrow bg \wedge eq. \quad (al : 0.8) \leftarrow \neg bg \wedge eq.$$
$$(al : 0.8) \leftarrow bg \wedge \neg eq. \quad (al : 0.1) \leftarrow \neg bg \wedge \neg eq.$$

Because this network contains only boolean nodes, we can translate each random variable into a single propositional symbol. As a result, in the CP-theory, all rules have just a single atom in their head; in the terminology defined above, no atoms are alternatives. The structure of the Bayesian network is now mirrored by the structure of the rules: the bodies of the rules for *earthquake* and *burglary* are empty, while the rules for *alarm* have both *burglary* and *earthquake* in their body. As such, this CP-theory expresses the same probabilistic independencies as the Bayesian network.

For nodes with a domain of more than two possible values, the situation is more complex. Indeed, such a node no longer corresponds to a single proposition, but rather to set of propositions. For instance, let us suppose that the alarm can be in three states: off (f), on (n), or disabled (d). We would then get rules of the following form:

$$(al(n) : 0.8) \vee (al(f) : 0.1) \vee (al(d) : 0.1) \leftarrow bg \wedge eq.$$
$$(al(n) : 0.7) \vee (al(f) : 0.1) \vee (al(d) : 0.1) \leftarrow bg \wedge \neg eq.$$
$$(al(n) : 0.7) \vee (al(f) : 0.1) \vee (al(d) : 0) \leftarrow \neg bg \wedge eq.$$
$$(al(n) : 0.2) \vee (al(f) : 0.8) \vee (al(d) : 0) \leftarrow \neg bg \wedge \neg eq.$$

This principle generalizes to a method of translating arbitrary networks to CP-logic.

The Role of Causality in CP-logic

In this section, we examine the differences between Bayesian networks and CP-logic in more detail.

Independent Causes

As already discussed, CP-logic incorporates the same kind of probabilistic independencies as Bayesian networks do. However, the principle of independent causation also allows a different kind of independence to be expressed, namely that between different causes for the same effect. In the introduction, we illustrated this by considering a number of different causes for HIV infection. We return to that example in the next section, where it will be used to illustrate the fact that our methodology for representing independent causation also applies when cyclic causal relations are involved. In this section, we focus on an example containing only acyclic causality:

Example 1. *Consider a game of Russian roulette with two guns, one in the player's right hand and one in his left. Each of the guns is loaded with a single bullet. What is the probability of the player dying?*

Firing a gun causes death with probability $\frac{1}{6}$. In CP-logic, we write: $(death : \frac{1}{6}) \leftarrow fire(Gun)$.This rule is all that is needed to model the operation of the guns. Indeed, if our alphabet also includes constants $left\_gun$ and $right\_gun$, then, after grounding, we get the following rules: $\{(death : \frac{1}{6}) \leftarrow fire(left\_gun).\ (death : \frac{1}{6}) \leftarrow fire(right\_gun).\}$. In words, "firing the left gun" and "firing the right gun" are two independent causes for death and each has a probability of $\frac{1}{6}$ of actually causing death. In a Bayesian network, this relation would typically be expressed as follows (the numbers can be computed by applying *noisy-or*[1] to the multiset of the probabilities with which the guns that are fired each cause death):

	l,r	¬l,r	l,¬r	¬l,¬r
death	11/36	1/6	1/6	0

In CP-logic, unlike in Bayesian networks, independence between different causal mechanisms is a *structural* property, rather than a quantitative one. From a knowledge representation point of view, there are a number of different reasons why this is useful. Firstly, it makes such properties more obvious. Indeed, the fact that $fire(left\_gun)$ and $fire(right\_gun)$ are independent possible causes of $death$ is clear from the fact that these atoms do not appear in the body of the same rule together. Secondly, independence between causes is more robust to changes in the specification of a problem than quantitative knowledge usually is. For instance, if we were to find out that one of the guns has a mechanical defect, making the probability of the bullet being in front of the hammer not $\frac{1}{6}$ precisely, but $\frac{11}{60}$ instead, then this would not affect the independence between the two possible causes for death. Thirdly, a large part of Pearl's book (Pearl 2000) is concerned with various manipulations of causal models. These are important, for instance, to deal with counterfactuals or concepts such as the "actual cause" of some effect. Typically, such a manipulation consists of preempting the normal relation between a node and its parents and replacing it by an entirely new relation. With the more fine-grained structure imposed by CP-logic, a number of new, interesting manipulations become possible. For instance,

[1] The *noisy-or* maps a multiset S of probabilities to $1 - \prod_{p \in S}(1 - p)$.

one could now consider the situation that would result by only inhibiting one of the causal mechanisms that might cause a particular atom or by adding a new such causal mechanism. This also improves the *elaboration tolerance* of the representation. For instance, if it is also possible that the player dies of a heart attack, we can simply include a CP-event "$(death : 0.2)$.", which might lead to death without firing any guns. Finally, like other qualitative properties, knowledge about these independencies often originates from an expert's background knowledge about the domain, instead of being derived from a dataset. For instance, when performing parameter learning in a Machine Learning setting, this is an important consideration.

Cyclic Causal Relations

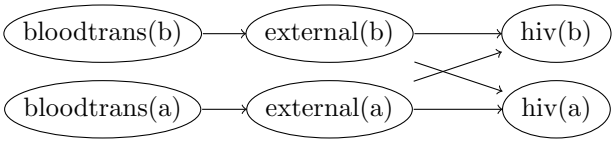
We now revisit the following example from the introduction:

Example 2. *There are two ways of getting infected by the HIV virus. Firstly, there is a probability of 0.01 of contracting the virus through a blood transfusion. Secondly, one might get infected by an already infected sexual partner. The probability of an infected person infecting his/her partner is 0.6.*

$$(hiv(X) : 0.6) \leftarrow hiv(Y) \wedge intercourse(X, Y).$$
$$(hiv(X) : 0.01) \leftarrow blood\_transfusion(X).$$

Because the *intercourse*-relation is symmetric, this example leads naturally to cyclic causal relations. Because of the "no deus ex machina"-principle of CP-logic, cyclic relations can be modeled in the same way as acyclic ones. To make the discussion more concrete, suppose that a and b are the only persons we wish to consider and that these have intercourse. For simplicity, we ignore the *intercourse* and $blood\_transfusion$ predicates and simply assume that our grounding process knows who have intercourse and who have received blood transfusions. The first rule of the CP-theory will lead to the grounding $\{(hiv(a) : 0.6) \leftarrow hiv(b).\ (hiv(b) : 0.6) \leftarrow hiv(a).\}$. Let us examine how these rules act when part of a larger theory. Firstly, if these two rules were to constitute the entire theory, then, because of the "no deus ex machina"-principle, neither partner would be infected. Now, if partner a undergoes a blood transfusion, the rule "$(hiv(a) : 0.01)$." would also appear, adding this as an additional cause for $hiv(a)$. In this case, a has a non-zero probability (namely 0.01) of being infected by an external cause and, therefore, b also has a non-zero probability (0.01×0.6) of infection. If both a and b undergo a blood transfusion, then the probability of, for instance, $hiv(a)$ will be higher still, because there are now two independent causes for $hiv(a)$: a could have gotten infected by a transfusion, but also because b was first infected by a transfusion and infected a in turn.

This causal loop can be represented in a Bayesian network by introducing, for every $hiv(x)$ in this loop, a new node $external(x)$ to represent the event that x has gotten infected by some external (i.e., not part of the loop) cause. These are then connected as follows:



	bt(a)	¬bt(a)
e(a)	0.01	0

	bt(b)	¬bt(b)
e(b)	0.01	0

	e(a),e(b)	e(a),¬e(b)	¬e(a),e(b)	¬e(a),¬e(b)
hiv(a)	1	1	0.6	0

	e(a),e(b)	e(a),¬e(b)	¬e(a),e(b)	¬e(a),¬e(b)
hiv(b)	1	0.6	1	0

The Absence of Causes

Another consequence of the "no deus ex machina"-principle is that CP-logic does not require cases in which an effect is not caused to be mentioned at all. Obviously, this can make representations more compact. This feature is made more powerful by the fact that CP-theories may contain negation, which allows the falsity of an atom, i.e., the absence of a cause for this atom, to act as a cause for other atoms. To illustrate, we consider the well-known dice game of craps. Similar phenomena occur, e.g., when representing *inertia axioms*, i.e., rules stating that some property persists unless these is a cause for it not to.

Example 3. *In craps, one keeps on rolling a pair of dice until one either wins or loses. In the first round, one immediately wins by rolling* 7 *or* 11 *and immediately loses by rolling* 2,3, *or* 12. *If any other number is rolled, this becomes the player's "box point". The game then continues until either the player wins by rolling the box point again or loses by rolling* 7.

$$(roll(T+1, 2) : 1/36) \vee \cdots \vee (roll(T+1, 12) : 1/36) \leftarrow \neg(win(T) \vee lose(T)).$$
$$win(1) \leftarrow roll(1, 7) \vee roll(1, 11).$$
$$lose(1) \leftarrow roll(1, 2) \vee roll(1, 3) \vee roll(1, 12).$$
$$boxpoint(X) \leftarrow roll(1, X) \wedge \neg win(1) \wedge \neg lose(1).$$
$$win(T) \leftarrow boxpoint(X) \wedge roll(T, X) \wedge T > 1.$$
$$lose(T) \leftarrow roll(T, 7) \wedge T > 1.$$

(In principle, this CP-theory would lead to an infinite grounding. While it is possible to define a semantics for CP-logic that is able to correctly handle such theories, this is beyond the scope of this paper. Because this issue is orthogonal to the points we want to discuss here, we simply assume that a certain maximum number of throws, i.e., an upper bound for the variable T, has been fixed up front and will be taken into account when performing the grounding of this theory.)

In this CP-theory, we only specify when the game is won or lost and use negation to express that, as long as neither happens, the game carries on. In Bayesian networks, there is no real way of ignoring irrelevant cases. Instead, there will be a probability of zero in the conditional probability table. For this game, we could use

variables $roll_t$, representing the outcome of a certain roll (with domain 2 through 12), and *bp* representing the box point (with possible values 4,5,6,7,8,9 or 10), that influence the state s_t of the game at time t as follows:

	$(bp, roll_t)$						
s_t	$(4,2)$	$(4,3)$	$(4,4)$	$(4,5)$	$(4,6)$	$(4,7)$	$\cdots$
win	0	0	1	0	0	0	$\cdots$
lose	0	0	0	0	0	1	$\cdots$

Logic Programs with Annotated Disjunctions

Logic Programs with Annotated Disjunctions (LPADs) are a probabilistic logic programming language, that was conceived in (Vennekens, Verbaeten, & Bruynooghe 2004) as a straightforward extension of logic programs with probability. In this section, we relate LPADs to CP-logic. This achieves the following goals:

- We can clarify the position of CP-logic among related work, such as Poole's Independent Choice Logic and McCain and Turner's causal theories.
- We gain additional insight into a number of probabilistic logic programming languages, by showing that theories in these languages can be seen as descriptions of causal processes. Moreover, as we will discuss in the next section, this also leads to an interesting way of looking at normal and disjunctive logic programs.
- Probabilistic logic programming languages are usually motivated in a bottom-up way, i.e., along the following lines: "Logic programs are a good way of representing knowledge about relational domains, probability is a good way of representing knowledge about uncertainty; therefore, a combination of both should be useful for modeling uncertainty in a relational domain." Our results provide an additional top-down motivation, by showing that these languages are a natural way of representing causal processes.

We first recall the formal definition of LPADs from (Vennekens, Verbaeten, & Bruynooghe 2004). An LPAD is a set of rules $(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow l_1 \wedge \cdots \wedge l_n$, where the h_i are atoms and the l_j literals. As such, LPADs are a syntactic sublogic of CP-logic. However, their semantics is defined quite differently. Every rule of the above form represents a probability distribution over the set of logic programming rules $\{\text{“}h_i \leftarrow l_1 \wedge \cdots \wedge l_n\text{”} \mid 1 \leq i \leq n\}$. From these distributions, a probability distribution over logic programs is then derived. To formally define this distribution, we introduce the following concept of a *selection*. In this definition, we use the notation $head^*(r)$ to denote the set of pairs $head(r) \cup \{(\emptyset, 1 - \sum_{(h:\alpha)\in head(r)} \alpha)\}$, where $\emptyset$ represents the possibility that none of the h_i's are caused by the rule r.

Definition 1 (C-selection). Let C be an LPAD. A *C-selection* is a function σ from C to $\bigcup_{r\in C} head^*(r)$, such that for all $r \in C$, $\sigma(r) \in head^*(r)$. By $\sigma^h(r)$ and $\sigma^\alpha(r)$ we denote, respectively, the first and second element of the pair $\sigma(r)$.

The probability $\pi(\sigma)$ of a selection σ is now defined as $\prod_{r\in C} \sigma^\alpha(r)$. By C^σ we denote the logic program $\{\text{“}\sigma^h(r) \leftarrow body(r)\text{”} \mid r \in C \text{ and } \sigma^h(r) \neq \emptyset\}$. Such a C^σ is called an *instance* of C. These instances are interpreted according to the well-founded model semantics (Van Gelder, Ross, & Schlipf 1991). In general, the well-founded model $wfm(P)$ of a program P is a pair (I, J) of interpretations, where I contains all atoms that are certainly true and J contains atoms that might possibly be true. If $I = J$, the model is said to be *two-valued*. Intuitively, if $wfm(P)$ is two-valued, then the truth of all atoms can be decided, i.e., everything that is not false can be derived. In the semantics of LPADs, we want to ensure that all uncertainty is expressed by means of the annotated disjunctions. In other words, given a specific selection, there should no longer be any uncertainty. We impose the following criterion.

Definition 2 (Soundness). An LPAD C is *sound* iff all instances of C have a two-valued well-founded model.

For such LPADs, the following semantics can now be defined.

Definition 3 (Instance based semantics μ_C). Let C be a sound LPAD. For an interpretation I, we denote by $W(I)$ the set of all C-selections σ for which $wfm(C^\sigma) = (I, I)$. The *instance based semantics* μ_C of C is the probability distribution on interpretations, that assigns to each I the probability $\sum_{\sigma\in W(I)} \pi(\sigma)$.

Now, the key result of this section is that this instance based semantics coincides with the semantics defined previously for CP-logic.

Theorem 1. Let C be a stratified CP-theory. Then C is also a sound LPAD and, moreover, for each interpretation J, $\mu_C(J) = \pi_C(J)$.

We remark that it is not the case that every sound LPAD is also a valid CP-theory. In other words, there are some sound LPADs that do not seem to represent a causal process.

In (Vennekens, Verbaeten, & Bruynooghe 2004), LPADs are compared to a number of different probabilistic logic programming formalisms. For instance, it was shown that this logic is very closely related to Poole's Independent Choice Logic. Because of the above theorem, these comparisons carry over to CP-logic.

CP-logic and Logic Programming

In this section, we examine some consequences of the results of the previous section from a logic programming point-of-view.

Disjunctive logic programs

In probabilistic modeling, it is often useful to consider the structure of a theory separately from its probabilistic parameters. Indeed, for instance, in machine learning, the problems of structure learning and parameter learning are two very different tasks. If we consider only the structure of a CP-theory, then, syntactically speaking, we end up with a *disjunctive logic program*[2], i.e., a set of rules $h_1 \vee \cdots \vee h_n \leftarrow \varphi$. Let us now consider the class of all CP-theories C that result from adding probabilities α_i to each rule, in such a way that, for every rule, $\sum \alpha_i = 1$. Every probability distribution π_C defined by such a C induces a possible world semantics, namely the set of interpretations I for which $\pi_C(I) > 0$. This set of possible worlds does not depend on the precise values of the α_i, i.e., it is the same for all CP-theories C in this class. As such, it captures precisely the structural information in such a CP-theory.

From the point of view of disjunctive logic programming, this set of possible worlds can be seen as an alternative semantics for such a program. Under this semantics, the intuitive reading of a rule should be: "if φ holds, there will be a non-deterministic event, that causes precisely one of $h_1, \ldots, h_n$." Clearly, this is a different informal reading than is used in the standard stable model semantics for disjunctive programs (Przymusinski 1991). Indeed, under our reading, a rule corresponds to a causal event, whereas, under the stable model reading, it is supposed to describe an aspect of the reasoning behaviour of a rational agent. This difference also manifests itself in the resulting formal semantics. Consider, for instance, the disjunctive program $\{p \vee q.\ p.\}$. To us, this program describes a set of two non-deterministic events: One event causes either p or q and another event always causes p. This might, for instance, correspond to the following story: "Someone is going to shoot a gun at Bob and this will either cause Bob's death or a hole in the wall behind Bob. Bob has also just ingested a lethal dose of poison and this is going to cause Bob's death." Our formal semantics reflects this interpretation, by considering both the interpretation $\{p\}$ (Bob is dead and there is no hole in the wall) and $\{p, q\}$ (Bob is dead and there is a hole in the wall) to be possible. Under the stable model semantics, these rules describe beliefs of a rational agent: The agent believes either p or q and the agents believes q. This interpretation might correspond to the following story: "I know that someone was going to shoot a gun at Bob, which would either result in Bob's death or in a hole in the wall. Moreover, I also learn that Bob is dead." In this case, I would have no reason to believe there might be a hole in the wall. Indeed, the only stable model is $\{p\}$.

CP-logic treats disjunction in a fundamentally different way than the stable semantics. Interestingly, the *possible model semantics* (Sakama & Inoue 1994) for disjunctive programs is very similar to our treatment. Indeed, it consists of the stable models of instances of a program. Because, as shown in the previous section, the semantics of CP-logic considers the well-founded models of instances, these two semantics are very closely related. Indeed, for a large class of programs, including all stratified ones, they coincide completely.

[2] In most of the literature, the bodies of the rules of a disjunctive logic program must be conjunctions of literals. For our purposes, however, this restriction is not relevant.

Normal logic programs

A normal logic program P is a set of rules $h \leftarrow \varphi$, with h an atom and φ a formula. If P is stratified, then, at least syntactically, it is also a CP-theory. Its semantics π_P assigns a probability of 1 to a single interpretation and 0 to all other interpretations. Moreover, the results from the previous section tell us that the interpretation with probability 1 will be precisely the well-founded model of P. As such, a logic program under the well-founded semantics can be viewed as a description of causal information about a deterministic process. Concretely, we can read a rule $h \leftarrow \varphi$ as: "if φ holds, there will be a deterministic event, that causes h."

This observation exposes an interesting connection between logic programming under the well-founded semantics and causality. Such a connection helps to explain, for instance, the usefulness of this semantics in dealing with recursive ramifications when reasoning about actions (Denecker, Theseider-Dupré, & Belleghem 1998). Moreover, there is also an interesting link here to the language of *ID-logic* (Denecker & Ternovska 2004). This is an extension of classical logic, that uses logic programs under the well-founded semantics to represent inductive definitions. Inductive definitions are a well-known mathematical construct, where a concept is defined in terms of itself. In mathematical texts, such a definition should be accompanied by a well-founded order, over which the induction happens, e.g., the well-known inductive definition of the satisfaction relation $\models$ of classical logic is a definition over the length of formulas. One of the key observations that underlie ID-logic is the fact that if such an order is not explicitly given, one can still be derived from the rule-based structure of a definition. This derived order is precisely the order imposed by the well-founded semantics. There is an obvious parallel here to the role of time in CP-logic: a complete description of a process should specify when events happen; however, if this information is not explicitly given, the order of events can still be derived from the rule-based structure of a CP-theory. It is interesting that the same mathematical construct of the well-founded semantics can be used to derive both the well-founded order for an inductive definition and the temporal order for a set of CP-events. This observation seems to imply that an inductive definition is nothing more than a representation of a causal process, that takes place in the domain of mathematical objects.

McCain and Turner's causal theories

In this section, we compare the treatment of causality in CP-logic to McCain and Turner's *causal theories* (McCain & Turner 1996). A causal theory is a set of rules of the form $\varphi \Leftarrow \psi$, where φ and ψ are propositional formulas. The semantics of such a theory T is defined as follows. An interpretation I is a model of T iff I is the *unique* classical model of the theory $T^I = \{\varphi \mid$ there exists a rule $\varphi \Leftarrow \psi$ in T such that $I \models \psi\}$. This semantics is based on the principle of *universal causation*, which states that: "every fact that obtains is caused" (McCain & Turner 1996). We now compare this language to deterministic CP-logic, i.e., CP-logic in which every CP-event causes one atom with probability 1. The most obvious difference concerns the fundamental knowledge representation methodology of these logics. In CP-logic, a proposition represents a property that is false unless there is a cause for it to be true. For McCain and Turner, however, truth and falsity are completely symmetric, i.e., not only is a property not true unless there is a cause for it to be true, but a property is also not false unless there is a cause for it to be false. It is up to the user to make sure there is always a cause for either falsity or truth. For instance, the CP-theory $\{p \leftarrow \neg q\}$ has $\{p\}$ as its model, while the causal theory $\{p \Leftarrow \neg q\}$ has no models, because neither q nor $\neg q$ is caused. The CP-logic view that falsity is the natural state of an atom can be simulated in causal theories, by adding a rule $\neg p \Leftarrow \neg p$ for every atom p. Essentially, this says that $\neg p$ is in itself reason enough for $\neg p$. Let C' be the result of adding such rules to some original CP-theory C. As shown in (McCain 1997), the models of C' are all interpretations I that consist of all heads of rules $r \in C$, for which $I \models \mathit{body}(r)$. In logic programming terms, these are the *supported models* of C, i.e., fixpoints of the immediate consequence operator T_C.

The difference such a CP-theory C and its corresponding causal theory C' is, therefore, precisely the difference between the well-founded model semantics and supported model semantics. It is well-known that this lies in the treatment of loops. In our context, it can be traced back to the fundamental principles of these logics. McCain and Turner's principle of "universal causation" states that everything that holds must have a cause. This is a weaker principle than our principle of no deus ex machina effects, which states that every true proposition must have a cause *and* that something cannot cause itself. Indeed, the CP-theory $\{p \leftarrow p\}$ has $\{\}$ as its model, whereas the causal theory $\{p \Leftarrow p\}$ has $\{p\}$ as its model. In other words, in McCain and Turner's theories, it can be stated that a certain atom might be true "on its own", i.e., without any additional causal explanation being required. This can be useful to incorporate exogenous actions into a theory, i.e., actions that can simply happen, without any part of the model describing why they happen. These currently cannot be represented in CP-logic. On the other hand, McCain and Turner's approach to self-causation does not allow them to directly represent cyclic causal relations of the kind appearing in our HIV example.

Related work

The correspondence to LPADs establishes a relation between CP-logic and probabilistic logic programming formalisms. In (Vennekens & Verbaeten 2003), a detailed comparison is made between LPADs and a number of such approaches. At the formal level, these comparisons carry over to CP-logic. Here, we briefly discuss some of these formalisms, with a focus on the causal interpretation of CP-logic.

It has been shown in (Vennekens & Verbaeten 2003) that LPADs are very closely to the *Independent Choice Logic (ICL)* (Poole 1997). This language is based on abductive logic programs under the stable model semantics and was developed within the framework of decision theory. In (Finzi & Lukasiewicz 2003), a connection was made between ICL and Pearl's structural model approach to causality. One of the motivations for studying this relation is that it allows concepts such as "actual cause" and "explanation", that have been investigated by Halpern and Pearl in the context of structural models (Halpern & Pearl 2001), to be used in ICL. By linking ICL to CP-logic, we show that ICL can also be seen as a logic that incorporates our more fine-grained concept of causality, based on causal events. This raises the question of whether there are meaningful adaptations of Halpern and Pearl's definitions that can take into account this additional structure. This is an interesting avenue for future research.

Bayesian Logic Programs (BLPs) (Kersting & Raedt 2000) and *Relational Bayesian Networks (RBNs)* (Jaeger 1997) are two formalisms that aim at lifting the propositional formalism of Bayesian networks to a first-order representation. Both these language allow arbitrary functions to be used to compute certain probabilities. By using a *noisy-or*, certain properties of CP-logic can be simulated. For instance, in a Relational Bayesian Network, one would model the Russian roulette example by the probability formula $P(\mathit{death}) = \mathit{noisy\text{-}or}(\{1/6 \cdot \mathit{fire}(x) \mid x\})$. However, neither language offers a way of dealing with cyclic causal relations, other than an encoding similar to that for Bayesian networks.

Baral et al. introduced *P-log* (Baral, Gelfond, & Rushton 2004), a probabilistic extension of A-prolog. This language seems to be quite similar to CP-logic, even though it is somewhat broader in scope, being aimed at combining probabilistic and logical reasoning, rather than simply representing a probability distribution. As far as the representation of probability knowledge is concerned, P-log appears to be closer to Bayesian networks in the sense that it does not share CP-logic's focus on independent causation; instead, in every situation that might arise, there has to be precisely one statement that defines the probability of a certain effect in terms of all its possible causes. An

interesting feature of P-log is that it allows random selections from a dynamic range of alternatives, which is more flexible than the static enumerations of possible outcomes used in CP-logic.

Conclusions

We have investigated the role of causality in modeling probabilistic processes. To this end, we introduced the concept of a CP-event as a formal representation of the intuitive notion of a causal event. We presented a semantics for the language consisting of sets of such CP-events. This is based on two fundamental principles, that govern the interaction between different CP-events. The first is the principle of independent causation, which establishes the basic modularity of our causal view of the world. The second principle, that of no deus ex machina effects, captures the intuition that nothing happens without a cause, even in the presence of cyclic causal relations.

The direct representation of causal events in CP-logic turns out to have a number of interesting properties when compared to a Bayesian network style representation in terms of conditional probability. In particular, a new kind of independence, namely that between different causes for the same effect, emerges as a structural property, improving the elaboration tolerance of the representation. Moreover, cyclic causal relations can also be represented in a natural way and do not require any special treatment.

We have related CP-logic to a class of existing probabilistic logic programming approaches. This shows that these languages can also be seen as representations of causal events. Moreover, this also shows that CP-logic induces a possible world semantics for disjunctive logic programs, that is quite different from the standard stable model semantics, but very similar to the possible model semantics. Another consequence of this results is that normal logic programs under the well-founded semantics can be seen as a logic of deterministic causality, which points towards an interesting relation between causality and inductive definitions. We have compared this way of handling causality to the McCain and Turner's causal theories.

References

Baral, C.; Gelfond, M.; and Rushton, N. 2004. Probabilistic reasoning with answer sets. In *Proc. Logic Programming and Non Monotonic Reasoning, LPNMR'04*, 21–33. Springer-Verlag.

Denecker, M., and Ternovska, E. 2004. A logic of non-monotone inductive definitions and its modularity properties. In *Proc. 7th LPNMR*, volume 2923 of *LNCS*.

Denecker, M.; Theseider-Dupré, D.; and Belleghem, K. V. 1998. An inductive definition approach to ramifications. *Linköping EACIS* 3(7):1–43.

Finzi, A., and Lukasiewicz, T. 2003. Structure-based causes and explanations in the independent choice logic. In *Proc. Uncertainty in Artificial Intelligence (UAI)*.

Halpern, J., and Pearl, J. 2001. Causes and explanations: A structural model approach – part I: Causes. In *Proc. Uncertainty in Artificial Intelligence (UAI)*.

Jaeger, M. 1997. Relational bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*.

Kersting, K., and Raedt, L. D. 2000. Bayesian logic programs. In Cussens, J., and Frisch, A., eds., *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 138–155.

McCain, N., and Turner, H. 1996. Causal theories of action and change. In *Proc. 13th AAAI/8th IAAI*.

McCain, N. 1997. *Causality in Commonsense Reasoning about Actions*. Ph.D. Dissertation, University of Texas at Austin.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Poole, D. 1997. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94(1-2):7–56.

Przymusinski, T. C. 1991. Stable semantics for disjunctive programs. *New Generation Computing* 3/4:401–424.

Sakama, C., and Inoue, K. 1994. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of automated reasoning* 13(1):145–172.

Van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM* 38(3):620–650.

Vennekens, J., and Verbaeten, S. 2003. Logic programs with annotated disjunctions. Technical Report CW386, K.U. Leuven.

Vennekens, J.; Denecker, M.; ; and Bruynoogh, M. 2006. On the equivalence of Logic Programs with Annotated Disjunctions and CP-logic. Technical report, K.U. Leuven.

Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *Proc. 20th ICLP*, volume 3132 of *LNCS*. Springer.

2.9 Model and experimental study of causality ascriptions

Model and experimental studies of causality ascriptions

Jean-François Bonnefon
LTC-CNRS
5 allées Antonio Machado
31058 Toulouse Cedex 9, France
bonnefon@univ-tlse2.fr

Rui Da Silva Neves
DSVP
5 allées Antonio Machado
31058 Toulouse Cedex 9, France
neves@univ-tlse2.fr

Didier Dubois and **Henri Prade**
IRIT-CNRS
118 Route de Narbonne
31062 Toulouse Cedex, France
{dubois,prade}@irit.fr

Abstract

A model is defined that predicts an agent's ascriptions of causality (and related notions of facilitation and justification) between two events in a chain, based on background knowledge about the normal course of the world. Background knowledge is represented by nonmonotonic consequence relations. This enables the model to handle situations of poor information, where background knowledge is not accurate enough to be represented in, e.g., structural equations. Tentative properties of causality ascriptions are explored, i.e., preference for abnormal factors, transitivity, coherence with logical entailment, and stability with respect to disjunction and conjunction. Empirical data are reported to support the psychological plausibility of our basic definitions.

INTRODUCTION

Models of causal ascriptions crucially depend on the choice of an underlying representation for the causality-ascribing agent's knowledge. Unlike standard diagnosis problems (wherein an unobserved cause must be inferred from observed events and known causal links), causality ascription is a problem of describing as 'causal' the link between two observed events in a sequence. The first step in modeling causal ascription is to define causality in the language chosen for the underlying representation of knowledge. In this article, we define and discuss a model of causal ascription that represents knowledge by means of nonmonotonic consequence relations.[1] Indeed, agents often must cope with poor knowledge about the world, under the form of default rules. Clearly, this type of background knowledge is less accurate than, e.g., structural equations. It is nevertheless appropriate to predict causal ascriptions in situations of restricted knowledge. We first presents the logical language we will use to represent background knowledge. We then define our main notions of causality and facilitation ascriptions. Empirical data are reported to support the distinction between these two notions. Next, we establish some formal properties of the model. We then distinguish the notion of epistemic justification from that of causality. Finally, we relate our model to other works on causality in AI.

[1]This model was advocated in a recent workshop paper (Dubois & Prade 2005). The present paper is a slightly expanded version of (Bonnefon *et al.* 2006)

MODELING BACKGROUND KNOWLEDGE

The agent is supposed to have observed or learned of a sequence of events, e.g.: $\neg B_t$, A_t, B_{t+1}. This expresses that B was false at time t, when A took place, and that B became true afterwards ($t+1$ denotes a time point after t). There is no uncertainty about these events.

Besides, the agent maintains a knowledge-base made of conditional statements of the form 'in context C, if A takes place then B is generally true afterwards', or 'in context C, B is generally true'. These will be denoted by $A_t \wedge C_t \vdash B_{t+1}$, and by $C_t \vdash B_t$, respectively. (Time indices will be omitted when there is no risk of confusion.) The conditional beliefs of an agent with respect to B when an action A takes place or not in context C can take three forms: (i) If A takes place B is generally true afterwards: $A_t \wedge C_t \vdash B_{t+1}$; (ii) If A takes place B is generally false afterwards: $A_t \wedge C_t \vdash \neg B_{t+1}$; (iii) If A takes place, one cannot say whether B is generally true or false afterwards: $A_t \wedge C_t \nvdash B_{t+1}$ and $A_t \wedge C_t \nvdash \neg B_{t+1}$. [2]

We assume that the nonmonotonic consequence relation $\vdash$ satisfies the requirements of 'System P' (Kraus, Lehmann, & Magidor 1990); namely, $\vdash$ is reflexive and the following postulates and characteristic properties hold ($\models$ denotes classical logical entailment):

Left Equivalence	$E \vdash G$ and $E \equiv F$	imply	$F \vdash G$
Right Weakening	$E \vdash F$ and $F \models G$	imply	$E \vdash G$
AND	$E \vdash F$ and $E \vdash G$	imply	$E \vdash F \wedge G$
OR	$E \vdash G$ and $F \vdash G$	imply	$E \vee F \vdash G$
Cautious Monotony	$E \vdash F$ and $E \vdash G$	imply	$E \wedge F \vdash G$
Cut	$E \vdash F$ and $E \wedge F \vdash G$	imply	$E \vdash G$

In addition, we assume $\nvdash$ to obey the property of Rational Monotony, a strong version of Cautious Monotony(Lehmann & Magidor 1992):

Rational Monotony	$E \nvdash \neg F$ and $E \vdash G$	imply	$E \wedge F \vdash G$

Empirical studies repeatedly demonstrated (Benferhat, Bonnefon, & Da Silva Neves 2004; 2005; Da Silva Neves,

[2]Note that $\nvdash$ can be understood in two different ways: Either $A_t \wedge C_t \nvdash B_{t+1}$ just means that $A_t \wedge C_t \vdash B_{t+1}$ is not deducible from the agent's knowledge base, or it means that the agent really knows it is impossible to say that $A_t \wedge C_t \vdash B_{t+1}$ (this requires that the agent knows everything that generally holds concerning B when $A \wedge C$ is true). However, this difference is not crucial to our present purpose.

Bonnefon, & Raufaste 2002; Ford 2004; Pfeifer & Kleiter 2005) that System P and Rational Monotony provide a psychologically plausible representation of background knowledge and default inference. Arguments for using nonmonotonic logics in modeling causal reasoning were also discussed in the cognitive science literature (Shoham 1990).

ASCRIBING CAUSALITY OR FACILITATION

In the following definitions, A, B, C, and F are either reported actions or statements describing states of affairs, even though notations do not discriminate between them, since the distinction does not yet play a crucial role in the model. When nothing takes place, the persistence of the truth status of statements is assumed in the normal course of things, i.e., $B_t \vdash B_{t+1}$ and $\neg B_t \vdash \neg B_{t+1}$.

Assume that in a given context C, the occurrence of event B is known to be exceptional (i.e., $C \vdash \neg B$). Assume now that F and A are such that $F \wedge C \nvdash \neg B$ on the one hand, and $A \wedge F \wedge C \vdash B$ on the other hand; we will say that in context C, A together with F are perceived as the *cause* of B (denoted $C : A \wedge F \Rightarrow_{\text{ca}} B$), while F alone is merely perceived to have *facilitated* the occurrence of B (denoted $C : F \Rightarrow_{\text{fa}} B$).

Definition 1 (Facilitation ascription). *An agent that, in context C, learns of the sequence $\neg B_t$, F_t, B_{t+1} will judge that $C : F \Rightarrow_{\text{fa}} B$ if it believes that $C \vdash \neg B$, and that both $F \wedge C \nvdash \neg B$ and $F \wedge C \nvdash B$.*

Definition 2 (Causality ascription). *An agent that, in context C, learns of the sequence $\neg B_t$, A_t, B_{t+1} will judge that $C : A \Rightarrow_{\text{ca}} B$ if it believes that $C \vdash \neg B$, and $A \wedge C \vdash B$.*

Example 1 (Driving while intoxicated). *When driving, one has generally no accident, Drive $\vdash \neg$Accident. This is no longer true when driving while drunk, which is not as safe, Drive $\wedge$ Drunk $\nvdash \neg$Accident); moreover, fast driving while drunk will normally lead to an accident, Drive $\wedge$ Fast $\wedge$ Drunk $\vdash$ Accident. Suppose now that an accident took place after the driver drove fast while being drunk. Fast $\wedge$ Drunk will be perceived as the* cause *of the accident, while Drunk will only be judged as having* facilitated *the accident.*

Of course, in the above definition A can stand for any compound reported fact such as $A' \wedge A''$. Here, $C \vdash \neg B$, $F \wedge C \nvdash \neg B$, and $A \wedge C \vdash B$ must be understood as pieces of default knowledge used by the agent to interpret the chain of reported facts $\neg B_t$ (in context C), A_t, B_{t+1}, together with the persistence law $\neg B_t \wedge C \vdash \neg B_{t+1}$ (which can be deduced from $C \vdash \neg B_t$ and $\neg B_t \vdash \neg B_{t+1}$). In such a case, A_t may indeed appear to the agent as being a cause for the change from $\neg B_t$ to B_{t+1}, since $C \vdash \neg B_t$ and $A_t \wedge C \vdash B_{t+1}$ entail $A_t \wedge \neg B_t \wedge C \vdash B_{t+1}$.

Note that Def. 1 is weaker than saying F 'prevents' $\neg B$ from persisting: $\nvdash$ does not allow the jump from 'not having $\neg B$' to 'B'. In Def. 2, the fact that B is exceptional in context C precludes the possibility for C to be the cause of B – but not the possibility that $B \models C$, i.e., that C is a necessary condition of B. Thus, context can be a necessary condition of B without being perceived as its cause.

An interesting situation arises when an agent only knows that $C \vdash \neg B$ and $F \wedge C \nvdash \neg B$, and learns of the sequence of events $\neg B_t$ (in context C), F_t, B_{t+1}. Although this situation should lead the agent to judge that $C : F_t \Rightarrow_{\text{fa}} B_{t+1}$, it may be tempting to judge that $C : F_t \Rightarrow_{\text{ca}} B_{t+1}$, as long as no other potential cause reportedly took place. Another interesting situation arises when, in context C, an agent learns of the sequence $\neg B_t$, A_t, and B_{t+1}, while it believes that $\neg B_t \wedge C \vdash \neg B_{t+1}$, and that $A_t \wedge C \vdash \neg B_{t+1}$. Then the agent cannot consider that $C : A_t \Rightarrow_{\text{ca}} B_{t+1}$, and it may suspect some fact went unreported: finding about it would amount to a diagnosis problem.

When an agent believes that $C \vdash \neg B$ and $A \wedge C \nvdash \neg B$, and learns of the sequence of events $\neg B_t$, A_t, and $\neg B_{t+1}$, the agent would conclude that action A_t failed to produce its normal effect, for unknown reasons.

According to (von Wright 1963), an action caused p to be true if and only if either:

- p was false before the action, and had the action not been taken, p would not have become true, or
- the action maintains p true against the normal course of things, thus preventing p from becoming false.

The first situation straightforwardly relates to our definition. The second situation can also be represented in our setting: B_t is known to be true, and after A_t takes place B_{t+1} is still true, although in the normal course of things, had A not happened, B would have become false, i.e., $B_t \wedge C \vdash \neg B_{t+1}$. The agent knowledge also includes $B_t \wedge A_t \wedge C \vdash B_{t+1}$. Letting $C' = B_t \wedge C$, this can be rewritten $C' \vdash \neg B_{t+1}$ and $A \wedge C' \vdash B_{t+1}$, which is formally Definition 2.

EXPERIMENTAL TESTS

There is no previous empirical support to the distinction we introduce between ascriptions of cause and facilitation. To check whether this distinction has intuitive appeal to lay reasoners, we conducted two experiments in which we presented participants with different sequences of events. We assessed their relevant background knowledge, from which we predicted the relations of cause and facilitation they should ascribe between the events in the sequence. We then compared these predictions to their actual ascriptions.

Experiment 1

Methods Participants were 46 undergraduate students. None was trained in formal logic or in philosophy. Participants read the stories of three characters, and answered six questions after reading each story. The three characters were described as constantly feeling very tired (an uncommon feeling for them) after two recent changes in their lives: working at night and having a stressful boss (for the first character), working at night and becoming a dad (for the second character), and having a stressful boss and becoming a dad (for the third character). The first three questions assessed participants' background knowledge with respect to (i) the relation between the first event and feeling constantly tired; (ii) the second event and feeling constantly tired; and (iii) the conjunction of the two events and feeling constantly tired. For example:

> What do you think is the most common, the most normal: Working at night and feeling constantly tired, or working at night and not feeling constantly tired? or are those equally common and normal?

Participants who chose the first, second, and third answer were assumed to endorse *WorkNight* $\vdash$ *Tired*; *WorkNight* $\vdash$ $\neg$*Tired*; and (*WorkNight* $\nvdash$ *Tired*) $\wedge$ (*WorkNight* $\nvdash$ $\neg$*Tired*), respectively. The fourth, fifth, and sixth questions assessed participants' ascriptions of causality or facilitation between (i) the first event and feeling constantly tired; (ii) the second event and feeling constantly tired; and (iii) the conjunction of the two events and feeling constantly tired. E.g., one of these questions read:

> Fill in the blank with the word 'caused' or 'facilitated', as seems the most appropriate. If neither seems appropriate, fill in the blank with 'xxx': *Working at night ... the fact that Julien feels constantly tired*.

The experiment was conducted in French,[3] and the order in which the stories were presented to the participants was counterbalanced.

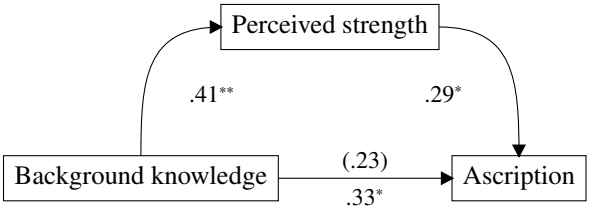
Results Out of the 116 ascriptions that the model predicted to be of facilitation, 68% indeed were, 11% were of causality, and 21% were neither. Out of the 224 ascriptions that the model predicted to be of causality, 46% indeed were, 52% were of facilitation, and 2% were neither. The global trend in the results is thus that background knowledge that theoretically matches a facilitation ascription indeed largely leads people to make such an ascription, while background knowledge that theoretically matches a causality ascription leads people to divide equally between causality and facilitation ascriptions. This trend is statistically reliable for almost all ascriptions required by the task. Relevant statistics (χ^2 scores) are higher than 7.7 for 7 out of the 9 ascriptions ($p < .05$, one-tailed, in all cases), and higher than 3.2 for the remaining two ascriptions ($p < .10$, one-tailed, in both cases). From these results, it appears that the notion of facilitation does have intuitive appeal to lay reasoners, and that it is broadly used as defined in our model. In particular, it clearly has a role to play in situations where an ascription of causality sounds too strong a conclusion, but no ascription at all sounds too weak.

Experiment 2

Experiment 2 was designed to consolidate the results of Experiment 1 and to answer the following questions: Does the fact that background knowledge match Def. 1 or Def. 2 affect the strength of the link participants perceive between two reported events, and does this perceived strength in turn determine whether they make an ascription of causality or facilitation?

[3]The term 'a favorisé' was used for 'facilitated', instead of the apparently straightforward translation 'a facilité', for it seemed pragmatically awkward to use the French verb 'faciliter' for an undesirable outcome.

Figure 1: Mediating role of perceived strength for the effect of background knowledge on ascription. Coefficients are standardized βs, $^{*}p < .05$, $^{**}p < .01$.



Methods Participants were 41 undergraduates. Elements of their background knowledge were assessed as in Exp. 1, in order to select triples of propositions $<$ *Context*, *Factor*, *Effect* $>$ that matched either Def. 1 or Def. 2. E.g., a participant might believe that one has generally no accident when driving, but that one will generally have an accident when driving after some serious drinking; for this participant, $<$ *Drive*, *SeriousDrinking*, *Accident* $>$ is a match with Def. 2. Participants then rated on a 9-point scale how strongly *Factor* and *Effect* were related. Finally, as a measure of ascription, they chose an appropriate term to describe the relation between *Factor* and *Effect*, from a list including 'causes' and 'facilitates'.

Results Out of the 16 ascriptions that the model predicted to be of facilitation, 14 were so, and 2 were of causality. Out of the 25 ascriptions that the model predicted to be of causality, 11 were so, and 14 were of facilitation. Beliefs thus had the expected influence on ascriptions, $\chi^2 = 4.5$, $p < .05$. The trend observed in Experiment 1 is replicated in Experiment 2. We also conducted a *mediation analysis* of our data, which consists in a series of 3 regression analyzes (see Figure 1). The direct effect of background knowledge on ascription was significant, $\beta = .33$, $p < .05$. The effect of background knowledge on perceived strength was also significant, $\beta = .41$, $p < .01$. In the third regression, background knowledge and perceived strength were entered simultaneously. Perceived strength was a reliable predictor of ascription, $\beta = .29$, $p < .05$, which was no longer the case for background knowledge, $\beta = .23$, $p > .05$. Data thus meet the requirement of a mediational effect: Whether the background knowledge of participants matches Def. 1 or Def. 2 determines their final ascription of $C : Factor \Rightarrow_{\text{fa}} Effect$ or $C : Factor \Rightarrow_{\text{ca}} Effect$ through its effect on the perceived strength of the link between *Factor* and *Effect*.

PROPERTIES OF CAUSAL ASCRIPTIONS

Impossibility of mutual causality

Proposition 1. *If* $C : A \Rightarrow_{\text{ca}} B$*, then it cannot hold that* $C : B \Rightarrow_{\text{ca}} A$.

Proof. If $C : A \Rightarrow_{\text{ca}} B$, it holds that $C \vdash \neg A$, $C \wedge A \vdash B$, and the sequence $\neg B_t, A_t, B_{t+1}$ has been observed. This is not inconsistent with $C \vdash \neg A$, $C \wedge B \vdash A$ (the background knowledge part of $C : B \Rightarrow_{\text{ca}} A$), but it is inconsistent with

the sequence $\neg A_t, B_t, A_{t+1}$ that would allow the ascription $C : B \Rightarrow_{\text{ca}} A$. □

Preference for abnormal causes

Psychologists established that abnormal conditions are more likely to be selected by human agents as the cause of an event (Hilton & Slugoski 1986) and more so if this event is itself abnormal (Gavansky & Wells 1989) (see also (Hart & Honoré 1985) in the area of legal philosophy). Our model reflects this preference: Only what is abnormal in a given context can be perceived as facilitating or causing a change in the normal course of things in this context.

Proposition 2. *If $C : A \Rightarrow_{\text{ca}} B$ or $C : A \Rightarrow_{\text{fa}} B$, then $C \nvdash \neg A$.*

Proof. $C \nvdash \neg A$ is false when either $C \vdash A$ or $C \nvdash \neg A$. If $C \vdash A$, it cannot be true that both $C \vdash \neg B$ and either $A \wedge C \nvdash \neg B$ (the definition of $C : A \Rightarrow_{\text{fa}} B$) or $A \wedge C \vdash B$ (the definition of $C : A \Rightarrow_{\text{ca}} B$). This is due to the Cautious Monotony property of $\vdash$, which forces $C \wedge A \vdash \neg B$ from $C \vdash A$ and $C \vdash \neg B$. Likewise, the Rational Monotony of $\vdash$ forces $C \wedge A \vdash \neg B$ from $C \nvdash \neg A$ and $C \vdash \neg B$; thus, it cannot be the case that $C : A \Rightarrow_{\text{fa}} B$ or $C : A \Rightarrow_{\text{ca}} B$ when $C \nvdash \neg A$. □

Example 2 (The unreasonable driver). *Let us imagine an agent who believes it is normal to be drunk in the context of driving (Drive $\vdash$ Drunk). This agent may think that it is exceptional to have an accident when driving (Drive $\vdash \neg$Accident). In that case, the agent cannot but believe that accidents are exceptional as well when driving while drunk: Drive $\wedge$ Drunk $\vdash \neg$Accident. As a consequence, when learning that someone got drunk, drove his car, and had an accident, this agent will neither consider that C : Drunk $\Rightarrow_{\text{fa}}$ Accident nor that C : Drunk $\Rightarrow_{\text{ca}}$ Accident.*

Transitivity

Def. 2 does not grant general transitivity to $\Rightarrow_{\text{ca}}$. If $C : A \Rightarrow_{\text{ca}} B$ and $C : B \Rightarrow_{\text{ca}} D$, it does not always follow that $C : A \Rightarrow_{\text{ca}} D$. Formally: $C \vdash \neg B$ and $A \wedge C \vdash B$ and $C \vdash \neg D$ and $B \wedge C \vdash D$ do not entail $C \vdash \neg D$ and $A \wedge C \vdash D$, because $\vdash$ itself is not transitive. Although $\Rightarrow_{\text{ca}}$ is not generally transitive, it becomes so in one particular case.

Proposition 3. *If $C : A \Rightarrow_{\text{ca}} B$, $C : B \Rightarrow_{\text{ca}} D$, and $B \wedge C \vdash A$, then $C : A \Rightarrow_{\text{ca}} D$.*

Proof. From the definition of $C : B \Rightarrow_{\text{ca}} D$, it holds that $B \wedge C \vdash D$. From $B \wedge C \vdash A$ and $B \wedge C \vdash D$, applying Cautious Monotony yields $A \wedge B \wedge C \vdash D$, which together with $A \wedge C \vdash B$ (from the definition of $C : A \Rightarrow_{\text{ca}} B$) yields by Cut $A \wedge C \vdash D$; since it holds from the definition of $C : B \Rightarrow_{\text{ca}} D$ that $C \vdash \neg D$, the two parts of the definition of $C : A \Rightarrow_{\text{ca}} D$ are satisfied. □

Example 3 (Mud on the plates). *Driving back from the countryside, you get a fine because your plates are muddy, Drive : Mud $\Rightarrow_{\text{ca}}$ Fine. Let us assume that you perceive your driving to the countryside as the cause for the plates to be muddy, Drive : Countryside $\Rightarrow_{\text{ca}}$ Mud. For transitivity to apply, i.e., to judge that Drive : Countryside $\Rightarrow_{\text{ca}}$ Fine, it must hold that Mud $\wedge$ Drive $\vdash$ Countryside: If mud on your plates usually means that you went to the countryside, then the trip can be considered the cause of the fine. If the presence of mud on your plates does not allow to infer that you went to the countryside (perhaps you also regularly drive through muddy streets where you live), then transitivity is not applicable; you will only consider that the mud caused the fine, not that the trip did.*

Entailment and causality ascriptions

Classical entailment $\models$ does not preserve $\Rightarrow_{\text{ca}}$. If $C : A \Rightarrow_{\text{ca}} B$ and $B \models B'$, one cannot say that $C : A \Rightarrow_{\text{ca}} B'$. Indeed, while $A \wedge C \vdash B'$ follows by right weakening (Kraus, Lehmann, & Magidor 1990) from $A \wedge C \vdash B$, it is not generally true that $C \vdash \neg B'$, given that $C \vdash \neg B$. Besides, according to Definition 2, if $A' \models A$, the fact that $C : A \Rightarrow_{\text{ca}} B$ does not entail that $C : A' \Rightarrow_{\text{ca}} B$, since $C \vdash \neg B$ and $A \wedge C \vdash B$ do not entail $A' \wedge C \vdash B$ when $A' \models A$. This fact is due to the extreme cautiousness of System P. It is contrasted in the following example with Rational Monotony.

Example 4 (Stone throwing). *An agent believes that a window shattered because a stone was thrown at it (Window : Stone $\Rightarrow_{\text{ca}}$ Shatter), based on its beliefs that Window $\vdash \neg$Shatter and Stone $\wedge$ Window $\vdash$ Shatter. Using the Cautious Monotony of System P, it is not possible to predict that the agent would make a similar ascription if a* small *stone had been thrown (SmallStone), or if a* white *stone had been thrown (WhiteStone), or even if a* big *stone had been thrown (BigStone), although it holds that SmallStone $\models$ Stone, WhiteStone $\models$ Stone, and BigStone $\models$ Stone. Adding Rational Monotony (Lehmann & Magidor 1992) to System P allows the ascriptions Window : BigStone $\Rightarrow_{\text{ca}}$ Shatter and Window : WhiteStone $\Rightarrow_{\text{ca}}$ Shatter, but also Window : SmallStone $\Rightarrow_{\text{ca}}$ Shatter. To block this last ascription, it would be necessary that the agent has specific knowledge about the harmlessness of small stones, such as Window $\wedge$ Smallstone $\nvdash$ Shatter or even Window $\wedge$ Smallstone $\vdash \neg$Shatter.*

Stability w.r.t. disjunction and conjunction

$\Rightarrow_{\text{ca}}$ is stable with respect to disjunction, both on the right and on the left, and stable w.r.t. conjunction on the right.

Proposition 4. *The following properties hold:*

1. *If $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, then $C : A \Rightarrow_{\text{ca}} B \vee B'$.*
2. *If $C : A \Rightarrow_{\text{ca}} B$ and $C : A' \Rightarrow_{\text{ca}} B$, then $C : A \vee A' \Rightarrow_{\text{ca}} B$.*
3. *If $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, then $C : A \Rightarrow_{\text{ca}} B \wedge B'$.*

Proof. Applying AND to the first part of the definitions of $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, i.e., $C \vdash \neg B$ and $C \vdash \neg B'$, yields $C \vdash \neg B \wedge \neg B'$, and thus $C \vdash \neg(B \vee B')$. Now, applying AND to the second part of the definitions of $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, i.e., $A \wedge C \vdash B$ and $A \wedge C \vdash B'$, yields $A \wedge C \vdash B \wedge B'$, which together with Right Weakening yields $A \wedge C \vdash B \vee B'$. The definition of $C : A \Rightarrow_{\text{ca}} B \vee B'$ is thus satisfied. The proof of Fact 2 is obtained by applying OR to the second part of the definitions of $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$. Finally, applying AND to the first part

of the definitions of $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, i.e., $C \vdash \neg B$ and $C \vdash \neg B'$, yields $C \vdash \neg B \wedge \neg B'$, which together with Right Weakening, yields $C \vdash \neg B \vee \neg B'$, and thus $C \vdash \neg(B \wedge B')$. Now, applying AND to the second part of the definitions of $C : A \Rightarrow_{\text{ca}} B$ and $C : A \Rightarrow_{\text{ca}} B'$, i.e., $A \wedge C \vdash B$ and $A \wedge C \vdash B'$, yields $A \wedge C \vdash B \wedge B'$. The definition of $C : A \Rightarrow_{\text{ca}} B \wedge B'$ is thus satisfied. □

$\Rightarrow_{\text{ca}}$ is not stable w.r.t. conjunction on the left. If $C : A \Rightarrow_{\text{ca}} B$ and $C : A' \Rightarrow_{\text{ca}} B$, then it is not always the case that $C : A \wedge A' \Rightarrow_{\text{ca}} B$ (see example 5). This lack of stability is once again due to the cautiousness of System P; for $C : A \wedge A' \Rightarrow_{\text{ca}} B$ to hold, it is necessary that $C \wedge A \vdash A'$ or, alternatively, that $C \wedge A' \vdash A$. Then Cautious Monotony will yield $A \wedge A' \wedge C \vdash B$. Rational Monotony can soften this constraint and make it enough that $C \wedge A \nvdash \neg A'$ or $C \wedge A' \nvdash \neg A$.

Example 5 (Busy professors). *Suppose that professors in your department seldom show up early at the office (Prof $\vdash$ ¬Early). However, they generally do so when they have tons of student papers to mark (Prof $\wedge$ Mark $\vdash$ Early), and also when they have a grant proposal to write (Prof $\wedge$ Grant $\vdash$ Early). When learning that a professor had tons of papers to grade and that she came in early, you would judge that Prof : Mark $\Rightarrow_{\text{ca}}$ Early. Likewise, when learning that a professor had a grant proposal to write and came in early, you would judge that Prof : Grant $\Rightarrow_{\text{ca}}$ Early. But what if you learn that a professor had tons of papers to grade* and *a grant proposal to write and that she came in early? That would depend on whether it is an exceptional situation to have to deal with both tasks on the same day. If it is not exceptional (Mark $\nvdash$ ¬Grant), then you will judge that Prof : Mark $\wedge$ Grant $\Rightarrow_{\text{ca}}$ Early. If, on the contrary, Mark $\wedge$ Grant is an exceptional event, it does not hold anymore that Mark $\wedge$ Grant $\vdash$ Early, and it is thus impossible to feel sure about Prof : Mark $\wedge$ Grant $\Rightarrow_{\text{ca}}$ Early. For example, it might be the case that faced with such an exceptional workload, a professor will prefer working at home all day rather than coming to the office. In that case, her coming in early would be due to another factor, e.g., a meeting that could not be cancelled.*

ASCRIPTIONS OF JUSTIFICATION

Perceived causality as expressed in Def. 2 should be distinguished from the situation that we term 'justification.' We write that $C : A \Rightarrow_{\text{ju}} B$ when an agent judges that the occurrence of A in context C gave reason to expect the occurrence of B.

Definition 3 (Justification). *An agent that learns in context C of the sequence $\neg B_t$, A_t, B_{t+1} will judge that $C : A \Rightarrow_{\text{ju}} B$ if it believes that $C \nvdash \neg B$, $C \nvdash B$ and $A \wedge C \vdash B$.*

Faced with facts C, $\neg B_t$, A_t, B_{t+1}, an agent believing that $C \nvdash \neg B$, $C \nvdash B$ and $A \wedge C \vdash B$ may doubt that the change from $\neg B_t$ to B_{t+1} is really due to A_t, although the latter is indeed the very reason for the lack of surprise at having B_{t+1} reported. Indeed, situation $\neg B_t$ at time t appears to the agent to be contingent, since it is neither a normal nor an abnormal course of things in context C. This clearly departs from the situation where $C \vdash \neg B$ and $A \wedge C \vdash B$, wherein the agent will judge that $C : A \Rightarrow_{\text{ca}} B$. In a nutshell, the case whereby $C \nvdash \neg B$, $C \nvdash B$ and $A \wedge C \vdash B$ cannot be interpreted as the recognition of a causal phenomenon by an agent: All that can be said is that reporting A caused the agent to start believing B, and that she should not be surprised of having B_{t+1} reported.

What we call justification is akin to the notion of explanation following Spohn (Spohn 1983): Namely, 'A is a reason for B' when raising the epistemic rank for A raises the epistemic rank for B. Gärdenfors (Gärdenfors 1990) captured this view to some extent, assuming that A is a reason for B if B is not retained in the contraction of A. Williams et al. (Williams *et al.* 1995) could account for the Spohnian view in a more refined way using kappa-rankings and transmutations, distinguishing between weak and strong explanations. As our framework can easily be given a possibilistic semantics (Benferhat, Dubois, & Prade 1997), it could properly account for this line of thought, although our distinction between perceived causation and epistemic justification is not the topic of the above works.

RELATED WORKS

Causality plays a central role in at least two problems studied in AI, diagnosis and the simulation of dynamical systems. Diagnosis problems are a matter of abduction: One takes advantage of the knowledge of some causal links to infer the most plausible causes of an observed event (Peng & Reggia 1990). In this setting, causality relations are often modelled by conditional probabilities P(effect|cause).[4] Dynamical systems are modelled in AI with respect, e.g., to qualitative physics (de Kleer & Brown 1986), and in logics of action. The relation of nonmonotonic inference to causality has already been emphasized by authors dealing with reasoning about actions and the frame problem (Giunchiglia *et al.* 2004; McCain & Turner 1995; Turner 1999). Material implication being inappropriate to represent a causal link, these approaches define a 'causal rule' as 'there is a cause for effect B to be true if it is true that A has just been executed', where 'there is a cause for' is modelled by a modal operator.

The problem discussed in this paper is not, however, one of classical diagnosis. Neither does it deal with the qualitative simulation of dynamical systems, nor with the problem of describing changes caused by the execution of actions, nor with what does not change when actions are performed. We are concerned here with a different question, namely the explanation of a sequence of reported events, in terms of pairs of events that can be considered as related by a causality relation. In that sense, our work is reminiscent of the 'causal logic' of Shafer (Shafer 1998), which provides a logical setting that aims at describing the possible relations of concomitance between events when an action takes place.

[4] Nevertheless, Bayesian networks (Pearl 1988) (that represent a joint probability distribution by means of a directed graph) do not necessarily reflect causal links between their nodes, for different graphical representations can be obtained depending on the ordering in which variables are considered (Dubois & Prade 1999).

However, Shafer's logic does not leave room for abnormality. This notion is central in our approach, as it directly relates to the relations of qualitative independence explored in (Dubois *et al.* 1999) – causality and independence being somewhat antagonistic notions.

Following (Pearl 2000), Halpern and Pearl (Halpern & Pearl to appeara; to appearb) have proposed a model that distinguishes real causes ('cause in fact') from potential causes, by using an a priori distinction between 'endogenous' variables (the possible values of which are governed by structural equations, for example physical laws), and 'exogenous' variables (determined by external factors). Exogenous variables cannot be deemed causal. Halpern and Pearl's definition of causality formalizes the notion of an active causal process. More precisely, the fact A that a subset of endogenous variables has taken some definite values is the real cause of an event B if (i) A and B are true in the real world, (ii) this subset is minimal, (iii) another value assignment to this subset would make B false, the values of the other endogenous variables that do not directly participate to the occurrence of B being fixed in some manner, and (iv) A alone is enough for B to occur in this context. This approach, thanks to the richness of background knowledge when it is represented in structural equations, makes it possible to treat especially difficult examples.

Building upon the notion of potential cause, Chockler and Halpern (Chockler & Halpern 2003) have introduced definitions of responsibility and blame: The extent to which a cause (or an agent) is responsible for an effect is graded, and depends on the presence of other potential causes (or agents). Clearly, the assessment of responsibility from identification of causal relationships raises further problems that will not be discussed here.

Our model is not to be construed as an alternative or a competitor to models based on structural equations. Indeed, we see our approach as either a 'plan B' or a complement to structural equation modeling. One might not have access to the accurate information needed to build a structural equation model; in this case, our less demanding model might still be operable. Alternatively, a decision support system may be able to build a structural equation model of the situation, although its users only have access to qualitative knowledge. In that case, the system will be able to compare its own causality ascriptions to the conclusions of the qualitative model, and take appropriate explanatory steps, would those ascriptions be too different. Indeed, our model does not aim at identifying the true, objective cause of an event, but rather at predicting what causal ascription an agent would make based on the limited information it has at its disposal.

Models based on structural equations are often supplemented with the useful notion of *intervention*. In many situations, finding the cause of an event will be much easier if the agent can directly intervene in the manner of an experimenter. In future work, we intend to explore the possibility of supplementing our own model with a similar notion by means of a $\mathbf{do}(\bullet)$ operator. An ascription of causality (resp., facilitation) would be made iff the requirements of Definition 2 (resp., 1) are met both for A, B, C *and* for $\mathbf{do}(A)$, B, C, where $\mathbf{do}(A)$ means that the occurrence of A is forced by an intervention (Pearl 2000). As for now, we only give a brief example of how such an operator can be used in our approach.

Example 6 (Yellow teeth). *An agent learns that someone took up smoking, that this person's teeth yellowed, and that this person developed lung cancer. The agent believes that generally speaking, it is abnormal to be a smoker, to have yellow teeth, and to develop lung cancer (resp., ($C \vdash \neg Smoke$, $C \vdash \neg Yellow$, $C \vdash \neg Lung$). The agent believes that it is normal for smokers to have yellow teeth ($C \wedge Smoke \vdash Yellow$) and to develop lung cancer ($C \wedge Smoke \vdash Lung$), and that it is not abnormal for someone who has yellow teeth to develop lung cancer ($C \wedge Yellow \nvdash \neg Lung$). From these beliefs and observations, Definitions 1 and 2 would allow for various ascriptions, including the following one: Smoking caused the yellow teeth which* in turn *facilitated lung cancer. With the additional constraint based on the* $\mathbf{do}(\bullet)$ *operator, only one set of ascriptions remains possible:* Both *the yellow teeth and the lung cancer were caused by smoking. Yellow teeth cannot be said anymore to facilitate lung cancer because, inasmuch as lung cancer is generally abnormal, it holds that* $C \wedge \mathbf{do}(Yellow) \vdash \neg\, Lung$*: There is no reason to think that one will develop lung cancer after painting one's teeth yellow.*

CONCLUDING REMARKS

We have presented a simple qualitative model of the causal ascriptions an agent will make from its background default knowledge, when confronted with a series of events. In addition to supplementing this model with a $\mathbf{do}(\bullet)$ operator, we intend to extend our present work in three main directions. First, we should be able to equip our framework with possibilistic qualitative counterparts to Bayesian networks (Benferhat *et al.* 2002), since System P augmented with Rational Monotony can be represented in possibilistic logic (Benferhat, Dubois, & Prade 1997). Second, we will derive postulates for causality from the independence postulates presented in (Dubois *et al.* 1999). Finally, in parallel to further theoretical elaboration, we will maintain a systematic experimental program that will test the psychological plausibility of our definitions, properties, and postulates.

ACKNOWLEDGMENTS

This work was supported by a grant from the Agence Nationale pour la Recherche, project number NT05-3-44479.

References

Benferhat, S.; Dubois, D.; Garcia, L.; and Prade, H. 2002. On the transformation between possibilistic logic bases and possibilistic causal networks. *International Journal of Approximate Reasoning* 29:135–173.

Benferhat, S.; Bonnefon, J. F.; and Da Silva Neves, R. M. 2004. An experimental analysis of possibilistic default reasoning. In *KR2004*, 130–140. AAAI Press.

Benferhat, S.; Bonnefon, J. F.; and Da Silva Neves, R. M. 2005. An overview of possibilistic handling of default reasoning: An experimental study. *Synthese* 146:53–70.

Benferhat, S.; Dubois, D.; and Prade, H. 1997. Nonmonotonic reasoning, conditional objects and possibility theory. *Artificial Intelligence* 92:259–276.

Bonnefon, J. F.; Da Silva Neves, R. M.; Dubois, D.; and Prade, H. 2006. Background default knowledge and causality ascriptions. In *ECAI2006*. IOS Press.

Chockler, H., and Halpern, J. 2003. Responsibility and blame. A structural-model approach. In *IJCAI'03*. San Francisco, CA: Morgan Kaufmann.

Da Silva Neves, R. M.; Bonnefon, J. F.; and Raufaste, E. 2002. An empirical test for patterns of nonmonotonic inference. *Annals of Mathematics and Artificial Intelligence* 34:107–130.

de Kleer, J., and Brown, J. S. 1986. Theories of causal ordering. *Artificial Intelligence* 29:33–61.

Dubois, D., and Prade, H. 1999. Probability theory in artificial intelligence. Book review of J. Pearl's 'Probabilistic Reasoning in Intelligent Systems'. *Journal of Mathematical Psychology* 34:472–482.

Dubois, D., and Prade, H. 2005. Modeling the role of (ab)normality in the ascription of causality judgements by agents. In *NRAC'05*, 22–27.

Dubois, D.; Fariñas Del Cerro, L.; Herzig, A.; and Prade, H. 1999. *A roadmap of qualitative independence*, volume 15 of *Applied Logic series*. Dordrecht, The Netherlands: Kluwer. 325–350.

Ford, M. 2004. System LS: A three tiered nonmonotonic reasoning system. *Computational Intelligence* 20:89–108.

Gärdenfors, P. 1990. The dynamics of belief systems: Foundations vs. coherence theories. *Revue Internationale de Philosophie* 44:24–46.

Gavansky, I., and Wells, G. L. 1989. Counterfactual processing of normal and exceptional events. *Journal of Experimental Social Psychology* 25:314–325.

Giunchiglia, E.; Lee, J.; McCain, N.; Lifschitz, V.; and Turner, H. 2004. Non-monotonic causal theories. *Artificial Intelligence* 153:49–104.

Halpern, J., and Pearl, J. to appeara. Causes and explanations: A structural-model approach — part 1: Causes. *British Journal for the Philosophy of Science*.

Halpern, J., and Pearl, J. to appearb. Causes and explanations: A structural-model approach — part 2: Explanations. *British Journal for the Philosophy of Science*.

Hart, H. L. A., and Honoré, T. 1985. *Causation in the law*. Oxford: Oxford University Press.

Hilton, D. J., and Slugoski, B. R. 1986. Knowledge-based causal attribution: The abnormal conditions focus model. *Psychological Review* 93:75–88.

Kraus, S.; Lehmann, D.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44:167–207.

Lehmann, D., and Magidor, M. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55:1–60.

McCain, N., and Turner, H. 1995. A causal theory of ramifications and qualifications. In *IJCAI'95*. San Francisco, CA: Morgan Kaufmann.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge: Cambridge University Press.

Peng, Y., and Reggia, J. A. 1990. *Abductive Inference Models for Diagnostic Problem-Solving*. Berlin: Springer Verlag.

Pfeifer, N., and Kleiter, G. D. 2005. Coherence and nonmonotonicity in human reasoning. *Synthese* 146:93–109.

Shafer, G. 1998. Causal logic. In *ECAI'98*, 711–719. Chichester, England: Wiley.

Shoham, Y. 1990. Nonmonotonic reasoning and causation. *Cognitive Science* 14:213–252.

Spohn, W. 1983. Deterministic and probabilistic reasons and causes. *Erkenntnis* 19:371–393.

Turner, H. 1999. A logic of universal causation. *Artificial Intelligence* 113:87–123.

von Wright, G. H. 1963. *Norm and Action: A Logical Enquiry*. London: Routledge.

Williams, M.-A.; Pagnucco, M.; Foo, N.; and Sims, B. 1995. Determining explanations using transmutations. In *IJCAI'95*, 822–830. Morgan Kaufmann.

2.10 Decidability of a Conditional-probability Logic with Non-standard Valued Probabilities

Decidability of a Conditional-probability Logic with Non-standard Valued Probabilities

Miodrag Rašković, Zoran Marković, Zoran Ognjanović
Matematički Institut
Kneza Mihaila 35, 11000 Beograd, Srbija i Crna Gora
zorano@mi.sanu.ac.yu zoranm@mi.sanu.ac.yu

Abstract

A probabilistic logic was defined in (Rašković, Ognjanović & Marković 2004; 2004) with probabilistic operators, both conditional and "absolute", which are applied to propositional formulas. The range of probabilistic operators is syntactically restricted to a recursive subset of a non standard interval ${}^*[0,1]$ which means that it contains infinitesimals. In this paper we prove the decidability of that logic. This means that the logic may be a suitable candidate for AI applications such as, e.g. modelling the default reasoning.

Introduction

The problem of reasoning with statements whose truth is only probable is an old problem which was rejuvenated in the 80's by the interest from AI (Nilsson 1986). One line of research consisted in studying a propositional calculus enriched with probabilistic operators (Fagin & Halpern 1994; Fagin, Halpern, & Megiddo 1990). Rather extensive bibliography of probability logics can be found in (Database of probability logic papers 2005). Usual semantics for such systems consists of a Kripke model of possible worlds with appropriate probability measure on sets of worlds. On the other hand, the study of default logics had a high point with (Kraus, Lehmann, & Magidor 1990), where a system P was introduced which is now considered to be the common core of default reasoning. In the same paper, a semantics for P was introduced which consisted of nonstandard (${}^*R$) probabilistic models.

In (Rašković, Ognjanović & Marković 2004; 2004) a logic (denoted LPP^S) was introduced in which five types of probabilistic operators are applied to propositional formulas: $P_{\geq s}$, $P_{\approx r}$, $CP_{=s}$, $CP_{\geq s}$, and $CP_{\approx r}$, where r is a rational number from [0,1] and $s \in S$, where S is a unit interval of the Hardy field $Q[\epsilon]$. The intended meaning of the operators is: "the probability is at least s", "approximatively r", and "the conditional probability is s", "at least s", and "approximatively r", respectively. The semantics consists of Kripke models extended with a finitely additive probability measure defined on an algebra of sets of possible worlds. The range of this probability measure is syntactically restricted to S. Namely, there is a rule which allows $A \rightarrow \bot$ to be derived from the set of formulas $\{A \rightarrow \neg P_{=s} : s \in S\}$. This logic makes it possible to discuss "conditioning on the events of zero probability". Namely, if the probability of $\alpha \wedge \beta$ is ϵ and probability of β is 2ϵ, the conditional probability of α given β will be $\frac{1}{2}$. In the absence of infinitesimals and following the approach based on Kolmogorov's ideas, the probabilities of $\alpha \wedge \beta$ and β would be 0, so the conditional probability would be 1. Another application of this logic is to default reasoning. It turns out that the formulas of the type $CP_{\approx 1}(\alpha, \beta)$ faithfully represent defaults in the sense that all rules of P can be derived in LPP^S for the formulas of this type. Furthermore, the same derivations can be made from a finite set of defaults. In the case of an infinite set of defaults, however, more conclusions can be derived in LPP^S, as demonstrated on an example from (Lehmann & Magidor 1992).

It was proved in (Rašković, Ognjanović & Marković 2004) that LPP^S is sound and complete for the set of $LPP^S_{Meas,Neat}$-models, but it remained unclear whether LPP^S is decidable. In this paper we prove that LPP^S is decidable, which makes it suitable for realistic applications.

The rest of the paper is organized as follows. At the beginning we briefly describe the logic LPP^S, giving its syntax, semantics and an axiomatic system (for more details see (Rašković, Ognjanović & Marković 2004)). The next section contains the proof of decidability of LPP^S. Finally, we conclude with some remarks on possibility to use our system to model default reasoning.

The logic LPP^S

Let S be the unit interval of the Hardy field $Q[\epsilon]$. $Q[\epsilon]$ is a recursive nonarchimedean field which contains all rational functions of a fixed positive infinitesimal ϵ which belongs to a nonstandard elementary extension ${}^*R$ of the standard real numbers (Keisler 1986; Robinson 1966). An element ϵ of ${}^*R$ is an infinitesimal if $|\epsilon| < \frac{1}{n}$ for every natural number n. $Q[\epsilon]$ contains all rational numbers. Let $Q[0,1]$ denote the set of rational numbers from $[0,1]$.

The language of the logic consists of:

- a denumerable set $\mathrm{Var} = \{p, q, r, \ldots\}$ of propositional letters,
- the classical connectives $\neg$, and $\wedge$,
- unary probabilistic operators $(P_{\geq s})_{s \in S}$, $(P_{\approx r})_{r \in Q[0,1]}$, and

- binary probabilistic operators $(CP_{\geq s})_{s\in S}$, $(CP_{=s})_{s\in S}$, $(CP_{\approx r})_{r\in Q[0,1]}$.

The set For_C of classical propositional formulas is defined as usual. Elements of For_C will be denoted by α, β, ... The set For_P^S of probabilistic propositional formulas is the smallest set Y containing all formulas of the forms:

- $P_{\geq s}\alpha$ for $\alpha \in For_C$, $s \in S$,
- $P_{\approx r}\alpha$ for $\alpha \in For_C$, $r \in Q[0,1]$,
- $CP_{=s}(\alpha,\beta)$ for $\alpha,\beta \in For_C$, $s \in S$,
- $CP_{\geq s}(\alpha,\beta)$ for $\alpha,\beta \in For_C$, $s \in S$ and
- $CP_{\approx r}(\alpha,\beta)$ for $\alpha,\beta \in For_C$, $r \in Q[0,1]$,

and closed under the formation rules:

- if A belongs to Y, then $\neg A$ is in Y.
- if A and B belong to Y, then $(A \wedge B)$ is in Y.

Formulas from For_P^S will be denoted by A, B, ... Neither mixing of pure propositional formulas and probability formulas, nor nested probabilistic operators are allowed. The other classical connectives ($\vee$, $\rightarrow$, $\leftrightarrow$) can be defined as usual, while we denote $\neg P_{\geq s}\alpha$ by $P_{<s}\alpha$, $P_{\geq 1-s}\neg\alpha$ by $P_{\leq s}\alpha$, $\neg P_{\leq s}\alpha$ by $P_{>s}\alpha$, $P_{\geq s}\alpha \wedge \neg P_{>s}\alpha$ by $P_{=s}\alpha$, $\neg P_{=s}\alpha$ by $P_{\neq s}\alpha$, $\neg CP_{\geq s}(\alpha,\beta)$ by $CP_{<s}(\alpha,\beta)$, $CP_{<s}(\alpha,\beta) \vee CP_{=s}(\alpha,\beta)$ by $CP_{\leq s}(\alpha,\beta)$, and $CP_{\geq s}(\alpha,\beta) \wedge \neg CP_{=s}(\alpha,\beta)$ by $CP_{>s}(\alpha,\beta)$. Let $For^S = For_C \cup For_P^S$. φ, ψ, ... will be used to denote formulas from the set For^S. For $\alpha \in For_C$, and $A \in For_P^S$, we abbreviate both $\neg(\alpha \rightarrow \alpha)$ and $\neg(A \rightarrow A)$ by $\bot$ letting the context determine the meaning, while $\top$ denotes $\neg\bot$.

The semantics for For^S will be based on Kripke models.

Definition 1 *An LPP^S-model is a structure $\langle W, H, \mu, v\rangle$ where:*

- *W is a nonempty set of elements called worlds,*
- *H is an algebra of subsets of W,*
- *$\mu : H \rightarrow S$ is a finitely additive probability measure, and*
- *$v : W \times \mathrm{Var} \rightarrow \{true, false\}$ is a valuation which associates with every world $w \in W$ a truth assignment $v(w)$ on the propositional letters.*

The valuation v is extended to a truth assignment on all classical propositional formulas. Let M be an LPP^S model and $\alpha \in For_C$. The set $\{w : v(w)(\alpha) = true\}$ is denoted by $[\alpha]_M$.

Definition 2 *An LPP^S-model M is measurable if $[\alpha]_M \in H$ for every formula $\alpha \in For_C$. An LPP^S-model M is neat if only the empty set has the zero probability. $LPP^S_{Meas,Neat}$ denotes the class of all neat and measurable LPP^S-models.*

The neatness-condition is introduced in order to make our models a subclass of ${}^*R$-probabilistic models of (Kraus, Lehmann, & Magidor 1990; Lehmann & Magidor 1992). This facilitates the explanation of a possible application of our system to default reasoning (see the last section). All the results can be also proved for the class of measurable LPP^S-models.

Definition 3 *The satisfiability relation $\models \subset LPP^S_{Meas,Neat} \times For^S$ is defined by the following conditions for every $LPP^S_{Meas,Neat}$-model M:*

1. *if $\alpha \in For_C$, $M \models \alpha$ if $(\forall w \in W)v(w)(\alpha) = true$,*
2. *$M \models P_{\geq s}\alpha$ if $\mu([\alpha]_M) \geq s$,*
3. *$M \models P_{\approx r}\alpha$ if for every positive integer n, $\mu([\alpha]_M) \in [\max\{0, r-1/n\}, \min\{1, r+1/n\}]$,*
4. *$M \models CP_{\geq s}(\alpha,\beta)$ if either $\mu([\beta]_M) = 0$ or $\mu([\beta]_M) > 0$ and $\frac{\mu([\alpha\wedge\beta]_M)}{\mu([\beta]_M)} \geq s$,*
5. *$M \models CP_{=s}(\alpha,\beta)$ if either $\mu([\beta]_M) = 0$ and $s = 1$ or $\mu([\beta]_M) > 0$ and $\frac{\mu([\alpha\wedge\beta]_M)}{\mu([\beta]_M)} = s$,*
6. *$M \models CP_{\approx r}(\alpha,\beta)$ if either $\mu([\beta]_M) = 0$ and $r = 1$ or $\mu([\beta]_M) > 0$ and for every positive integer n, $\frac{\mu([\alpha\wedge\beta]_M)}{\mu([\beta]_M)} \in [\max\{0, r-1/n\}, \min\{1, r+1/n\}]$.*
7. *if $A \in For_P^S$, $M \models \neg A$ if $M \not\models A$,*
8. *if $A, B \in For_P^S$, $M \models A \wedge B$ if $M \models A$ and $M \models B$.*

Note that the conditions 3 and 6 are equivalent to saying that the (conditional) probability equals $r - \epsilon_i$ (or $r + \epsilon_i$) for some infinitesimal $\epsilon_i \in S$.

A formula $\varphi \in For^S$ is satisfiable if there is an $LPP^S_{Meas,Neat}$-model M such that $M \models \varphi$; φ is valid if for every $LPP^S_{Meas,Neat}$-model M, $M \models \varphi$; a set of formulas is satisfiable if there is a model in which every formula from the set is satisfiable.

The main theorem proved in (Rašković, Ognjanović & Marković 2004) concerns completeness of the logic with respect to the class $LPP^S_{Meas,Neat}$:

Theorem 1 (Extended completeness theorem) *A set T of formulas has an $LPP^S_{Meas,Neat}$-model if and only if it is consistent with respect to the following axiom system:*
Axiom schemes:

1. *all For_C-instances of classical propositional tautologies*
2. *all For_P^S-instances of classical propositional tautologies*
3. $P_{\geq 0}\alpha$
4. $P_{\leq s}\alpha \rightarrow P_{<t}\alpha$, $t > s$
5. $P_{<s}\alpha \rightarrow P_{\leq s}\alpha$
6. $P_{\geq 1}(\alpha \leftrightarrow \beta) \rightarrow (P_{=s}\alpha \rightarrow P_{=s}\beta)$
7. $(P_{=s}\alpha \wedge P_{=t}\beta \wedge P_{\geq 1}\neg(\alpha \wedge \beta)) \rightarrow P_{=\min(1,s+t)}(\alpha \vee \beta)$
8. $P_{\approx r}\alpha \rightarrow P_{\geq r_1}\alpha$, *for every rational* $r_1 \in [0, r)$
9. $P_{\approx r}\alpha \rightarrow P_{\leq r_1}\alpha$, *for every rational* $r_1 \in (r, 1]$
10. $CP_{=s}(\alpha,\beta) \rightarrow \neg CP_{=t}(\alpha,\beta)$, $s \neq t$
11. $P_{=0}\beta \rightarrow CP_{=1}(\alpha,\beta)$
12. $(P_{=t}\beta \wedge P_{=s}(\alpha \wedge \beta)) \rightarrow CP_{=s/t}(\alpha,\beta)$, $t \neq 0$
13. $CP_{=s}(\alpha,\beta) \rightarrow \neg CP_{\geq t}(\alpha,\beta)$, $s < t$
14. $CP_{=s}(\alpha,\beta) \rightarrow CP_{\geq t}(\alpha,\beta)$, $s \geq t$
15. $CP_{=s}(\alpha,\beta) \rightarrow (P_{=ts}(\alpha \wedge \beta) \leftrightarrow P_{=t}\beta)$, $t \neq 0$
16. $CP_{\approx r}(\alpha,\beta) \rightarrow CP_{\geq r_1}(\alpha,\beta)$, *for every rational* $r_1 \in [0, r)$
17. $CP_{\approx r}(\alpha,\beta) \rightarrow CP_{\leq r_1}(\alpha,\beta)$, *for every rational* $r_1 \in (r, 1]$

18. $CP_{=r}(\alpha,\beta) \rightarrow CP_{\approx r}(\alpha,\beta)$

Inference rules:

1. *From* φ *and* $\varphi \rightarrow \psi$ *infer* ψ.
2. *If* $\alpha \in For_C$, *from* α *infer* $P_{\geq 1}\alpha$.
3. *From* $A \rightarrow P_{\neq s}\alpha$, *for every* $s \in S$, *infer* $A \rightarrow \bot$.
4. *From* $A \rightarrow (P_{=ts}(\alpha \wedge \beta) \leftrightarrow P_{=s}\beta)$, *for every* $s \in S \setminus \{0\}$, *infer* $A \rightarrow CP_{=t}(\alpha,\beta)$.
5. *For every* $r \in Q[0,1]$, *from* $A \rightarrow P_{\geq r-1/n}\alpha$, *for every integer* $n \geq 1/r$, *and* $A \rightarrow P_{\leq r+1/n}\alpha$ *for every integer* $n \geq 1/(1-r)$, *infer* $A \rightarrow P_{\approx r}\alpha$.
6. *For every* $r \in Q[0,1]$, *from* $A \rightarrow CP_{\geq r-1/n}(\alpha,\beta)$, *for every integer* $n \geq 1/r$, *and* $A \rightarrow CP_{\leq r+1/n}(\alpha,\beta)$ *for every integer* $n \geq 1/(1-r)$, *infer* $A \rightarrow CP_{\approx r}(\alpha,\beta)$.

Decidability

It is well known that there is a procedure to decide whether a classical propositional formula is satisfiable. Thus, to prove decidability of our logic, it is enough to show that the satisfiability problem for probability formulas is decidable.

We can perform some easy transformations which will reduce the satisfiability problem to checking probability formulas of simpler form. Let A be a probability formula and $p_1, \ldots, p_n$ be a list of all propositional letters from A. An atom a of A is a formula $\pm p_1 \wedge \ldots \pm p_n$, where $\pm p_i$ is either p_i, or $\neg p_i$. For different atoms a_i and a_j we have $\vdash a_i \rightarrow \neg a_j$. We use $At(A)$ to denote the set of all atoms from A, and n to denote the number of propositional letters from A. Obviously, $|At(A)| = 2^n$.

Using propositional reasoning and the fact that if $\vdash \alpha \leftrightarrow \beta$, then $\vdash P_{\geq s}\alpha \leftrightarrow P_{\geq s}\beta$, it is easy to show that every probability formula A is equivalent to a formula:

$$DNF(A) = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} \pm X_{i,j}(p_1, \ldots, p_n)$$

called a disjunctive normal form of A, where:

- $X_{i,j} \in \{P_{\geq s}, CP_{=s}, CP_{\geq s}\}_{s \in S} \cup \{P_{\approx r}, CP_{\approx r}\}_{r \in Q[0,1]}$,
- $\pm X_{i,j}$ is either $X_{i,j}$ or $\neg X_{i,j}$, and
- $X_{i,j}(p_1, \ldots, p_n)$ denotes that the propositional formula which is in the scope of the probability operator $X_{i,j}$ is in the complete disjunctive normal form, i.e. the propositional formula is a disjunction of the atoms of A.

Obviously, to prove decidability of our logic, it is enough to show that satisfiability of probability formulas of the form $\bigwedge_{i=1}^{k} \pm X_i(p_1, \ldots, p_n)$ is decidable.

Every formula of the form $P_{\geq s}\alpha$ is equivalent to the formula $CP_{\geq s}(\alpha, \top)$. Furthermore, for any conditional probability formula ($\pm CP_{\geq s}(\alpha,\beta)$, $\pm CP_{=s}(\alpha,\beta)$, and $\pm CP_{\approx r}(\alpha,\beta)$) we can distinguish two cases:

1. the probability of β is zero, in which case
   - $CP_{\geq s}(\alpha,\beta)$, for $s \in S$, $\neg CP_{=s}(\alpha,\beta)$, for $s \in S \setminus \{1\}$, $CP_{=1}(\alpha,\beta)$, and $CP_{\approx 1}(\alpha,\beta)$ hold - and can be deleted from the formula, while
   - $\neg CP_{\geq s}(\alpha,\beta)$, for $s \in S$, $CP_{=s}(\alpha,\beta)$, $CP_{\approx r}(\alpha,\beta)$, for $s \in S \setminus \{1\}$, $r \in Q[0,1] \setminus \{1\}$ $\neg CP_{=1}(\alpha,\beta)$, and $\neg CP_{\approx 1}(\alpha,\beta)$ do not hold - and the whole formula is not satisfiable,
2. the probability of β is greater than zero.

As a consequence, to prove decidability of our logic it is enough to prove decidability of satisfiability of formulas which are conjunctions of conditional probabilistic formulas of the forms: $\pm CP_{\geq s}(\alpha,\beta)$, $\pm CP_{=s}(\alpha,\beta)$, and $\pm CP_{\approx r}(\alpha,\beta)$, such that the probability of β is greater than 0.

In the next step, we will reduce the satisfiability problem to linear programming problem. Note that the same is done to prove decidability in papers (Fagin, Halpern, & Megiddo 1990; Rašković 1993; Ognjanović & Rašković 2000) that deal with the standard real-valued probabilities. However, in the logic we discuss here, the range of probabilities is recursive and contains non-standard values, and there are operators of the form $CP_{\approx r}$ that do not appear in the mentioned papers. Thus, we should perform the reduction carefully to obtain linear systems that are suitable for establishing decidability, which, in our approach, means that Fourier-Motzkin elimination can be applied to them. The idea is to eliminate $\approx$ and $\not\approx$ signs and to try to solve linear systems in an extension of $Q(\epsilon)$. In the rest of this section we will use the following abbreviations:

- x_i denotes the measure of the atom $a_i \in At(A)$, $i = 1, 2^n$,
- $a_i \models \alpha$ means that the atom a_i appears in the complete disjunctive normal form of a classical propositional formula α,
- $\sum(\alpha)$ denotes $\sum_{a_i \in At(A): a_i \models \alpha} x_i$, and
- $\mathrm{C}\sum(\alpha,\beta)$ denotes $\frac{\sum(\alpha \wedge \beta)}{\sum(\beta)}$.

Recall that $[\alpha]_M$ denotes the set of all worlds of an $LPP^S_{Meas,Neat}$-model M that satisfy α. Since $[\alpha]_M = \cup_{a_i \in At(A): a_i \models \alpha}[a_i]_M$, and different atoms are mutually exclusive, i.e., $[a_i]_M \cap [a_j]_M = \emptyset$ for $i \neq j$, $CP_{\geq s}(\alpha,\beta)$ holds in M iff $\sum(\beta) = 0$, or $\sum(\beta) > 0$ and $\mathrm{C}\sum(\alpha,\beta) \geq s$ (and similarly for $CP_{=s}$, and $CP_{\approx r}$).

Let us consider a formula A of the form:

$$(\wedge_{i=1,I} \pm CP_{\geq s_i}(\alpha_i,\beta_i)) \wedge (\wedge_{j=1,J} \pm CP_{=s_j}(\alpha_j,\beta_j)) \wedge (\wedge^{\pm}_{l=1,L} CP_{\approx r_l}(\alpha_l,\beta_l)).$$

Then, A is satisfiable iff the following system of linear equalities and inequalities is satisfiable:

$\sum_{i=1}^{2^n} x_i = 1$

$x_i \geq 0$ for $i = 1, 2^n$

$\sum(\beta) > 0$ for every formula β appearing in the formulas of the form $\pm CP_{\diamond}(\alpha, \beta)$ from A and $\diamond \in \{\geq s_i, = s_j, \approx r_l\}$

$\mathrm{C}\sum(\alpha_i, \beta_i) \geq s_i$ for every formula $CP_{\geq s_i}(\alpha_i, \beta_i)$ from A

$\mathrm{C}\sum(\alpha_i, \beta_i) < s_i$ for every formula $\neg CP_{\geq s_i}(\alpha_i, \beta_i)$ from A

$\mathrm{C}\sum(\alpha_j, \beta_j) = s_j$ for every formula $CP_{=s_j}(\alpha_j, \beta_j)$ from A

$\mathrm{C}\sum(\alpha_j, \beta_j) > s_j$ or $\mathrm{C}\sum(\alpha_j, \beta_j) < s_j$ for every formula $\neg CP_{=s_j}(\alpha_j, \beta_j)$ from A

$\mathrm{C}\sum(\alpha_l, \beta_l) \approx r_l$ for every formula $CP_{\approx r_l}(\alpha_l, \beta_l)$ from A

$\mathrm{C}\sum(\alpha_l, \beta_l) \not\approx s_l$ for every formula $\neg CP_{\approx r_l}(\alpha_l, \beta_l)$ from A.

We can further simplify the above system by observing that every expression of the form $\mathrm{C}\sum(\alpha_l, \beta_l) \approx r_l$ can be seen as

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \approx 0 \text{ and } \mathrm{C}\sum(\alpha_l, \beta_l) - r_l \geq 0 \quad (1)$$

or

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \approx 0 \text{ and } r_l - \mathrm{C}\sum(\alpha_l, \beta_l) \geq 0. \quad (2)$$

Similarly, every expression of the form $\mathrm{C}\sum(\alpha_l, \beta_l) \not\approx r_l$ can be seen as

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \not\approx 0 \text{ and } \mathrm{C}\sum(\alpha_l, \beta_l) - r_l > 0 \quad (3)$$

or

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \not\approx 0 \text{ and } r_l - \mathrm{C}\sum(\alpha_l, \beta_l) > 0. \quad (4)$$

Thus, we will consider systems containing expressions of the forms (1 - 4) instead of $\mathrm{C}\sum(\alpha_l, \beta_l) \approx r_l$, and $\mathrm{C}\sum(\alpha_l, \beta_l) \not\approx r_l$, respectively.

Let us use $S(\overrightarrow{x}, \epsilon)$ to denote a system of that form. Note that

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \approx 0 \quad \text{and} \quad \mathrm{C}\sum(\alpha_l, \beta_l) - r_l \geq 0 \quad (5)$$

is equivalent to

$$(\exists n_l \in N)\ 0 \leq \mathrm{C}\sum(\alpha_l, \beta_l) - r_l < n_l \cdot \epsilon,$$

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \approx 0 \quad \text{and} \quad r_l - \mathrm{C}\sum(\alpha_l, \beta_l) \geq 0 \quad (6)$$

is equivalent to

$$(\exists n_l \in N)\ 0 \geq \mathrm{C}\sum(\alpha_l, \beta_l) - r_l > -n_l \cdot \epsilon,$$

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \not\approx 0 \quad \text{and} \quad \mathrm{C}\sum(\alpha_l, \beta_l) - r_l > 0 \quad (7)$$

is equivalent to

$$(\exists n_l \in N) \quad \mathrm{C}\sum(\alpha_l, \beta_l) - r_l > \frac{1}{n_l},$$

$$\mathrm{C}\sum(\alpha_l, \beta_l) - r_l \not\approx 0 \quad \text{and} \quad r_l - \mathrm{C}\sum(\alpha_l, \beta_l) > 0 \quad (8)$$

is equivalent to

$$(\exists n_l \in N) \quad \mathrm{C}\sum(\alpha_l, \beta_l) - r_l < -\frac{1}{n_l}.$$

Since we have only finitely many expressions of the forms (1 – 4) in our system, we can use a unique $n_0 \in N$ instead of many n_l's in expressions (5 – 8). We use $S(\overrightarrow{x}, n_0, \epsilon)$ to denote the obtained system. Then,

$$S(\overrightarrow{x}, \epsilon) \text{ has a solution in } Q(\epsilon) \text{ iff} \quad (9)$$
$$S(\overrightarrow{x}, n_0, \epsilon) \text{ has a solution in } Q(\epsilon),$$

and, since $Q(\epsilon)$ is dense in ${}^*R$, for every fixed and finite n_0,

$$S(\overrightarrow{x}, n_0, \epsilon) \text{ has a solution in } Q(\epsilon) \text{ iff} \quad (10)$$
$$S(\overrightarrow{x}, n_0, \epsilon) \text{ has a solution in } {}^*R.$$

Note that n_0 is not determined in the (9) and (10) above. Now, we will replace n_0 with another, infinite but fixed, parameter H which will also have some suitable characteristics in relation to ϵ. The role of H is to help us to avoid the standard approach to the analysis of inequalities, where we have very often to discuss arguments of the form "it holds for all, large enough integers". Since H is a positive infinite integer, if an inequality holds for every n greater than some fixed finite n_0, by the overspill principle it also holds for H. The other direction is a consequence of the underspill principle which says that if an inequality holds for every infinite number less than H, it also holds for some finite positive integer. Thus, let us consider the following set

$$O = \{n \in {}^*N : S(\overrightarrow{x}, n, \epsilon) \text{ has a solution in } {}^*R\}.$$

O is an internal set which contains all natural numbers greater than some fixed natural number n'. Using the overspill and underspill principles, we conclude that, if $S(\overrightarrow{x}, \epsilon)$ is solvable in $Q(\epsilon)$, then O also contains all infinite numbers from ${}^*N$ which are less than a fixed infinite natural number H. In other words, for some $n' \in N$, and $H \in {}^*N \setminus N$, $[n', H] = \{n \in {}^*N : n' \leq n \leq H\} \subset O$. Then,

$$S(\overrightarrow{x}, n_0, \epsilon) \text{ has a solution in } {}^*R \text{ iff} \quad (11)$$
$$S(\overrightarrow{x}, H, \epsilon) \text{ has a solution in } {}^*R.$$

We can choose H so that for every $k \in N$, $H^k \cdot \epsilon \approx 0$. That can be explained as follows. Let us consider the internal set $O' = \{n \in {}^*N : n^n < \frac{1}{\sqrt{\epsilon}}\}$. Obviously, $N \subset O'$. Using the overspill principle, there is some $H \in {}^*N \setminus N$ such that

$0 < H^H < \frac{1}{\sqrt{\epsilon}}$, and $0 < H^H \cdot \epsilon < \sqrt{\epsilon}$. Thus, for every $k \in N$, $0 < H^k \cdot \epsilon < \sqrt{\epsilon}$, and $H^k \cdot \epsilon \approx 0$.

Note that $\approx$ and $\not\approx$ do not appear in the system $S(\overrightarrow{x}, H, \epsilon)$. Thus, we can freely multiply (in)equalities by the denominators of the expressions of the form $\mathrm{C}\sum(\alpha, \beta)$ and in that way obtain linear (in)equalities of the form

$$\sum(\alpha \wedge \beta) - s \sum(\beta)\ \rho\ 0,$$

where s is a polynomial in ϵ and H, and $\rho \in \{\geq, >, =, <, \leq\}$.

Now, we can perform Fourier-Motzkin elimination, which iteratively rewrites the starting system into a new system without a variable x_i such that two systems are equisatisfiable. During the procedure, numerators and denominators of coefficients in (in)equalities remain polynomials in ϵ and H. When no variables are left, we have to check satisfiability of relations between numerical expressions with parameters ϵ and H which can be done since: $Q(\epsilon)$ is recursive and ordered field, and H is chosen so that for every $k \in N$, $H^k \cdot \epsilon \approx 0$. Namely, we consider two polynomials $Q_1(\epsilon, H)$ and $Q_2(\epsilon, H)$ in ϵ and H of the forms: $Q_1(\epsilon, H) = q_{1,0}Q_{1,0}(H)\epsilon^0 + q_{1,1}Q_{1,1}(H)\epsilon^1 + \ldots + q_{1,n_1}Q_{1,n_1}(H)\epsilon^{n_1}$, and $Q_2(\epsilon, H) = q_{2,0}Q_{2,0}(H)\epsilon^0 + q_{2,1}Q_{2,1}(H)\epsilon^1 + \ldots + q_{2,n_2}Q_{2,n_2}(H)\epsilon^{n_2}$, where $q_{i,j}$'s are rationals, and $Q_{i,j}(H)$'s are polynomials in H with rational coefficients. Comparison of polynomials $Q_1(\epsilon, H)$ and $Q_2(\epsilon, H)$ starts by examining $q_{1,0}Q_{1,0}(H)$ and $q_{2,0}Q_{2,0}(H)$ in the standard way. If they are equal, we have to examine $q_{1,1}Q_{1,1}(H)$ and $q_{2,1}Q_{2,1}(H)$ and so on. Since ϵ is an infinitesimal, the above examination of expressions sharing the same powers of ϵ is done in a reverse order with respect to the standard procedure of comparison of polynomials.

It follows that the problem of solving whether $S(\overrightarrow{x}, H, \epsilon)$ has a solution in ${}^*R$ is decidable. From the equalities (9), (10), and (11), it follows that the problem of solving whether $S(\overrightarrow{x}, \epsilon)$ has a solution in $Q(\epsilon)$ is decidable, too.

If $S(\overrightarrow{x}, \epsilon)$ is solvable, we can define an LPP^S-model $M = \langle W, H, \mu, v\rangle$ such that $W = At(A)$, $H = 2^W$, μ is defined according to the solutions of $S(\overrightarrow{x}, \epsilon)$, and v satisfies $v(a)(p) = \top$ iff p (and not $\neg p$) occurs in the conjunction which constitutes the atom a. Obviously, $M \models A$. However, even if $S(\overrightarrow{x}, \epsilon)$ has a solution, some of x_i's might be 0. It means that M does not satisfy the neatness condition, i.e., that some non-empty sets of worlds (represented by the corresponding atoms that hold in those worlds) have the zero probabilities. In that case, we can simply remove those worlds and denote the obtained model by M'. It is easy to see that for every $A \in For_P^S$, $M \models A$ iff $M' \models A$. Thus, we have:

Theorem 2 *The problem of $LPP^S_{Meas,Neat}$-satisfiability is decidable.*

As an example, let us consider the following problem. Let b, f, and l denote 'bird', 'flies', and 'living creature', respectively. Suppose that our knowledge base is $KB = \{CP_{\approx 1}(f, b), CP_{\approx .3}(b, l)\}$ which means that birds generally fly, and that approximately 30% of living creatures are birds. Then, we can ask for the probability that a randomly chosen living creature flies, i.e., for the conditional probability $CP(f \mid l)$. Using the above procedure we can check for what $k(k \leq 1)$, $CP_{=k}(f, l)$ is consistent with KB.

Let

$x_1 = \mu(b \wedge f \wedge l)$,
$x_2 = \mu(b \wedge f \wedge \neg l)$,
$x_3 = \mu(b \wedge \neg f \wedge l)$,
$x_4 = \mu(b \wedge \neg f \wedge \neg l)$,
$x_5 = \mu(\neg b \wedge f \wedge l)$,
$x_6 = \mu(\neg b \wedge f \wedge \neg l)$,
$x_7 = \mu(\neg b \wedge \neg f \wedge l)$ and
$x_8 = \mu(\neg b \wedge \neg f \wedge \neg l)$.

Supposing that we consider living creatures only, we have that $x_2 = 0$, $x_4 = 0$, $x_6 = 0$, and $x_8 = 0$. Then, from $CP_{\approx 1}(f, b)$, $CP_{\approx .3}(b, l)$ and $CP_{=k}(f, l)$, we obtain the following system:

$x_1 + x_3 + x_5 + x_7 = 1$
$x_2 + x_4 + x_6 + x_8 = 0$
$x_i \geq 0, i = 1, 2, \ldots, 8$
$\frac{x_1 + x_2}{x_1 + x_2 + x_3 + x_4} \approx 1$
$\frac{x_1 + x_3}{x_1 + x_3 + x_5 + x_7} \approx 0.3$
$\frac{x_1 + x_5}{x_1 + x_3 + x_5 + x_7} = k$

Since $x_1 + x_3 + x_5 + x_7 = 1$, we have also that:

$\frac{x_1}{x_1 + x_3} \approx 1$
$x_1 + x_3 \approx 0.3$, and
$x_1 + x_5 = k$,

Now, we can eliminate $\approx$:

$x_1 + x_3 + x_5 + x_7 = 1$
$x_i \geq 0, i = 1, 3, 5, 7$
$1 - \frac{x_1}{x_1 + x_3} \leq n_1\epsilon$
$0 < x_1 + x_3 - 0.3 < n_2\epsilon$ or $0 < 0.3 - (x_1 + x_3) < n_2\epsilon$
$x_1 + x_5 = k$

Let $n_0 = \max\{n_1, n_2\}$. Let H be an infinite natural number fixed as above. Then, an easy calculation shows that the last system is solvable iff the following condition is fulfilled:

$$0.3 - 1.3H\epsilon - H^2\epsilon^2 < k \leq 1.$$

Conclusion

In this paper we have proved decidability of the $LPP^S_{Meas,Neat}$-satisfiability problem. One of the questions that are still open is to find a precise characterization of the corresponding computational complexity. It is important having in mind possible applications of LPP^S to problems involving uncertain probabilistic knowledge and default reasoning. Namely, in (Kraus, Lehmann, & Magidor 1990; Lehmann & Magidor 1992) a set of properties which form a core of default reasoning, the corresponding formal system P, and a family of nonstandard (${}^*R$) probabilistic models characterizing the default consequence relation defined

by the system P, are proposed. Probabilities from ${}^*R$-probabilistic models are ${}^*R$-valued, while in our approach the range of probabilities is a countable subset S of the unit interval of ${}^*R$. In (Rašković, Ognjanović & Marković 2004) we describe in details how our system can be used to model default reasoning. The main results are ($CP_{\approx 1}(\beta, \alpha)$ corresponds to the default 'if α, then generally β', denoted by $\alpha \rightarrowtail \beta$):

- If we consider the language of defaults and finite default bases, the entailment coincides with the one in the system P.
- If we consider the language of defaults and arbitrary default bases, more conclusions can be obtained in our system than in the system P. For example, in our system we can go beyond the system P, when we consider the infinite default base $\Delta = \{p_i \rightarrowtail p_{i+1}, p_{i+1} \rightarrowtail \neg p_i\}$, $i = 0, 1, \ldots$ Namely, p_0 is P-consistent (Lehmann & Magidor 1992), while we obtain $\Delta \vdash_{Ax_{LPPS}} CP_{\approx 1}(\bot, p_0)$.
- When we consider our full language, we can express probabilities of formulas, negations of defaults, combinations of defaults with the other (probabilistic) formulas etc. For example, the translation of rational monotonicity, $((\alpha \rightarrowtail \beta) \wedge \neg(\alpha \rightarrowtail \neg\gamma)) \rightarrow ((\alpha \wedge \gamma) \rightarrowtail \beta)$, which is an important default-reasoning principle is $LPP^S_{Meas,Neat}$-valid, while it cannot be even formulated in the framework of the pure language of defaults.
- Our system is not sensitive to the syntactical form which represents the available knowledge (duplications of rules in the knowledge base).

Although the ideas of using probabilities and infinitesimals in default reasoning are not new ((Adams 1975; Benferhat, Saffiotti, & Smets 2000; Goldszmidt & Pearl 1996; Lehmann & Magidor 1992; Satoh 1990)), the above facts show that our approach does not coincide with any of those systems.

Finally, note that in this paper the probabilistic operators may be applied to classical propositional formulas only. It is enough to reason about probabilities of events described by (classical propositional) formulas, but we cannot speak about higher order probabilities (probabilities of probabilities, probabilities of defaults, defaults defined on uncertain knowledge). A logic which allows that was presented in (Ognjanović, Marković & Rašković 2005).

References

E. W. Adams. *The logic of Conditional*. Dordrecht: Reidel. 1975.

N. Alechina. Logic with probabilistic operators. In *Proc. of the ACCOLADE '94*, 121 – 138. 1995.

F. Bacchus, A.J. Grove, J.Y. Halpern, and D. Koller. From Statistical Knowledge Bases to Degrees of Belief. Artificial Intelligence (87): 75–143. 1996.

S. Benferhat, A. Saffiotti, and P. Smets. Belief functions and default reasoning. *Artificial Intelligence* (122):1 – 69. 2000.

V. Biazzo, A. Gilio, T. Lukasiewicz, and G. Sanfilippo. Probabilistic logic under coherence, model-theoretic probabilistic logic, and default reasoning in System P. *Journal of Applied Non-Classical Logics* 12(2): 189–213. 2002.

G. Coletti, and R. Scozzafava. *Probabilistic logic in a coherent setting*. Kluwer Academic Press, Dordrecht, The Netherlands. 2002.

Database of probability logic papers. http://problog.mi.sanu.ac.yu, *Mathematical Institute*, Belgrade. 2005.

R. Đorđević, M. Rašković, and Z. Ognjanović. Completeness theorem for propositional probabilistic models whose measures have only finite ranges. *Archive for Mathematical Logic* 43, 557 – 563. 2004.

R. Fagin, and J. Halpern. Reasoning about knowledge and probability. *Journal of the ACM* 41(2):340 – 367. 1994.

R. Fagin, J. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation* 87(1-2):78 – 128. 1990.

A. Gilio. Probabilistic reasoning under coherence in System P. *Annals of Mathematics and Artificial Intelligence* 34, 5 – 34. 2002.

M. Goldszmidt, and J. Pearl. Qualitative probabilities for default reasoning, belief revision and causal modeling. *Artificial Intelligence* 84(1-2):57 – 112. 1996.

J. Keisler. *Elementary calculus. An infinitesimal approach. 2nd ed.* Boston, Massachusetts: Prindle, Weber & Schmidt. 1986.

S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44:167 – 207. 1990.

D. Lehmann, and M. Magidor. What does a conditional knowledge base entail? *Artificial Intelligence* 55:1 – 60. 1992.

T. Lukasiewicz. Probabilistic Default Reasoning with Conditional Constraints. *Annals of Mathematics and Artificial Intelligence* 34, 35 – 88. 2002.

T. Lukasiewicz. Weak nonmonotonic probabilistic logics. *Artificial Intelligence* 168(1-2): 119–161. 2005.

Z. Marković, Z. Ognjanović, and M. Rašković. A probabilistic extension of intuitionistic logic. *Mathematical Logic Quarterly* 49:415 – 424. 2003.

N. Nilsson. Probabilistic logic *Artificial Intelligence*, 28, 71 – 87. 1986.

Z. Ognjanović, and M. Rašković. Some probability logics with new types of probability operators. *Journal of Logic and Computation* 9(2):181 – 195. 1999.

Z. Ognjanović, and M. Rašković. Some first-order probability logics. *Theoretical Computer Science* 247(1-2):191 – 212. 2000.

Z. Ognjanović, Z. Marković, and M. Rašković. Completeness Theorem for a Logic with Imprecise and Conditional Probabilities. *Publications de l'Institut Mathématique*, Nouvelle Série, Beograd, vol. 78(92): 35 – 49. 2005.

M. Rašković. Classical logic with some probability operators. *Publications de l'Institut Mathématique, Nouvelle Série, Beograd* (53(67)):1 – 3. 1993.

M. Rašković, Z. Ognjanović, and Z. Marković. A Probabilistic Approach to Default Reasoning. In *Proc. of the NMR '04*, 335 – 341. 2004.

M. Rašković, Z. Ognjanović, and Z. Marković. A Logic with Conditional Probabilities. In *Proc. of the JELIA'04*, Lecture notes in artificial intelligence (LNCS/LNAI), 3229, 226 – 238, Springer-Verlag. 2004.

Robinson, A. *Non-standard analysis*. Amsterdam: North-Holland. 1966.

Satoh, K. A probabilistic interpretation for lazy nonmonotonic reasoning. In *Proc. of the Eighth American Conference on Artificial Intelligence*, 659 – 664. 1990.

2.11 About the computation of forgetting symbols and literals

About the computation of forgetting symbols and literals

Yves Moinard
INRIA/IRISA, Campus de Beaulieu, 35042 RENNES-Cedex FRANCE
moinard@irisa.fr

Abstract

Recently, the old logical notion of forgetting propositional symbols (or reducing the logical vocabulary) has been generalized to a new notion: forgetting literals. The aim was to help the automatic computation of various formalisms which are currently used in knowledge representation, particularly for nonmonotonic reasoning. We develop here a further generalization, allowing propositional symbols to vary while forgetting literals. We describe the new notion, on the syntactical and the semantical side. We provide various manipulations over the basic definitions involved, including for the original version, which hopefully should help improving again the efficiency of the computation. This work concerns especially circumscription, since it is known that one way of computing circumscription uses the forgetting of literals.

Introduction

The well-known notion of forgetting propositional symbols, which is known at least since a 1854 paper by Boole under the name "elimination of middle terms", has been used for a long time in mathematical logic and in its applications for knowledge representation (see e.g. (Lin & Reiter 1994; Lin 2001; Su, Lv, & Zhang 2004)). It is a reduction of the vocabulary, thanks to the suppression of some propositional symbols. Let us consider the formula

$$(bird \wedge \neg exceptional \rightarrow flies) \wedge \neg exceptional.$$

We may want to "forget" the symbol $exceptional$, considered here as "auxiliary", then we get the formula

$$bird \rightarrow flies.$$

Recently, Lang et al. (Lang, Liberatore, & Marquis 2003) have extended this notion in a signifi cant manner, by allowing the forgetting of *literals*. In above example, it happens that in fact what has been done is equivalent to forgetting the literal $\neg exceptional$. In the general case, forgetting a literal is more precise than forgetting its propositional symbol: we get a formula standing "somewhere between" the original formula and the formula obtained by forgetting the propositional symbol.

This new defi nition is a natural extension of the classical defi nition of forgetting propositional symbols. Lang et al. have shown that this new notion also is useful for knowledge representation and particularly for nonmonotonic reasoning. In some cases, this provides a simplifi cation of the computations, and the authors provide various ways for computing the forgetting of literals, in order to obtain concrete examples of simplifi cation of the computation of some already known formalism.

We extend the notion by allowing some propositional symbols to *vary* when forgetting literals. The new defi nitions are a simple and natural extension of the original ones, and they have the same kind of behavior

We describe various ways for computing these notions (including the original ones, without varying symbols), and we provide hints showing that the complexity of the new notion should be comparable to the complexity of the notion without variable symbols. This is of some importance in order to apply the results given in (Lang, Liberatore, & Marquis 2003) to the new notion, since this should simplify signifi cantly the overall computation. The main application example of the interest of these methods for computing already known formalisms given in (Lang, Liberatore, & Marquis 2003) concerns circumscription, and (Moinard 2005) has shown how the new notion with varying symbols allows to reduce a two stage method to a single stage one.

Firstly, we give the preliminary notations and defi nitions. Then we remind the notion of propositional symbol forgetting, with a few more technical tools. Then we remind the notion of literal forgetting as introduced by Lang et al. Then we introduce our generalization, allowing symbols to vary when literals are forgotten. Finally, we detail yet another method for computing these notions.

Technical preliminaries

We work in a propositional language $\mathbf{PL}$. As usual, $\mathbf{PL}$ also denotes the set of all the formulas, and the *vocabulary of* $\mathbf{PL}$ is a set of *propositional symbols* denoted by $\mathcal{V}(\mathbf{PL})$. We restrict our attention to fi nite sets $\mathcal{V}(\mathbf{PL})$ in this text.

Letters φ, ψ denote formulas in $\mathbf{PL}$, $\top$ and $\bot$ denote respectively the true and the false formulas. *Interpretations for* $\mathbf{PL}$, identifi ed with subsets of $\mathcal{V}(\mathbf{PL})$, are denoted by the letter ω. The notations $\omega \models \varphi$ and $\omega \models X$ for a set X

of formulas are defined classically. For a set E, $\mathcal{P}(E)$ denotes the set of the subsets of E. The set $\mathcal{P}(\mathcal{V}(\mathbf{PL}))$ of the interpretations for $\mathbf{PL}$ is denoted by **Mod**. A *model* of X is an interpretation ω such that $\omega \models X$, $\mathbf{Mod}(\varphi)$ and $\mathbf{Mod}(X)$ denote respectively the sets of the models of $\{\varphi\}$ and X.

A *literal* l is either a symbol p in $\mathcal{V}(\mathbf{PL})$ (*positive literal*) or its negation $\neg p$ (*negative literal*). If l is a literal, $\sim l$ denotes its *complementary literal*: $\sim \neg p = p$ and $\sim p = \neg p$. Similarly, we define $\sim \top = \bot$ and $\sim \bot = \top$.

A *clause* (respectively a *term*) is a disjunction (respectively a conjunction) of literals. Subsets of $\mathcal{V}(\mathbf{PL})$ are denoted by P, Q, V. P^+ (respectively P^-) denotes the set of the positive (respectively negative) literals built on P, and $P^\pm$ denotes the set $P^+ \cup P^-$ of all the literals built on P (P and P^+ can be assimilated). For any (finite) set X of formulas, $\bigwedge X$ (respectively $\bigvee X$) denotes the conjunction (respectively disjunction) of all the formulas in X. We get: $\bigwedge X \equiv X$, $\bigwedge \emptyset \equiv \top$ and $\bigvee \emptyset \equiv \bot$. $\mathcal{V}(X)$ denotes the set of the propositional symbols appearing in X.

A *disjunctive normal form* or DNF of φ is a disjunction of consistent terms which is equivalent to φ. A set L of literals in $V^\pm$ (and the term $\bigwedge L$) is *consistent and complete in* V if each propositional symbol of V appears once and only once in L; the clause $\bigvee L$ is then *non trivial and complete in* V. For any set L of literals, $\sim L$ denotes the set of the literals complementary to those in L (notice that $\sim P = P^-$).

We need the following notions and notations, many of them coming from (Lang, Liberatore, & Marquis 2003):

If φ is some formula and p is a propositional symbol in $\mathbf{PL}$, $\varphi_{p:\top}$ (respectively $\varphi_{p:\bot}$) is the formula obtained from φ by replacing each occurrence of p by $\top$ (respectively $\bot$). If $l = p$ is a positive literal, $\varphi_{l:i}$, denotes the formula $\varphi_{p:i}$[1]; if $l = \neg p$ is a negative literal, $\varphi_{l:i}$ denotes the formula $\varphi_{p:\sim i}$.

Notations 1 *1. If $v_1, \cdots, v_n$ are propositional symbols, $\varphi_{(v_1:\epsilon_1, \cdots, v_n:\epsilon_n)}$ with each $\epsilon_j \in \{\bot, \top\}$, denotes the formula $(\cdots((\varphi_{v_1:\epsilon_1})_{v_2:\epsilon_2})\cdots)_{v_n:\epsilon_n}$.*
If the v_j's in the list are all distinct, the order of the v_j's is without consequence for the final result. Thus, if V_1 and V_2 are disjoint subsets of V, we may define $\varphi_{[V_1:\top, V_2:\bot]}$ as $\varphi_{(v_1:\top, \cdots, v_n:\top, v_{n+1}:\bot, \cdots, v_{n+m}:\bot)}$, where $(v_1, \cdots, v_n)$ and $(v_{n+1}, \cdots, v_{n+m})$ are two orderings of all the elements of V_1 and V_2 respectively.

2. *If $L = (l_1, \cdots, l_n)$ is a list of literals, $\varphi_{(l_1:\epsilon_1 \cdots l_n:\epsilon_n)}$ denotes the formula $(\cdots((\varphi_{l_1:\epsilon_1})_{l_2:\epsilon_2})\cdots)_{l_n:\epsilon_n}$.*
3. *Let $\mathcal{V}(\mathbf{PL})^\pm$ be ordered in some arbitrary way. If $L_1, \cdots, L_n$ are disjoint sets of literals, $\varphi_{\langle L_1:\epsilon_1, \cdots, L_n:\epsilon_n\rangle}$ denotes the formula $\varphi_{(l_1:\gamma_1, \cdots, l_n:\gamma_n)}$ where $(l_1, \cdots, l_n)$ is the enumeration of the set $L_1 \cup \cdots \cup L_n$ which respects the order chosen for the set of all the literals, and where, for each l_j, γ_j is equal to ϵ_r where $r \in \{1, \cdots, n\}$ is such that $l_j \in L_r$.*

[1] Notice that in (Lang, Liberatore, & Marquis 2003), "$\varphi_{\,l:\bot}$" (respectively "$\varphi_{\,l:\top}$") is denoted by "$\varphi_{\,l\leftarrow 0}$" (respectively "$\varphi_{\,l\leftarrow 1}$").

Forgetting propositional symbols

Let us remind a possible definition for this well known and old notion [2].

Definition 2 *If $V \subseteq \mathcal{V}(\mathbf{PL})$ and $\varphi \in \mathbf{PL}$, $ForgetV(\varphi, V)$ denotes a formula, in the propositional language $\mathbf{PL}_{\overline{V}}$ built on the vocabulary $\overline{V} = \mathcal{V}(\mathbf{PL}) - V$, which is equivalent to φ in this restricted language: $ForgetV(\varphi, V) \equiv Th(\varphi) \cap \mathbf{PL}_{\overline{V}}$ where $Th(\varphi) = \{\varphi' \in \mathbf{PL} / \varphi \models \varphi'\}$.*

For any $\psi \in \mathbf{PL}_{\overline{V}}$, $\varphi \models \psi$ iff $ForgetV(\varphi, V) \models \psi$.

Here are two known ways to get $ForgetV(\varphi, V)$:

1. In a DNF form of φ, for each term suppress all the literals in $V^\pm$ ("empty terms" being equivalent to $\top$ as usual).
2. For any formula φ, and any list V of propositional symbols, we get
   (a) $ForgetV(\varphi, \{v\} \cup V) = ForgetV(\varphi, V)_{v:\top} \vee ForgetV(\varphi, V)_{v:\bot}$,
   (b) $ForgetV(\varphi, \emptyset) = \varphi$.

The iterative point 2 applies to any formula, and shows that we can forget one symbol at a time. Also, the order is irrelevant: the final formulas are all equivalent when the order is modified. Here is the corresponding "global formulation" (cf Notations 1-1):

Definition 3 $ForgetV(\varphi, V) = \bigvee_{V' \subseteq V} \varphi_{[V':\top, (V-V'):\bot]}$.

Considering the formulation $ForgetV(\varphi, V) \equiv Th(\varphi) \cap \mathbf{PL}_{\overline{V}}$, the following obvious technical remark happens to be very useful:

Remark 4 *When considering a formula equivalent to a set $Th(\varphi) \cap X$, the set of formulas X can be replaced by any set Y having the same $\wedge$-closure: $\{\bigwedge X' / X' \subseteq X\} = \{\bigwedge X' / X' \subseteq Y\}$. Indeed, we have:*

- *If X and Y have the same $\wedge$-closure, then $Th(\varphi) \cap X \equiv Th(\varphi) \cap Y$.*
- *The converse is true,* provided that we assimilate equivalent formulas*: if $Th(\varphi) \cap X \equiv Th(\varphi) \cap Y$ for any $\varphi \in \mathbf{PL}$, then X and Y have the same $\wedge$-closure.*

Since we work in finite propositional languages, there exists a unique smallest (for set inclusion, and up to logical equivalence) possible set, the $\wedge$-reduct of X, equal to the set $X - \{\varphi \in X / \varphi$ is in the $\wedge$-closure of $X - \{\varphi\}\}$. Thus, X can be replaced by any set containing the $\wedge$-reduct of X and included in the $\wedge$-closure of X.

Thus, instead of considering the whole set $\mathbf{PL}_{\overline{V}}$ in $ForgetV(\varphi, V) \equiv Th(\varphi) \cap \mathbf{PL}_{\overline{V}}$ (Definition 2), we can consider the set of all the clauses built on $\overline{V}$, the smallest (for $\subseteq$) set that can be considered here being the set of these clauses which are non trivial and complete in $\overline{V}$.

[2] "V" in $ForgetV$ stands for "[propositional] variable", meaning "propositional symbol", and is in accordance with the notations of (Lang, Liberatore, & Marquis 2003), even if using term "variable" here could provoke confusions with the notions described later in this text.

On the semantical side, the set of the models of $ForgetV(\varphi, V)$ is the set of all the interpretations for $\mathbf{PL}$ which coincide with a model of φ for all the propositional symbols not in V: $\mathbf{Mod}(ForgetV(\varphi, V)) = \{\omega \in \mathbf{Mod} \ / \ \exists \omega', \ \omega' \models \varphi \ \text{ and } \omega \cap \overline{V} = \omega' \cap \overline{V}\}$.

These syntactical and semantical characterizations justify the name "$Forget$".

Example 1 *Here* $\mathcal{V}(\mathbf{PL}) = \{a, b, c, d\}$, *and* $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d)$.

DNF rule: $ForgetV(\varphi, \{b, c\}) \equiv (\neg a) \vee (a \wedge \neg d) \equiv \neg a \vee \neg d$.

iteratively: $ForgetV(\varphi, \{c\}) \equiv (\neg a \wedge b) \vee (a \wedge \neg b \wedge \neg d)$. $ForgetV(ForgetV(\varphi, \{c\}), \{b\}) \equiv ForgetV(\varphi, \{b, c\})$.

semantics:

Starting with $\mathbf{Mod}(\varphi) = \{\{a\}, \{b, c\}, \{b, c, d\}\}$, *we get the twelve models of* $ForgetV(\varphi, \{b, c\})$ *by adding all the interpretations varying on* $\{b, c\}$, *which gives the twelve interpretations:* $\emptyset \cup E, \{a\} \cup E, \{d\} \cup E$, *for any subset* E *of the set of the forgotten symbols* $\{b, c\}$.

Remind that for any formulas φ_1 and φ_2,
we get $ForgetV(\varphi_1 \vee \varphi_2, V) \equiv ForgetV(\varphi_1, V) \vee ForgetV(\varphi_2, V)$,
and $ForgetV(\varphi_1 \wedge \varphi_2, V) \models ForgetV(\varphi_1, V) \wedge ForgetV(\varphi_2, V)$.

Here is counter-example for the converse entailment:
$\varphi_1 = a \vee \neg b, \varphi_2 = b$, thus $\varphi_1 \wedge \varphi_2 = a \wedge b$ and
we get $ForgetV(\varphi_1, \{b\}) = ForgetV(\varphi_2, \{b\}) = \top$,
while $ForgetV(\varphi_1 \wedge \varphi_2, \{b\}) = a$.

We need now another definition:

Definition 5 *(Lang, Liberatore, & Marquis 2003, pp. 396–397) Let* ω *be an interpretation for* $\mathbf{PL}$, p *be a propositional symbol in* $\mathbf{PL}$ *and* L *be a consistent set of literals in* $\mathbf{PL}$.

We define the interpretations
$Force(\omega, p) = \omega \cup \{p\}$ *and*
$Force(\omega, \neg p) = \omega - \{p\}$ *and more generally,*
$Force(\omega, L) = \omega \cup \{p / p \in \mathcal{V}(\mathbf{PL}), p \in L\} - \{p / p \in \mathcal{V}(\mathbf{PL}), \neg p \in L\}$.

Thus, $Force(\omega, L)$ is the interpretation for $\mathbf{PL}$ equal to ω for all the propositional symbols in $\mathcal{V}(\mathbf{PL}) - \mathcal{V}(L)$ and which satisfies all the literals of L.

An immediate consequence of the definition of $\varphi_{l:\top}$ is that we get: $\mathbf{Mod}(\varphi_{l:\top}) =$
$\{\omega / \omega \models \varphi, \omega \models l\} \cup \{Force(\omega, \sim l) / \omega \models \varphi, \omega \models l\} =$
$\{Force(\omega, l), Force(\omega, \sim l) / \omega \models \varphi, \omega \models l\}$ ($\text{Mod}_{l:\top}$).

It is then interesting to relate $ForgetV(\varphi, v)$ [$v \in \mathcal{V}(\mathbf{PL})$] to the formulas $\varphi_{v:\top}$ and $\varphi_{v:\bot}$:

$$\begin{array}{rcl} \varphi_{v:\top} & \equiv & ForgetV(v \wedge \varphi, v); \\ \varphi_{v:\bot} & \equiv & ForgetV(\neg v \wedge \varphi, v). \\ ForgetV(\varphi, v) & \equiv & \varphi_{v:\top} \vee \varphi_{v:\bot}. \end{array}$$

Indeed, $ForgetV(\varphi, v) \equiv \varphi_{v:\top} \vee \varphi_{v:\bot}$ and $\varphi \equiv (v \wedge \varphi_{v:\top}) \vee (\neg v \wedge \varphi_{v:\bot})$ are obvious, while choosing $l = v$ in result ($\text{Mod}_{l:\top}$) gives: $\varphi_{v:\top} \equiv ForgetV(v \wedge \varphi, v)$.

Thus, we get, for each $\epsilon \in \{\bot, \top\}$:
$(\varphi \vee \psi)_{l:\epsilon} \equiv \varphi_{l:\epsilon} \vee \psi_{l:\epsilon}$, and also $(\varphi \wedge \psi)_{l:\epsilon} \equiv \varphi_{l:\epsilon} \wedge \psi_{l:\epsilon}$.

Remark 6 *Let* φ *be any formula and* l *some literal with* v_l *as its propositional symbol. Then, the following six formulas are all equivalent:*

$$\begin{array}{llll} ForgetV(l \wedge \varphi, v_l) \vee \varphi & & & \equiv \\ (\neg l \wedge ForgetV(l \wedge \varphi, v_l)) \vee \varphi & & & \equiv \\ \varphi_{l:\top} \vee (\neg l \wedge \varphi) & \equiv & \varphi_{l:\top} \vee (\neg l \wedge \varphi_{l:\bot}) & \equiv \\ \varphi_{l:\top} \vee \varphi & \equiv & \varphi \vee (\neg l \wedge \varphi_{l:\bot}). & \end{array}$$

Indeed, the set of the models of each of these formulas is $\{Force(\omega, l) / \omega \models \varphi, \omega \models l\} \cup \{Force(\omega, \sim l) / \omega \models \varphi, \omega \models l\} \cup \{Force(\omega, \sim l) / \omega \models \varphi, \omega \models \neg l\}$.

Forgetting literals

Variable forgetting as been generalized as detailed now, beginning with the semantical side.

Definition 7 (Literal forgetting) *(Lang, Liberatore, & Marquis 2003, Prop. 15) If* φ *is a formula and* L *a set of literals in* $\mathbf{PL}$, $ForgetLit(\varphi, L)$ *is a formula having for models the set of all the interpretations for* $\mathbf{PL}$ *which can be turned into a model of* φ *when forced by a consistent subset of* L*:*

$$\mathbf{Mod}(ForgetLit(\varphi, L)) = \{\omega / Force(\omega, L_1) \models \varphi \text{ and } L_1 \text{ is a consistent subset of } L\}.$$

Thus, the models of $ForgetLit(\varphi, L)$ are built from the models of φ by allowing to "negate" (or "complement") an arbitrary number of values of literals in L:

$$\mathbf{Mod}(ForgetLit(\varphi, L)) = \{Force(\omega', L'_1) \ / \omega' \models \varphi \text{ and } L'_1 \text{ is a consistent subset of } \sim L\}.$$

Let us consider the syntactical side now. One way is to start from a DNF formulation of φ:

Proposition 8 *(Lang, Liberatore, & Marquis 2003) If* $\varphi = t_1 \vee \cdots \vee t_n$ *is a DNF,* $ForgetLit(\varphi, L)$ *is equivalent to the formula* $t'_1 \vee \cdots \vee t'_n$ *where* t'_i *is the term* t_i *without the literals in* L.

The similar method for obtaining $ForgetV(\varphi, V)$ when φ is a DNF has been reminded in point 1 following Definition 2. Similarly, the following syntactical definition, analogous to Definition 3, can be given:

Definition 9 *If* L *is a set of literals in* $\mathbf{PL}$, *then*

$$ForgetLit(\varphi, L) = \bigvee_{L' \subseteq L} \left(\left(\bigwedge \sim L' \right) \wedge \varphi_{\langle (L - L'):\top \rangle} \right).$$

This is a "global formulation", easily shown to be equivalent to the following ***iterative definition*** (Lang, Liberatore, & Marquis 2003, Definition 7):

1. $ForgetLit(\varphi, \emptyset) = \varphi$.

2. $ForgetLit(\varphi, \{l\}) = \varphi_{l:\top} \vee \varphi$.
3. $ForgetLit(\varphi, \{l\} \cup L) = ForgetLit(ForgetLit(\varphi, L), l)$.

We refer the reader to (Lang, Liberatore, & Marquis 2003) which shows the adequacy with Definition 7 and Proposition 8, and also that choosing any order of the literals does not modify the meaning of the final formula (cf Notations 1-3). It follows that this independence from the order of the literals also applies to the global formulation in Definition 9. The fact that, exactly as with the notion of forgetting symbols (cf Definition 2 and following comment), the notion of forgetting literals has such an iterative definition is important from a computational point of view (Lang, Liberatore, & Marquis 2003).

Notice that (Lang, Liberatore, & Marquis 2003) uses the formula $\varphi_{l:\top} \vee (\neg l \wedge \varphi)$ in point 2, and also the variant $\varphi_{l:\top} \vee (\neg l \wedge \varphi_{l:\top})$, instead of $\varphi_{l:\top} \vee \varphi$. Remark 6 shows that any of the six formulas given there could be used here, which could marginally simplify the computation, depending on the form in which φ appears.

The presence of $\left(\bigwedge_{l' \in L'} \neg l'\right)$ in Definition 9, which is what differentiates $ForgetLit(\varphi, ...)$ from $ForgetV(\varphi, ...)$, comes from the fact that here we forget $l \in L$ but we do not forget $l' \in \sim L$.

A proof in (Lang, Liberatore, & Marquis 2003), using Proposition 8, shows that we get $ForgetLit(\varphi, V^{\pm}) \equiv ForgetV(\varphi, V)$. This proof is easily extended to get the following result:

Remark 10 *Since any set of literals can be written as a disjoint union between a consistent set L' and a set $V^{\pm}$ of complementary literals, here is a useful formulation:*

$$ForgetLit(\varphi, L' \cup V^{\pm}) \equiv ForgetLit(ForgetV(\varphi, V), L').$$

Notice that we could also forget the literals first, i.e. consider the formula $ForgetV(ForgetLit(\varphi, L'), V)$, even if it seems likely that this is less interesting from a computational point of view.

This remark has the advantage of separating clearly the propositional symbols into three kinds. Let V' denote the set $\mathcal{V}(L')$ of the propositional symbols in L', and $V'' = \mathcal{V}(\mathbf{PL}) - V - V'$ be the set of the remaining symbols. Then we get:

1. The propositional symbols in V are forgotten.
2. The propositional symbols in V'' are *fixed*, since the literals in $V''^{\pm}$ are not forgotten.
3. The remaining symbols, in V', are neither forgotten nor fixed, since only the literals in L' are forgotten, but not the literals in $\sim L'$.

Thus, $ForgetLit(\varphi, L_1)$ can be defined as: *forgetting literals with some propositional symbols fixed*. It is then natural to generalize the notion, by allowing some propositional symbols to *vary* in the *forgetting* process.

Forgetting literals with varying symbols

As done with the original notion, let us begin with the semantical definitions.

Definition 11 *Let φ be a formula, V a set of propositional symbols, and L a consistent set of literals, in $\mathbf{PL}$, with V and $\mathcal{V}(L)$ disjoint in $\mathcal{V}(\mathbf{PL})$. $ForgetLitVar(\varphi, L, V)$ is a formula having the following set of models:*

$$\mathbf{Mod}(ForgetLitVar(\varphi, L, V)) = \{\omega \,/\, Force(\omega, L_1 \cup L_2) \models \varphi, L_1 \subseteq L, \; L_2 \subseteq V^{\pm}, L_2 \textit{ consistent, and } (\omega \not\models L_1 \textit{ or } L_2 = \emptyset)\}.$$

This is equivalent to:

$$\mathbf{Mod}(ForgetLitVar(\varphi, L, V)) = \mathbf{Mod}(\varphi) \cup \{Force(\omega, L_1 \cup L_2) \,/\, \omega \models \varphi, \omega \not\models L_1, \; L_1 \subseteq\sim L, \;\; L_2 \subseteq_{cons} V^{\pm}\}.$$

Notice the notation $\subseteq_{cons} V^{\pm}$ for "included in $V^{\pm}$ and consistent".

Since $\omega \models L_2$ iff $Force(\omega, L_2) = \omega$, the condition "*($\omega \not\models L_1$ or $L_2 = \emptyset$)*" can be replaced by "*($\omega \not\models L_1$ or $\omega \models L_2$)*", and then we can replace everywhere here "*L_2 consistent*" by "*L_2 consistent and complete in V*" (there are $3^{card(V)}$ consistent sets L_2 and "only" $2^{card(V)}$ consistent and complete sets).

We could be more general, by allowing to forget some propositional symbols, which amounts to allow non consistent sets L. This generalization does not present difficulties, however, since we have not found any application for it till now, we leave it for future work.

With respect to Definition 7, what happens here is that the non consistent part of the set of literals, which allowed to forget some set V of propositional symbols altogether, has been replaced by a set of varying propositional symbols.

Remark 12 *Since $ForgetLit(L_1, \varphi) \models ForgetLit(L_1 \cup L_2, \varphi)$ holds from (Lang, Liberatore, & Marquis 2003) (*"the more we forget, the less we know"*), we get:*

$$\varphi \models ForgetV(\varphi, V) \models ForgetLit(\varphi, L \cup V^{\pm}).$$

Similarly, it is clear that the new definition allows a finer (more cautious) forgetting than $ForgetLit$:

$$\varphi \models ForgetLitVar(\varphi, L, V) \models ForgetLit(\varphi, L \cup V^{\pm}).$$

Remind the motivations for introducing $ForgetLitVar$: we want to "forget" the literals in L, even at the price of modifying the literals in $V^{\pm}$: if we effectively forget at least one literal in L, then, we allow any modification for the literals in $V^{\pm}$. However, we do not want to modify the literals in $V^{\pm}$ "for nothing" our aim being to forget as many literals in L as possible. This justifies the appearance of the condition "$\omega \not\models L_1$" in the definition and in the alternative formulation.

The syntactical aspect is slightly more tricky, but it remains rather simple and it allows to revisit and improve

already known results. As with the original notion (see Proposition 8), the simplest way is to start from a DNF.

Since L is consistent, without loss of generality and in order to simplify the notations, we can consider that L is a set of negative literals (otherwise, replace any $p \in \mathcal{V}(L)$ such that $p \in L$ by $\neg p'$, p' being a new propositional symbol, then after the computations, replace p' by $\neg p$). Thus, till the end of this section, we will consider two disjoint subsets P and V of $\mathcal{V}(\mathbf{PL})$, and $L = P^-$ with $Q = \mathcal{V}(\mathbf{PL}) - V - P$ denoting the set of the remaining propositional symbols.

Proposition 13 *(See proof in Appendix) Let $\varphi = t_1 \vee \cdots \vee t_n$ be a DNF, with*

$$t_i = (\bigwedge P_{i,1}) \wedge (\bigwedge \neg(P_{i,2})) \wedge (\bigwedge V_{i,l}) \wedge (\bigwedge Q_{i,l}),$$

where $P_{i,1} \subseteq P$, $P_{i,2} \subseteq P - P_{i,1}$, with $V_{i,l} \subseteq V^{\pm}$ and $Q_{i,l} \subseteq Q^{\pm}$ being consistent sets of literals. Then $ForgetLitVar(\varphi, P^-, V) \equiv t'_1 \vee \cdots \vee t'_n$ where

$$t'_i = (\bigwedge P_{i,1}) \wedge (\bigwedge Q_{i,l}) \wedge [(\bigvee (P - P_{i,1})) \vee (\bigwedge V_{i,l})], i.e.$$

$$t'_i = (\bigwedge P_{i,1}) \wedge (\bigwedge Q_{i,l}) \wedge [\bigwedge_{l \in V_{i,l}} (l \vee (\bigvee (P - P_{i,1})))].$$

Thus, t'_i is t_i except that the literals in P^- are suppressed while each literal in $V^{\pm}$ must appear in disjunction with the clause $\bigvee(P - P_1)$, this clause denoting the disjunction of all the literals in P^+ which do not appear (positively) in t_i. Naturally, the literals of $L = P^-$ appearing in t_i disappear. Moreover, it is important to notice that the literals from $P^{\pm} = L \cup {\sim}L$ in t_i which remain are those which do not appear positively in t_i. This means that t_i could be "completed in P" by the conjunction of all the $\neg p$ for each symbol $p \in P$ not appearing in t_i, without modifying the "forget" formula.

We have provided the semantical definition (in the lines of Definition 7) and a characterization from a DNF formulation (in the lines of Proposition 8). Let us provide now other characterizations, and a comparison with $ForgetLit$.

Proposition 14 *Let φ be a formula in $\mathbf{PL}$, and P, Q and V be three pairwise disjoint sets of propositional symbols such that $P \cup Q \cup V = \mathcal{V}(\mathbf{PL})$.*

1. *$ForgetLit(\varphi, P^- \cup V^{\pm})$ is equivalent to the set $Th(\varphi) \cap X$ where X is the set of the formulas in $\mathbf{PL}$ which are disjunctions of terms of the kind $(\bigwedge P_1) \wedge (\bigwedge Q_l)$ with $P_1 \subseteq P$ and $Q_l \subseteq Q^{\pm}$.*
2. *$ForgetLitVar(\varphi, P^-, V)$ is equivalent to the set $Th(\varphi) \cap X$ where X is the set of the formulas in $\mathbf{PL}$ which are disjunctions of terms of the kind $(\bigwedge P_1) \wedge (\bigwedge Q_l) \wedge [\bigwedge_{l \in V_l} (l \vee (\bigvee (P - P_1)))]$, where $P_1 \subseteq P$, $V_l \subseteq_{cons} V^{\pm}$ and $Q_l \subseteq Q^{\pm}$.*

(We can clearly consider consistent sets Q_l only.)

These two results are immediate consequences of Propositions 8 and 13 respectively. We get the following alternative possibilities for the sets X's, firstly by boolean duality from the preceding results, then by considering some set having the same $\wedge$-closure as X (Remark 4):

Proposition 14 (following)

1(a) *For $ForgetLit(\varphi, P^- \cup V^{\pm})$, X is the set of the conjunctions of the clauses of the kind $(\bigvee P_1) \vee (\bigvee Q_l)$ with $P_1 \subseteq P$ and $Q_l \subseteq Q^{\pm}$ (we can clearly consider consistent sets Q_l only).*

(b) *We can also consider the set X of the clauses $(\bigvee P_1) \vee (\bigvee Q_l)$ with $P_1 \subseteq P$ and $Q_l \subseteq Q^{\pm}$.*

(c) *The smallest set X possible is the set of the clauses $(\bigvee P_1) \vee (\bigvee Q_l)$ with $P_1 \subseteq P$, $Q_l \subseteq Q^{\pm}$, Q_l consistent and complete in Q.*

2(a) *For $ForgetLitVar(\varphi, P^-, V)$, X is the set of the conjunctions of the formulas $flv(P_1, Q_l, V_l) = (\bigvee P_1) \vee (\bigvee Q_l) \vee \bigvee_{l \in V_l} (l \wedge (\bigwedge (P - P_1)))$, where $P_1 \subseteq P$, $V_l \subseteq_{cons} V^{\pm}$ and $Q_l \subseteq_{cons} Q^{\pm}$.*

(b) *We can also consider the set X of all the formulas $flv(P_1, Q_l, V_l)$ of this kind.*

(c) *The smallest set X possible is the set of the formulas $flv(P_1, Q_l, V_l)$ with $P_1 \subseteq P$, Q_l and V_l being sets of literals consistent and complete in Q and V respectively.*

These results provide the analogous, for $ForgetLit$ and $ForgetLitVar$, of the results for $ForgetV$ reminded in Definition 2, and in Remark 4.

The next definition is analogous to Definitions 3 and 9 (see appendix for a proof of the adequacy with Definition 11):

Definition 15 *If φ is a formula and P and V are two disjoint subsets of $\mathcal{V}(\mathbf{PL})$, then*
$ForgetLitVar(\varphi, P^-, V)$ is the formula

$$\bigvee_{P_1 \subseteq P} \Big(\bigwedge P_1 \wedge (\varphi_{[P_1:\top,(P-P_1):\bot]} \vee (ForgetV(\varphi_{[P_1:\top,(P-P_1):\bot]}, V) \wedge (\bigvee (P - P_1)))) \Big).$$

Example 2 *Here $P = \{a, b\}, V = \{c\}, Q = \{d\}$, with $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d)$.*

Syntactical side:

Since φ is a DNF, the rules from a DNF after Definition 2 (for $ForgetV$), in Proposition 8 (for $ForgetLit$) and Proposition 13 (for $ForgetLitVar$), give the three results:

- $ForgetV(\varphi, V) \equiv (\neg a \wedge b) \vee (a \wedge \neg b \wedge \neg d)$.
- $ForgetLit(\varphi, P^- \cup V^{\pm}) \equiv b \vee (a \wedge \neg d)$.
- $ForgetLitVar(\varphi, P^-, V) \equiv (a \wedge b) \vee (a \wedge \neg c \wedge \neg d) \vee (b \wedge c)$. $\quad FLV1$

Definitions 9 and 15 can be used also, as shown now for Definition 15 where, in each case, $\psi = \varphi_{[P_1:\top,(P-P_1):\bot]}$:

$P_1 = \emptyset : \psi \vee (ForgetV(\psi, c) \wedge (a \vee b)) \equiv \bot \vee (\bot \wedge (a \vee b)) \equiv \bot.$ $\quad (\varphi_1)$

$P_1 = \{a\} : a \wedge (\psi \vee (ForgetV(\psi, c) \wedge b)) \equiv a \wedge ((\neg c \wedge \neg d) \vee (\neg d \wedge b)).$ $\quad (\varphi_2)$

$$P_1 = \{b\} : b \wedge (\psi \vee (ForgetV(\psi, c) \wedge a)) \equiv b \wedge (c \vee (\top \wedge a)). \quad (\varphi_3)$$

$$P_1 = \{a, b\} : a \wedge b \wedge (\psi \vee (ForgetV(\psi, c) \wedge \bot)) \equiv a \wedge b \wedge (\bot \vee \bot) \equiv \bot. \quad (\varphi_4)$$

The disjunction $\bigvee_{i=1}^{4} \varphi_i$ *is equivalent to* $FLV1$.

Semantical side:

We get $\mathbf{Mod}(\varphi) = \{\{a\}, \{b, c\}, \{b, c, d\}\}$.

- *The six models of* $ForgetV(\varphi, V)$ *are obtained by adding the three interpretations differing from the three models of* φ *by the value attributed to* c *(cf example 1):* $\{a, c\}, \{b\}$, *and* $\{b, d\}$.
- *The ten models of* $ForgetLit(\varphi, P^- \cup V^{\pm})$ *are obtained by adding to the models of* φ *the seven interpretations differing from these models by adding any subset of* $\{a, b\}$ *and by either do nothing else or modify the value of* c *(adding* c *if it is not present and removing* c *if it is present). This gives the six models of* $ForgetV(\varphi, V)$ *plus the four interpretations including* $\{a, b\}$.
- *The seven models of* $ForgetLitVar(\varphi, P^-, V)$ *are obtained by adding to the three models of* φ *the four interpretations differing from these models by adding a* non empty *subset of* $\{a, b\}$ *and by either do nothing else or modify the value of* c*, which gives here the four interpretations including* $\{a, b\}$.

Here is a technical result which can be drawn from this example, and which may have a computational interest:

Remark 16 *1. For any formula* φ *we get:* $ForgetV(\varphi, V) \vee ForgetLitVar(\varphi, P^-, V) \equiv ForgetLit(\varphi, P^- \cup V^{\pm})$

2. *For any formula* φ *which is* uniquely defined in P*, we get:* $ForgetV(\varphi, V) \wedge ForgetLitVar(\varphi, P^-, V) \equiv \varphi$.

By formula uniquely defined in P *we mean a formula which is equivalent to a conjunction* $\varphi_1 \wedge \varphi_2$*, where* φ_1 *is a term complete in* P *and* φ_2 *is without symbol of* P.

See the Appendix for a proof. This remark can be compared with Remark 12. Notice that in Example 2, the formula φ is uniquely defined in P [indeed, $\varphi \equiv (\neg a \wedge b) \wedge (c \vee (\neg c \wedge \neg d))$], thus points 1. and 2. of this Remark are satisfied. Here is a simple counter-example (where the important fact to notice is that φ is a term which is *not complete in* P, i.e. $P_{i,1} \cup P_{i,2} \neq P$) showing that the second equivalence does not hold for any formula.

Example 3 P, V, Q*, and* $\mathbf{PL}$ *as in example 2,* $\varphi = t = a \wedge c$. *We get:*

- $ForgetV(t, V) \equiv a$.
- $ForgetLit(t, P^- \cup V^{\pm}) \equiv a$.
- $ForgetLitVar(t, P^-, V) \equiv ForgetLitVar(a \wedge \neg b \wedge c, P^-, V) \equiv a \wedge (b \vee c)$.

Notice also that, once we have all the models of φ, the complexity of the construction of all the models of $ForgetLitVar(\varphi, P^-, V)$ is not greater than the complexity of the construction of all the models of $ForgetLit(\varphi, P^- \cup V^{\pm})$.

More about the computation of these notions

On the syntactical side, we have the same kind of iterative definition than we had for $ForgetV$ and $ForgetLit$ (cf the two "iterative definitions", in Point 2 just before Definition 3 for $ForgetV$, and after Definition 9 for $ForgetLit$):

Remark 17 *Let us suppose that* V *is a set of propositional symbols and that* $L \cup \{l\}$ *is a consistent set of literals without symbol in* V *and such that* $l \notin L$.

1. $ForgetLitVar(\varphi, \emptyset, V) = \varphi$;
2. $ForgetLitVar(\varphi, \{l\}, V) = \varphi \vee ForgetV(\neg l \wedge ForgetV(l \wedge \varphi, v_l), V)$ *(where* v_l *denotes the symbol of* l*).*
3. $ForgetLitVar(\varphi, \{l\} \cup L, V) = ForgetLitVar(ForgetLitVar(\varphi, L, V), \{l\}, V)$.

We get equivalent formulas for each order of appearance of the literals in the iterative process. The complexity of the computation of $ForgetLitVar(\cdots, L, V)$ should be only slightly harder than for the computation of $ForgetLit$. Indeed, we have to "forget V" for each new literal, which introduces a rather small new complication, otherwise, computing $\neg l \wedge ForgetV(l \wedge ForgetLitVar(\varphi, L, V), v_l)$ is not harder than computing $ForgetLit(ForgetLit(\varphi, L), l)$.

See the appendix for the proof of the equivalence with Definition 15. Notice already that the formula $(\neg l \wedge ForgetV(l \wedge \Phi, v_l))$ has for models the models of Φ which are *actively forced by* $\neg l$ (l was true in the initial model, and l is forced to be false).

Formally, $\mathbf{Mod}(\neg l \wedge ForgetV(l \wedge \Phi, v_l)) = \{Force(\omega, \sim l)/\omega \models \Phi \wedge l\}$. (M¬lFVl)

It seems important, from a computational point of view, to describe an alternative syntactical way to compute this formula (besides the possibility of using the formulation in $ForgetV$ given above). Here is a syntactical method.

From (M¬lFVl), we get

$$\neg l \wedge ForgetV(l \wedge \Phi, v_l) \equiv \neg l \wedge [l \wedge \Phi]_{l:\top}. \quad (\text{F}\neg\text{lFVl})$$

An interesting point in the proof of the equivalence between Remark 17 and Definition 15 is that it shows how to improve the computation a bit. Indeed, once a model has been modified by some $l \in L$, the set of all its variant in V (i.e. the set $\{Force(\omega, L_2)/L_2 \subseteq_{cons} V\pm\}$) is already computed. Thus, for such a model, it is useless to compute again all the variants in V, since they are already present, and forgetting one more literal in L will have no consequence to that respect: since we had already all the variants in V, modifying a new symbol brings only one more model (at most, it was not already present) without the need to consider again all the variants in V for this model.

This gives rise to the following iterative process:

1. $ForgetLitVar(\varphi, \emptyset, V) = \varphi$;
2. $ForgetLitVar(\varphi, \{l\} \cup L, V) =$
$\Phi \vee \Phi_{l:\top} \vee ForgetV(\neg l \wedge [l \wedge \varphi]_{l:\top}, V)$
where $\Phi = ForgetLitVar(\varphi, L, V)$.

Remind that $\neg l \wedge [l \wedge \varphi]_{l:\top}$ can be replaced by $\neg l \wedge ForgetV(l \wedge \varphi, v_l)$ (see formula (F¬lFVl).

The simplification with respect to Remark 17 comes from the fact that only the "fixed" formula φ is considered when forgetting the symbols in V, instead of the "moving" formula $ForgetLitVar(\varphi, L, V)$. This can be interesting, since φ can be simplified before the computations, which will then be facilitated.

Let us apply this improved iterative method to Example 2:

Example 4 *cf Example 2:* $P = \{a, b\}, V = \{c\}, Q = \{d\}$, *with* $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d)$.

- *We compute* $ForgetLitVar(\varphi, P, V)$ *again:*

1. $\Phi^0 = ForgetLitVar(\varphi, \emptyset, \{c\}) = \varphi$;
2. $\Phi^1 = \Phi^0 \vee \Phi^0_{\neg a:\top} \vee ForgetV(a \wedge [\neg a \wedge \varphi]_{\neg a:\top}, c) \equiv$
$((\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d)) \vee$
$(b \wedge c) \vee ForgetV(a \wedge (b \wedge c), \{c\}) \equiv$
$(\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d)) \vee (b \wedge c) \vee (a \wedge b) \equiv$
$(a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (a \wedge b) \vee (b \wedge c)$;
3. $ForgetLitVar(\varphi, P, V) = \Phi^2 =$
$\Phi^1 \vee \Phi^1_{\neg b:\top} \vee ForgetV(b \wedge [\neg b \wedge \varphi]_{\neg b:\top}, c)) \equiv$
$((a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (a \wedge b) \vee (b \wedge c)) \vee (a \wedge \neg c \wedge \neg d) \vee$
$ForgetV(b \wedge (a \wedge \neg c \wedge \neg d), \{c\}) \equiv$
$(a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (a \wedge b) \vee (b \wedge c) \vee (a \wedge \neg c \wedge \neg d) \vee$
$(a \wedge b \wedge \neg d) \equiv$
$(a \wedge b) \vee (b \wedge c) \vee (a \wedge \neg c \wedge \neg d)$ *(cf Example 2).*

Conclusion and perspectives

Why could this work be useful:

The notion of forgetting literals consists in small manipulations of propositional formulas. This notion can help the effective computation of various useful already known knowledge representation formalisms. As shown in (Lang, Liberatore, & Marquis 2003), we cannot hope that this will solve all the problems, but it should help in providing significant practical improvements. And the introduction of varying symbols while forgetting literals should enhance these improvements in a significant way. However, the present text has not developed this applicative matter. Let us just remind a few indications on this subject now [see (Lang, Liberatore, & Marquis 2003; Moinard 2005) for more details]. Various knowledge representation formalisms are known to be concerned, we will only evoke circumscription.

Circumscription (McCarthy 1986) is a formalism aimed at *minimizing* some set of propositional symbols. For instance, circumscribing the symbol $exceptional$ in the sub-formula $bird \wedge \neg exceptional \rightarrow flies$ of our introductory example would conclude $\neg exceptional$ since it is compatible with the sub-formula that "no exception" happens. Notice that even on this simple example a complication appears: we cannot "circumscribe" $exceptional$ alone, if we want the expected minimization to hold here. Instead, we must also allow at least one other symbol to *vary* during the circumscription (e.g. we could allow $flies$ to *vary* while $exceptional$ is *circumscribed*). Circumscription is used in action languages and other formalizations of common sense reasoning, but a key and limiting issue is the efficient computation. The notion of forgetting literals provides a (limited, but real) progress on the subject. The main result is the following one:

$Circ(P, Q, V)(\varphi) \models \psi$ iff
$\varphi \models ForgetLitVar(\varphi \wedge \psi, P^-, V)$.

The propositional symbols in P, V, Q are respectively *circumscribed*, *varying*, and *fixed* in the "circumscription of the formula φ" here.

This result is known to improve (from a computational perspective) previously known results, mainly a result from (Przymusinski 1989). The notion of varying symbols allows some simplification with respect to Przymunsinski's method and even with respect to the computational improvements of this method discovered by (Lang, Liberatore, & Marquis 2003).

What has been done here:

We have provided the semantical and several syntactical characterizations for a new notion, extending the notion of literal forgetting introduced in (Lang, Liberatore, & Marquis 2003) to the cases where some propositional symbols are allowed to vary. These results show that the new notion is not significantly harder than literal forgetting without varying symbols. The various characterizations provide effective ways for computing the results, depending on the form in which the formulas appear. These different ways for computing the notions introduced should help the effective computation in many cases. This is why we have provided several equivalent formulas for the main formulas introduced here, and also for some important auxiliary formulas involved in the definitions. This kind of work is absolutely necessary when coming to the effective computation. Indeed, as shown in (Lang, Liberatore, & Marquis 2003), no formulation can be considered as the best one in any case.

Hopefully, the various ways of defining the formulas and notions introduced here could also help getting a better grasp of these notions, since they are not very well known till now.

What remains to be done:

Various knowledge representation formalisms are known to be concerned (Lang, Liberatore, & Marquis 2003). Moreover, it is highly probable that these notions of forgetting literals for themselves can give rise to new useful formalizations of old problems in knowledge representation. It seems even likely that new knowledge representation formalisms could emerge from these enhanced notions of "forgetting".

More concretely, the notion of "forgetting" can still be generalized: we could directly "forget formulas" (instead of just "literals"), in the lines of what has been done with formula circumscription with respect to predicate circumscription.

Again more concretely, the present work [after the initiating work of (Lang, Liberatore, & Marquis 2003)] has given a preliminary idea on what kind technical work can be done for simplifying the effective computation of the formulas involved in the forgetting process. It is clear that a lot of important work should still be done on the subject.

Also, the complexity results, which have been described in (Lang, Liberatore, & Marquis 2003), should be extended to the new notion, and to the new methods of computation. This is far from simple since, as shown in (Lang, Liberatore, & Marquis 2003), it seems useless to hope for a general decrease of complexity with respect to the already known methods. So, the methods should be examined one by one, and for each method, its range of utility (the particular formulations for a given formula φ for which the method is interesting) should be discovered and discussed.

Appendix

Proof of Proposition 13:

Let us consider complete terms first, such as

$$t_i = t = (\bigwedge P_1) \wedge (\bigwedge \neg(P - P_1)) \wedge (\bigwedge V_l) \wedge (\bigwedge Q_l),$$

where $P_1 \subseteq P$, V_l and Q_l being consistent and complete sets of literals in V and Q respectively. t corresponds to an interpretation ω. The set $F(\omega) = \{Force(\omega, L_1 \cup L_2) \ / \ L_1 \subseteq P, \ L_2 \subseteq V^{\pm}, L_2$ consistent and complete in V, and $\omega \not\models L_1$ or $\omega \models L_2\}$ is the set of the models of the formula $t^1 \wedge t^2$ where $t^1 = (\bigwedge P_1) \wedge (\bigwedge Q_l)$ and $t^2 = \neg(\bigwedge \neg(P - P_1)) \vee (\bigwedge V_l)$, i.e. $t^2 \equiv (\bigvee(P - P_1)) \vee (\bigwedge V_l))$.

Indeed, for each $\omega' \in F(\omega)$, t^1 holds since it holds in ω, and the symbols in $P - P_1$ and V can take any value satisfying the condition $\omega \not\models L_1$ or $\omega \models L_2$. Since $\omega \models t$, this means $L_1 \cap (P - P_1) \neq \emptyset$ or $L_2 \subseteq V_l$, which is equivalent to $\omega' \models t_2$. Conversely, any model ω'' of $t_1 \wedge t_2$ is easily seen to be in $F(\omega)$.

The same result holds for any (consistent) term $t = t_i = (\bigwedge P_1) \wedge (\bigwedge \neg(P_2)) \wedge (\bigwedge V_l) \wedge (\bigwedge Q_l)$, where $P_1 \subseteq P$, $P_2 \subseteq P - P_1$, V_l and Q_l being consistent subsets of $V^{\pm}$ and $Q^{\pm}$ respectively: Let us first consider separately the cases where some symbols in P are missing, then symbols in V, then symbols in Q.

(1) If $p \in P$ does not appear in t, for any model ω' of t, $\omega'' = Force(\omega', \{\neg p\})$ and $Force(\omega'', \{p\})$ are two models of t (one of these is ω'). By considering all the missing p's, we get that the set $\{Force(\omega', L_1 \cup L_2)/\omega' \models t, \ L_1 \subseteq P^-, L_2 \subseteq_{cons} V^{\pm}, \omega' \not\models L_1$ or $L_2 = \emptyset\}$ is included in the set $\{Force(\omega'', L_1 \cup L_2)/\omega'' \models t \wedge \bigwedge \neg(P - P_1), L_1 \subseteq P^-, L_2 \subseteq_{cons} V^{\pm}, \omega'' \not\models L_1$ or $L_2 = \emptyset\}$. Thus any missing p in t behaves as if the negative literal $\neg p$ was present: we get a term "completed in P" satisfying $ForgetLitVar(t, P^-, V) \equiv ForgetLitVar(t \wedge \neg(P - P_1), P^-, V)$.

(2) The reasoning for a missing q in t ($q \in Q$) is simpler yet: if some $q \in Q$ does not appear in t, it can be interpreted as false or true for any model of $ForgetLitVar(t, L, Q)$, which means that we keep the part $\bigwedge Q_l$ unmodified, exactly as in the case where Q_l is complete in Q.

(3) The case for V is similar (the disjunction of all the formulas with all the possibilities for the missing symbols gives the formula where these symbols are missing): If some $v \in V$ is missing in t, then any model ω' of t has its counterpart where the value for v is modified. Let us call V_m the set of the symbols in V which are absent in t. By considering the disjunctions of all the possibilities, we get the formula $\bigvee_{V_l' \in \mathcal{L}_m}((\bigwedge P_1) \wedge (\bigwedge Q_l) \wedge ((\bigvee(P - P_1)) \vee (\bigwedge V_l \wedge \bigwedge V_l')))$, where $\mathcal{L}_m$ is the set of all the sets of literals consistent and complete in V_m. This is equivalent to the formula $(\bigwedge P_1) \wedge (\bigwedge Q_l) \wedge ((\bigvee(P - P_1)) \vee (\bigwedge V_l))$.

Combining "the three incompleteness" (1)–(3) gives:
$ForgetLitVar(t_i, P^-, V) \equiv (\bigwedge P_1) \wedge (\bigwedge Q_l) \wedge ((\bigvee(P - P_1)) \vee (\bigwedge V_l))$.
The disjunction for all the t_i's gives the result. □

Proof of the adequacy of Definition 15 with Definition 11:

Each model ω of φ gives rise to the following models of $ForgetLitVar(\varphi, P^-, V)$:

- ω itself, model of $\psi_1 = \bigwedge P_1 \wedge \bigwedge \neg(P - P_1) \wedge \varphi_{[P_1:\top, \ (P-P_1):\bot]}$ where $P_1 = \omega \cap P$, together with
- all the interpretations differing from ω in that they have at least one more $p \in P$, and no constraint holds for the symbols in V; this set of interpretations being the set of models of the formula $\psi_2 = \bigwedge P_1 \wedge ForgetV(\varphi_{[P_1:\top, \ (P-P_1):0]}, V) \wedge \bigvee(P - P_1)$.

Since $\varphi_{[P_1:\top,(P-P_1):\bot]} \models ForgetV(\varphi_{[P_1:\top,(P-P_1):\bot]}, V)$ and $\bigwedge \neg(P - P_1) \equiv \neg(\bigvee(P - P_1))$, when considering the disjunction $\psi_1 \vee \psi_2$, we can suppress $\wedge \bigwedge \neg(P - P_1)$ in ψ_1. The disjunction of all these formulas $\psi_1 \vee \psi_2$ for each model ω of φ, gives the formula as written in this definition. □

Proof of Remark 16:

1. For any formula φ, **Mod**$(ForgetV(\varphi, V)) = \{Force(\omega, L_2)/L_2 \subseteq_{cons} V^{\pm}\} = \{Force(\omega, L_1 \cup L_2)/L_1 \subseteq P^-, L_2 \subseteq_{cons} V^{\pm}, \omega \models L_1\}$ and **Mod**$(ForgetLitVar(\varphi, P^-, V)) = \{Force(\omega, L_1 \cup L_2)/L_1 \subseteq P^-, L_2 \subseteq_{cons} V^{\pm}, [\omega \not\models L_1$ or $L_2 = \emptyset]\}$. Thus, **Mod**$(ForgetV(\varphi, V) \vee ForgetLitVar(\varphi, P^-, V)) =$ **Mod**$(ForgetV(\varphi, V)) \cup$ **Mod**$(ForgetLitVar(\varphi, P^-, V)) = \{Force(\omega, L_1 \cup L_2)/L_1 \subseteq P^-, L_2 \subseteq_{cons} V^{\pm}\} =$ **Mod**$(ForgetLit(\varphi, P^- \cup V^{\pm}))$.
2. We get **Mod**$(ForgetV(\varphi, V) \wedge ForgetLitVar(\varphi, P^-, V)) =$ **Mod**$(ForgetV(\varphi, V)) \cap$ **Mod**$(ForgetLitVar(\varphi, P^-, V))$. Let us suppose now that φ is a formula uniquely defined in P. This means that the set **Mod**$(\varphi) \cap P$ is a singleton. Then, if $L_1 \subseteq P^-$, $\omega \models \varphi$ and $\omega \not\models L_1$, we get $Force(\omega, L_1) \notin$ **Mod**(φ),

and also, for any $\omega' \in \mathbf{Mod}(\varphi)$ and any consistent subsets L_2, L_2' of $V^{\pm}$, $Force(\omega, L_1 \cup L_2) \neq Force(\omega', L_2')$. Thus, for any element $Force(\omega, L_1 \cup L_2)$ of $\mathbf{Mod}(ForgetLitVar(\varphi, P^-, V))$ which is also in $\mathbf{Mod}(ForgetV(\varphi, V)$, we get $\omega \models L_1$, thus also $L2 = \emptyset$, thus $Force(\omega, L_1 \cup L_2) = \omega$, thus this element is in $\mathbf{Mod}(\varphi)$. Thus we get $ForgetV(\varphi, V) \wedge ForgetLitVar(\varphi, P^-, V) \models \varphi$, and, by Remark 12, $ForgetV(\varphi, V) \wedge ForgetLitVar(\varphi, P^-, V) \equiv \varphi$. □

Proof of the adequacy of Remark 17 with Definition 15:

Let V be a set of propositional symbols and $L \cup \{l\}$ be a consistent set of literals without symbol in V such that $l \notin L$.

For any formula Φ, we have $\mathbf{Mod}(\neg l \wedge ForgetV(l \wedge \Phi, v_l)) = \{Force(\omega, \sim l)/\omega \models \Phi, \omega \models l\}$.
This is the set of all the models of Φ *actively forced by* $\sim l$: l was satisfi ed by ω while $Force(\omega, \sim l)$ differs from ω in that it satisfi es $\neg l$. Then we get
$\mathbf{Mod}(ForgetV(\neg l \wedge ForgetV(l \wedge \Phi, v_l), V)) =$
$\{Force(Force(\omega, \sim l), L_2)/\omega \models \Phi, \omega \models l, L_2 \subseteq_{cons} V\pm\} =$
$\{Force(\omega, \{\sim l\} \cup L_2)/\omega \models \Phi, \omega \models l, L_2 \subseteq_{cons} V\pm\}$.

Thus, from Defi nition 11, wet get
$\mathbf{Mod}(ForgetLitVar(\varphi, L, V)) = \mathbf{Mod}_1 \cup \mathbf{Mod}_2$ and
$\mathbf{Mod}(ForgetV(\neg l \wedge ForgetV(l \wedge ForgetLitVar(\varphi, L, V), v_l), V)) = \mathbf{Mod}_3 \cup \mathbf{Mod}_4$
where

1. $\mathbf{Mod}_1 = \{\omega/\omega \models \varphi\}$;
2. $\mathbf{Mod}_2 = \{Force(\omega, L_1 \cup L_2)/\omega \models \varphi, \omega \not\models L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm\}$;
3. $\mathbf{Mod}_3 = \{Force(\omega, \{\sim l\} \cup L_2)/\omega \models \varphi, \omega \models l, L_2 \subseteq_{cons} V\pm\}$;
4. $\mathbf{Mod}_4 = \{Force(Force(\omega, L_1 \cup L_2), \{\sim l\} \cup L_2')/\omega \models \varphi, \omega \models l, \omega \not\models L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm, L_2' \subseteq_{cons} V\pm\}$.

Notice that we get: $v_l \notin L, v_l \notin V$ and $\mathcal{V}(L \cup \{l\}) \cap V = \emptyset$. Thus we get
$\mathbf{Mod}_4 =$
$\{Force(\omega, \{\neg l\} \cup L_1 \cup L_2' \cup (L_2 - \sim L_2'))/\omega \models \varphi, \omega \models l, \omega \not\models L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm, L_2' \subseteq_{cons} V\pm\}$.

When the sets L_2 and L_2' run over the set of the consistent subsets of $V^{\pm}$, the set $L_2'' = L_2' \cup (L_2 - \sim L_2')$ also runs over the same set and we get:

$\mathbf{Mod}_4 = \{Force(\omega, \{\sim l\} \cup L_1 \cup L_2'')/\omega \models \varphi, \omega \models l, \omega \not\models L_1, L_1 \subseteq\sim L, L_2'' \subseteq_{cons} V\pm\}$.

If $L_1 \subseteq\sim L$ and $\omega \models L_1$, we get
$Force(\omega, \{\sim l\} \cup L_2) = Force(\omega, \{\sim l\} \cup L_1 \cup L_2)$.

Thus we get $\mathbf{Mod}_3 \cup \mathbf{Mod}_4 = \mathbf{Mod}_{34} =$
$\{Force(\omega, \{\sim l\} \cup L_1 \cup L_2)/\omega \models \varphi, \omega \models l, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm\}$.

Similarly, if $\omega \not\models l$ (i.e. $\omega \models \neg l$), we get
$Force(\omega, L_1 \cup L_2) = Force(\omega, \{\sim l\} \cup L_1 \cup L_2)$.
Thus we get $\mathbf{Mod}_2 = \mathbf{Mod}_{2a} \cup \mathbf{Mod}_{2b}$ where:
$\mathbf{Mod}_{2a} = \{Force(\omega, \{\sim l\} \cup L_1 \cup L_2)/\omega \models \varphi, \omega \not\models l, \omega \not\models L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm\}$ and
$\mathbf{Mod}_{2b} = \{Force(\omega, L_1 \cup L_2)/\omega \models \varphi, \omega \not\models L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm\} =$
$\{Force(\omega, L_1' \cup L_2)/\omega \models \varphi, \omega \not\models L_1', \neg l \notin L'1, L_1' \subseteq \{\sim l\} \cup \sim L, L_2 \subseteq_{cons} V\pm\}$.

Since $\omega \not\models \{l\} \cup L_1$ iff $\omega \not\models l$ or $\omega \not\models \cup L_1$, we get:
$\mathbf{Mod}_{2a} \cup \mathbf{Mod}_{34} = \mathbf{Mod}_{2a34} =$
$\{Force(\omega, \{\sim l\} \cup L_1 \cup L_2)/\omega \models \varphi, \omega \not\models \{\sim l\} \cup L_1, L_1 \subseteq\sim L, L_2 \subseteq_{cons} V\pm\} =$
$\{Force(\omega, L_1' \cup L_2)/\omega \models \varphi, \omega \not\models L_1', L_1' \subseteq \{\sim l\} \cup \sim L, \sim l \in L_1', L_2 \subseteq_{cons} V\pm\}$.

Thus we get $\mathbf{Mod}_{2a34} \cup \mathbf{Mod}_{2b} =$
$\mathbf{Mod}_{234} = \{Force(\omega, L_1 \cup L_2)/\omega \models \varphi, \omega \not\models L_1, L_1 \subseteq \{\sim l\} \cup \sim L, L_2 \subseteq_{cons} V\pm\}$.

Finally we get the result which achieves the proof:
$\mathbf{Mod}(ForgetLitVar(\varphi, L, V) \vee ForgetV(\neg l \wedge ForgetV(l \wedge ForgetLitVar(\varphi, L, V), v_l), V)) =$
$\mathbf{Mod}_1 \cup \mathbf{Mod}_2 \cup \mathbf{Mod}_3 \cup \mathbf{Mod}_4 = \mathbf{Mod}_1 \cup \mathbf{Mod}_{234} =$
$\mathbf{Mod}(ForgetLitVar(\varphi, \{l\} \cup L, V))$.

Thus, we have shown:
$ForgetLitVar(\varphi, \{l\} \cup L, V) = ForgetLitVar(\varphi, L, V) \vee ForgetV(\neg l \wedge ForgetV(l \wedge ForgetLitVar(\varphi, L, V), v_l), V)$. □

References

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional Independence - Formula-Variable Independence and Forgetting. *(Electronic) Journal of Artificial Intelligence Research* 18:391–443. `http://WWW.JAIR.ORG/`.

Lin, F., and Reiter, R. 1994. Forget it! In Mellish, C. S., ed., *AAAI Fall Symposium on Relevance*, 1985–1991. New Orleans, USA: Morgan Kaufmann.

Lin, F. 2001. On strongest necessary and weakest suffi cient conditions. *Artficial Intelligence* 128(1–2):143–159.

McCarthy, J. 1986. Application of circumscription to formalizing common sense knowledge. *Artificial Intelligence* 28(1):89–116.

Moinard, Y. 2005. Forgetting literals with varying propositional symbols. In McIlraith, S.; Peppas, P.; and Thielscher, M., eds., 7^{th} *Int. Symposium on Logical Formalizations of Common Sense Reasoning*, 169–176.

Przymusinski, T. C. 1989. An Algorithm to Compute Circumscription. *Artificial Intelligence* 38(1):49–73.

Su, K.; Lv, G.; and Zhang, Y. 2004. Reasoning about Knowledge by Variable Forgetting. In Dubois, D.; Welty, C. A.; and Williams, M.-A., eds., *KR'04*, 576–586. AAAI Press.

2.12 Handling (un)awareness and related issues in possibilistic logic: A preliminary discussion

Handling (un)awareness and related issues in possibilistic logic: A preliminary discussion

Henri Prade

IRIT
118 route de Narbonne
31062 Toulouse Cedex 9
prade@irit.fr

Abstract

Possibilistic logic has been developed as a framework where classical logic formulas are associated with levels that express partial certainty, or encode priority when handling contexts in non-monotonic reasoning, or specifying goals when modeling preferences. Thus, basic features of possibilistic logic are the fact that it deals with layered sets of formulas, and it can handle incomplete, uncertain and inconsistent information. In this paper, we provide a preliminary discussion of how different forms of (un)awareness could be processed, in possibilistic logic taking advantage of the layered structure and of the different modalities available.

1 Introduction

Agents may be unaware of propositions that are true. Clearly, this lack of knowledge may affect their capabilities for making right judgments and good choices. This is why a proper representation of awareness and unawareness is of interest in economic modeling.

In their approach to this concern, Modica and Rustichini (1999) claim "that simple uncertainty is not an adequate model of a subject's ignorance, because a major component of it is the inability to give a complete description of the states of the world, and we provide a formal model of unawareness" and that "without weakening the inference rules of the logic one would face the unpleasant alternative between full awareness and full unawareness".

Indeed a first attempt to model the awareness of a proposition a, as being equivalent to the knowledge of a or to the knowledge that a is not known, was made by Modica and Rustichini (1994) in a modal logic setting, following an earlier proposal of Fagin and Halpern (1988) where knowledge and unawareness were handled by the means of separate modalities. A new proposal by Modica and Rustichini (1999) is further discussed in Halpern (2001).

In this note we provide a preliminary discussion and investigation of what kinds of (un)awareness might be captured in the framework of possibility theory and possibilistic logic. This setting allows for there presentation of qualitative uncertainty thanks to a limited use of graded modalities in agreement with propositional logic. States of complete ignorance can be easily represented in possibility theory, which contrasts with the probabilistic framework. However, the handling of some forms of unawareness go beyond the simple representation of uncertainty and ignorance.

We first start with an informal discussion of (un)awareness may mean. Then, we progressively restate the knowledge representation capabilities of the possibility theory and possibilistic logic settings, and point out how the layered structure of possibilistic knowledge base and the use of different modalities in possibility theory, can be useful for capturing various forms of (un)awareness.

2 Being (un)aware of what ?

Intuitively speaking, the idea of unawareness relates to the distinction between implicit and explicit knowledge (Halpern, 2001), where explicit knowledge implies implicit knowledge, while the converse may not hold. Then, one possible understanding of unawareness, is to see it as due to limited reasoning capabilities.

An agent may be aware of a, just because he knows a (i. e., he knows that a is true). Being aware of a and aware of b in this trivial sense, he might be unaware that he should know $a \vee b$, or that he should know $a \wedge b$ also. This is limited omniscience and limited reasoning capabilities.

An agent may be aware that himself, or another agent, does not know if *a* is true or if *a* is false, but he may be also aware that himself, or another agent, or any other agent cannot know if *a* is true or if *a* is false. Consequently, he or another agent cannot claim that *b* is true if they have no direct knowledge about *b*, and if they are unaware of any formula involving *b* except $\neg a \vee b$.

Clearly, there are other forms of unawareness. In particular, the agent may have never heard of *a* or of $\neg a$, and then may have never figured out if *a* might be true or if *a* might be false.

In the following, we discuss how these different forms of (un)awareness could be handled in a possibilistic setting.

3 Limited awareness of clauses

The core of possibilistic logic, which uses bounds on necessity measures only, is first recalled. Then its use for distinguishing between formulas that the agent is aware that they are true from formulas that would require a higher level of awareness, is outlined. The approach also enables us to control and limit the awareness of disjunctions of formulas that the agent is aware of.

3.1 Background on possibilistic logic

A possibilistic logic formula is essentially a pair made of a classical first order logic formula and of a weight, which expresses certainty or priority (Dubois *et al.*, 1994). In possibilistic logic, the weight associated with a formula *a* is semantically interpreted in terms of lower bounds $\alpha \in (0,1]$ of a necessity measure, i.e., a possibilistic logic expression (a, α) is understood as $N(a) \geq \alpha$, where N is a necessity measure.

More generally, from a semantic point of view, a possibilistic knowledge base $K = \{(a_i, \alpha_i)\}_{i=1,n}$ is understood as the possibility distribution π_K representing a fuzzy set of models of K on the set Ω of the interpretations induced by the logical language that is used:

$$\pi_K(\omega) = \min_{i=1,n} \max(\mu_{[ai]}(\omega), 1 - \alpha_i) \quad (1)$$

where $[a_i]$ denotes the sets of models of a_i so that $\mu_{[ai]}(\omega) = 1$ if $\omega \in [a_i]$ (i.e. $\omega \models a_i$), and $\mu_{[ai]}(\omega) = 0$ otherwise. The degree of possibility of ω according to (1) is computed as the complement to 1 of the largest weight of a formula falsified by ω. Thus, ω is all the less possible as it falsifies formulas of higher degrees. In particular, if ω is a counter-model of a formula with weight 1, then ω is impossible, i. e. $\pi_K(\omega) = 0$. Moreover $\pi_K(\omega)$ results from the min-based conjunctive combination of the elementary possibility distributions $\pi_{(ai,\ \alpha_i)}(\omega) = \max(\mu_{[ai]}(\omega), 1 - \alpha_i)$ that pertain to one formula (a_i, α_i). Note that $\pi_{(ai,\ \alpha_i)}(\omega) = 1$ if $\omega \models a_i$ and $\pi_{(a_i,\ \alpha_i)}(\omega) = 1 - \alpha_i$ otherwise, which means that $\Pi([a_i]) = 1$ and $\Pi([\neg pi]) = 1 - \alpha_i = 1 - N([a_i])$ and thus $N([a_i]) = \alpha_i$, having in mind that a possibility measure Π and a necessity measure N are associated with a possibility distribution π by the definition $\Pi(A) = \sup\{\pi(\omega) \mid \omega \in A\}$ and the duality $N(A) = 1 - \Pi(A^c)$ where A^c is the complement of A in Ω.

A principle of minimal specificity (which justifies the use of a min-combination) is at work in (1), since the greatest possible possibility degree is assessed to each ω in agreement with the constraints $N(a_i) \geq \alpha_i \Leftrightarrow \Pi(\neg a_i) \leq 1 - \alpha_i$. Note also that a state of complete ignorance about a is represented by $\pi_{[a]}(\omega) = 1$ if $\omega \models a$ and $\pi_{[\neg a]}(\omega) = 1$, or $N(a) = N(\neg a) = 0$ (the '[ ]' are now omitted).

It can be shown that π_K is the largest possibility distribution such that $N_K(a_i) \geq \alpha_i$, $\forall i = 1,n$, where N_K is the necessity measure associated with π_K, namely $N_K(a) = \min_{v \in [\neg a]}(1 - \pi_K(v))$). It may be that $N_K(a_i) > \alpha_i$, for some i, due to logical constraints linking formulas in K.

At the syntactic level, the inference rules are:

- $(\neg a \vee b, \alpha); (a, \beta) \vdash (b, \min(\alpha, \beta))$ (modus ponens)
- for $\beta \leq \alpha$ $(a, \alpha) \vdash (a, \beta)$ (weight weakening),

where $\vdash$ denotes the syntactic inference of possibilistic logic.

The min-decomposability of necessity measures allows us to work with weighted clauses without lack of generality, since $N(\wedge_{i=1,n}\ a_i) \geq \alpha \Leftrightarrow \forall i$, $N(a_i) \geq \alpha$. It means that in terms of possibilistic logic expressions we have $(\wedge_{i=1,n}\ a_i,\ \alpha) \Leftrightarrow$

$\wedge_{i=1,n}$ (a_i, α). In other words, any weighted logical formula put in Conjunctive Normal Form is equivalent to a set of weighted clauses. This feature considerably simplifies the proof theory of possibilistic logic. The basic inference rule in possibilistic logic put in clausal form is the resolution rule (cut):

$$(\neg a \vee b, \alpha); (a \vee c, \beta) \vdash (b \vee c, \min(\alpha, \beta))$$

Classical resolution is retrieved when all the weights are equal to 1. Other noticeable, valid inference rules are:

- if a entails b classically, $(a, \alpha) \vdash (b, \alpha)$ (formula weakening)
- $(a, \alpha); (a, \beta) \vdash (a, \max(\alpha, \beta))$ (weight fusion).

Observe that since $(\neg a \vee a, 1)$ is an axiom, formula weakening is a particular case of the resolution rule (indeed $(a, \alpha); (\neg a \vee a \vee b, 1) \vdash (a \vee b, \alpha)$). Formulas of the form (a, 0) which do not contain any information ($\forall a$, $N(a) \geq 0$ always holds), are not usually part of the possibilistic language since they bring nothing.

Refutation is easily extended to possibilistic logic. Let K be a knowledge base made of possibilistic formulas, i.e., $K = \{(a_i, \alpha_i)\}_{i=1,n}$. Proving (a, α) from K amounts to adding $(\neg a,1)$, put in clausal form, to K, and using the above rules repeatedly until getting $K \cup \{(\neg a, 1)\} \vdash (\perp, \alpha)$, where $\perp$ is the empty clause. Clearly, we are interested herein getting the empty clause with the greatest possible weight. It holds that $K \vdash (a, \alpha)$ if and only if $K_\alpha \vdash a$ (in the classical sense), where $K_\alpha = \{a: (a, \beta) \in K$ and $\beta \geq \alpha\}$. Possibilistic logic is sound and complete for refutation with respect to the above semantics where the semantic entailment corresponds to point wise fuzzy set inclusion ($K \models (a, \alpha)$ if and only if $\pi_K \leq \pi_{(a, \alpha)}$), Dubois *et al.*, 1994).

An important feature of possibilistic logic is its ability to deal with inconsistency. The level of inconsistency of a possibilistic logic base is defined as

$$inc(K) = \max\{\alpha \mid K \vdash (\perp, \alpha)\}$$

(by convention $\max\emptyset = 0$).

More generally, $inc(K) = 0$ if and only if $K^* = \{a_i \mid (a_i, \alpha_i) \in K)\}$ is consistent in the usual sense.

3.2 Level of awareness and disjunctions

In classical logic, we cannot make any difference between the two propositional bases corresponding to situations 1 and 2 below:

$$S1 = \{a, b\},$$

i.e. we are aware of 'a' and aware of 'b';

$$S2 = \{a, a \rightarrow b\},$$

i.e. we are aware of 'a' and aware of 'a→b'.

In both cases, we have the same deductive closure. This is due to the fact that from 'b' we can infer '$\neg a \vee b$' ($\equiv a \rightarrow b$). This points out that when we are aware of a formula, we should be aware as well of any disjunction involving this formula. However, S2 expresses a logical dependency between a and b, while this is not the case for S1.

More formally, let c(S) denote the closure of a set S of propositional formulas by iterated application of the cut rule. Then $c(S1) = S1 = \{a, b\}$ and $c(S2) = \{a, a \rightarrow b, b\}$. From this point of view, S1 and S2 are no longer equivalent, although they are semantically equivalent to the interpretation where a and b are true.

In possibilistic logic, this problem can be circumvented. Namely, $(\neg a \vee b, \alpha)$ is no longer subsumed by (b, β) if $\alpha > \beta$. Semantically speaking (a, β) means $N(a) \geq \beta$ where N is a necessity measure. Thus, the possibilistic bases

$$S'1 = \{(a, \beta), (b, \beta)\},$$
$$S'2 = \{(a, \beta), (a \rightarrow b, \alpha)\},$$

are associated with two different possibility distributions, respectively:

$$\pi_1(ab) = 1; \pi_1(\neg ab) = 1 - \beta;$$
$$\pi_1(a\neg b) = 1 - \beta; \pi_1(\neg a\neg b) = 1 - \beta,$$

$$\pi_2(ab) = 1; \pi_2(\neg ab) = 1 - \beta;$$
$$\pi_2(a\neg b) = 1 - \alpha; \pi_2(\neg a\neg b) = 1 - \beta.$$

Since $\alpha > \beta$, we have $\pi_1 > \pi_2$. Thus S'2 is better N-informed than S'1 (remember that total ignorance is represented by $\pi(\omega) = 1$ for all interpretations ω, and the minimal specificity principle expresses in a graded way that anything not stated as impossible is possible). The idea is that S'2 corresponds to a situation of greater awareness.

Formally, since $\pi_1 > \pi_2$, we may write

$$\pi_2 = \min(\pi_1, \pi) \qquad (2)$$

Thus S'2 is the combination of the information contained in S'1 and of an additional piece of information S. Let π^* be the largest solution of the above equation. When $\pi_1 > \pi_2$, π^* always exists and is unique. In our example, we have

$$\pi^*(ab) = 1 \; ; \; \pi^*(\neg ab) = 1;$$
$$\pi^*(a\neg b) = 1 - \alpha \; ; \; \pi^*(\neg a\neg b) = 1.$$

It corresponds to the possibilistic base

$$S' = \{(a \rightarrow b, \alpha)\}.$$

The syntactic counterpart of (2) in terms of bases, writes

$$S'2 = S'1 \cup S'.$$

Moreover in syntactic terms, the *closure* of S'2: $\{(a, \beta), (a \rightarrow b, \alpha)\}$ *by the cut rule* in possibilistic logic $((\neg a \vee b, \alpha), (a \vee c, \beta) \vdash (b \vee c, \min(\alpha, \beta))$, writes S'2: $\{(a, \beta), (a \rightarrow b, \alpha), (b, \beta)\}$. This differs from S'1: $\{(a, \beta), (b, \beta)\}$, where $a \rightarrow b$ cannot be obtained by cut from 'a' and 'b', but only as a weakening of b.

This suggests the following approach. Given a possibilistic logic base K with formulas having levels $1 = \alpha_1 > \alpha_2 > \ldots > \alpha_n > 0$, being aware of K at level α_j, means that we only access the formulas in K that are associated with weights equal to or smaller than α_j, or to formulas that can be deduced from those formulas by application of the cut rule. Thus, the weights are no longer viewed as certainty levels, but as increasing levels of awareness.

Let $K_{\geq\beta}$ (resp. $K_{>\beta}$, $K_{=\beta}$) be the set of formulas in K whose level is greater than or equal to (resp. strictly greater than, equal to) β. Namely,

$$K_{\geq\beta} = \{(a, \alpha) \text{ s. t. } (a, \alpha) \in K \text{ and } \alpha \geq \beta\};$$
$$K_{>\beta} = \{(a, \alpha) \text{ s. t. } (a, \alpha) \in K \text{ and } \alpha > \beta\};$$
$$K_{=\beta} = \{(a, \alpha) \text{ s. t. } (a, \alpha) \in K \text{ and } \alpha = \beta\}.$$

Then obviously $K_{\geq\beta} = K_{=\beta} \cup K_{>\beta}$.

This provides a decomposition of a knowledge base between the set of formulas $K_{=\beta}$ that an agent is aware of, say at level β, and the formulas in $K_{>\beta}$ that are at higher levels of awareness. Clearly, at the semantic level, we have

$$\pi_{K\geq\beta} = \min(\pi_{K=\beta}, \pi_{K>\beta}). \quad (3)$$

With this reading of a possibilistic base K, we have the following result, where (K)* and c(K) respectively denote the set of the formulas in K without their weights, and the closure of K by the possibilistic cut rule only, K being a possibilistic base. In the particular case where all the weights in K are equal to 1, the closure of K by the possibilistic cut rule and the closure of (K)* by the classical cut rule are equivalent. Moreover, note that c((K)*) = (c(K))*. In the following, $(c(K_{=\beta}))^*$ is thus abridged into $c(K_{=\beta})^*$, and similarly for $K_{>\beta}$ and for $K_{\geq\beta}$.

If a formula belongs to the deductive closure of $(K_{=\beta})^*$, but not to $c(K_{=\beta})^*$, while it belongs to $c(K_{\geq\beta})^*$, then this formula is in $c(K_{>\beta})^*$.

For instance, in the above example $\neg a \vee b$ is in the deductive closure of $(K_{=\beta})^* = \{a, b\}$, but not in $c(K_{=\beta})^*$, while it belongs to $c(K_{\geq\beta})^* = \{a, \neg a \vee b, b\}$, where $\neg a \vee b$ is in $c(K_{>\beta})^* = \{\neg a \vee b\}$.

In summary, in the above approach,

i) The formulas, for which the agent is supposed to be aware that they are true, are the formulas in $(K_{=\beta})^*$, and the formulas in $c(K_{=\beta})^*$ that can be obtained by the cut rule from $(K_{=\beta})^*$; note that the agent is not supposed to be aware of the formulas in the deductive closure of $(K_{=\beta})^*$ that are not in $c(K_{=\beta})^*$.

ii) The agent is supposed not to be aware of formulas in $(K_{>\beta})^*$ (provided they are not in $c(K_{=\beta})^*$);

iii) It would be possible to deal with several levels of (non)-awareness $K_{=\beta_1}, \ldots, K_{=\beta_k}$ with

$\beta_1 < \ldots < \beta_k$, such that the agent depending his role may access a different level i of awareness, i. e., the agent is aware of formulas $K_{=\beta_1} \cup \ldots \cup K_{=\beta_i}$, but not in $K_{=\beta_{i+1}} \cup \ldots \cup K_{=\beta_k}$.

iv) The fact that a formula is in $c(K_{\geq\beta})$ with level β does not necessarily means that the agent is aware of it, if it can be only inferred using higher level formulas (which can be easily detected). For instance, if $(K_{=\beta})^* = \{a\}$, $(K_{>\beta})^* = \{\neg a \vee b\}$, then $c(K_{\geq\beta})^* \supset \{b\}$.

v) The approach may be simplified if the agent is supposed to be aware of all the formulas in the deductive closure of $K_{=\beta}$, (but not of the formulas in $K_{>\beta}$). Then the standard possibilistic logic inference mechanism based on refutation can be used, rather than using the limited closure based on the cut rule.

4 Awareness of inability to know

As recalled in section 3.1, the usual possibilistic logic handles constraints of the form $N(a) \geq \alpha$ Constraints of the form $\Pi(a) \geq \alpha$ can be also handled. They represent poor pieces of information, while $N(a) \geq \alpha \Leftrightarrow \Pi(\neg a) \leq 1 - \alpha$ expresses partial certainty of a and impossibility of $\neg a$.

However, the inability to know if a is true or false can be expressed by $\Pi(a) = 1 = \Pi(\neg a)$, which states that both a and $\neg a$ are fully possible. In that view, the unawareness of a and the unawareness of $\neg a$ are the same thing.

The following cut rule, which mixes the two types of lower bound constraints on Π and N, has been established (Dubois and Prade, 1990):

$$N(\neg a \vee b) \geq \alpha;\ \Pi(a \vee c) \geq \beta \ |\!-\ \Pi(b \vee c) \geq \alpha \,\&\, \beta$$

with $\alpha \,\&\, \beta = 0$ if $\alpha + \beta \leq 1$; $\alpha \,\&\, \beta = \beta$ if $\alpha + \beta > 1$.

As a particular case, the following rule holds for $\beta = 1$, namely

$$N(\neg a \vee b) > 0;\ \Pi(a \vee c) = 1 \ |\!-\ \Pi(b \vee c) = 1 \qquad (4)$$

This is easy to check since $N(\neg a \vee b) > 0$ is equivalent to $\Pi(a \wedge \neg b) < 1$. Then $\Pi(a \vee c) = \Pi((a \wedge \neg b) \vee (a \wedge b) \vee c) = \max(\Pi(a \wedge \neg b), \Pi(a \wedge b), \Pi(c))$ applying the max-decomposability of Π. Hence $\max(\Pi(a \wedge b), \Pi(c)) = 1 \leq \max(\Pi(b), \Pi(c)) = \Pi(b \vee c)$.

As a consequence of (4), if the awareness of a is equivalent to the one of b, i.e. we have $N(\neg a \vee b) > 0$ and $N(\neg b \vee a) > 0$, and if the agent is unable to know a, i.e. $\Pi(a) = 1 = \Pi(\neg a)$, then he is also unable to know b, i.e., $\Pi(b) = 1 = \Pi(\neg b)$, as expected.

Thus, the inability to know can be represented and propagated in the possibilistic framework. It can be conjointly handled with the approach of section 3.2, since a possibilistic logic with the two types of bounds has been developed (Dubois et al., 1994).

5 Limited awareness of conjunctions

With the approach outlined in section 3.2, the agent may not be aware of $a \wedge b$, while he is aware of a and he is aware of b. But we may like to have the agent aware of 'a' and 'b', without being aware of '$a \wedge b$'. Since $N(a \wedge b) = \min(N(a), N(b))$, $(a \wedge b, \alpha)$ is semantically equivalent to $\{(a, \alpha); (b, \alpha)\}$. Thus the above approach cannot be applied.

However, this is achievable by using Δ–based formulas rather than N-based formulas as above. Indeed a measure denoted Δ, and called "guaranteed possibility, has been introduced in possibility theory (e.g., (Dubois and Prade, 2004)).

This measure Δ is associated with a distribution δ in the following way:

$$\Delta(a) = \min\{\delta(\omega) \mid \omega \models a\}.$$

Thus, $\Delta(a \vee b) = \min(\Delta(a), \Delta(b))$. $\Delta(a)$ corresponds to the minimal level of possibility of a model of a. It is thus a guaranteed level of possibility, and is the basis for a logic of observations (Dubois *et al.*, 2000).

Then a weighted formula (now written between brackets) $[a, \gamma]$ is understood as $\Delta(a) \geq \gamma$. The associated cut rule is now:

$$[\neg a \wedge b, \gamma], [a \wedge c, \eta] \ |\!-\ [b \wedge c, \min(\gamma, \eta)].$$

Mind it works in a reverse way w. r. t. classical entailment. Total lack of Δ-information is represented by $\delta(\omega) = 0$ for all interpretations ω, and a maximal specificity principle now applies expressing in a graded way that anything not stated

as possible is impossible (closed world assumption).

Indeed, T'1 = {[a, γ]; [b, γ]} is now represented by

$$\delta_1(ab) = \gamma;\ \delta_1(\neg ab) = \gamma;$$
$$\delta_1(a\neg b) = \gamma\ ;\ \delta_1(\neg a\neg b) = 0$$

and T'2 = {[a, γ], [b, γ], [a ∧ b, η]} with γ < η, is associated with the distribution

$$\delta_2(ab) = \eta;\ \delta_2(\neg ab) = \gamma;$$
$$\delta_2(a\neg b) = \gamma;\ \delta_2(\neg a\neg b) = 0.$$

while T' = {[a ∧ b, η]} is associated with the distribution with

$$\delta(ab) = \eta;\ \delta(\neg ab) = 0;$$
$$\delta(a\neg b) = 0;\ \delta(\neg a\neg b) = 0.$$

Clearly, it can be checked that the following decomposition holds

$$\delta_2 = \max(\delta_1, \delta) \qquad (5)$$

and δ_2 ($>\delta_1$) is better Δ-informed than δ_1.

Thus in the Δ–possibilistic base {[a, γ], [b, γ], [a ∧ b, η]} with γ < η, we are aware of 'a ∧ b', while this is not the case in {[a, γ], [b, γ]}.

Then an approach similar to the one of section 3.2 could be developed. It should be manageable to combine the two approaches, using *two different scales* separately, and thus to be aware of 'a' and 'b' without being aware of 'a ∧ b' and 'a → b' for instance.

6 Concluding remarks

This research note has outlined potentialities of the possibilistic framework for handling (un)awareness. Some other lines of research can be mentioned.

First, it would be possible to combine the above approach with beliefs of different levels. Namely the agent would be aware of propositions with some certainty levels, but might be not aware that the same propositions can be regarded in fact as being more certain (or even fully certain).

Another road that might be worth investigating, would be to use an extension of possibilistic logic with ill-known certainty levels. Then a formula that the agent is not aware of could receive an unknown level.

In Halpern (2001)'s approach to different forms of unawareness, 'being aware' is viewed as a modal operator distinct from 'knowing'. In the approaches outlined here in Sections 3 and 5, a similar distinction does not exist. Rather, the distinction is made through the introduction of a level of awareness that can be controlled through the possibilistic inference machinery. This contrasts with section 4, where the inability to know *a* (or ¬ *a*) may be viewed as the counterpart of a specific modal information. A comparison of the notions of (un)awareness captured in Halpern (2001)'s approach and the ones discussed in this paper is a topic for further research.

Clearly, (un)awareness is still more interesting, especially from an application point of view in a multiple-agent setting (Heifetz *et al.* , 2003), where, e. g., an agent may be unaware of something that another agent is aware of; moreover this other agent may be also aware that the first agent is not aware of the thing. Modeling (un)awareness may be a crucial issue in negotiation. Then the possibilistic handling of (un)awareness would require first to work with a multiple agent extension of possibilistic logic. Such an extension is currently under study.

Lastly, there are also dynamic aspects in unawareness. For instance, believing that one can't be aware of *a*, one may receive the information that *a*, that forces the agent to reconsider the unawareness status of some propositions. Believing *a*, one may also receive the information that one cannot be aware of *a*. This raises new revision problems.

References

D. Dubois, P. Hajek, H. Prade (2000) Knowledge-driven versus data-driven logics. *Journal of Logic, Language, and Information*, 9, 65-89.

D. Dubois, H. Prade Resolution principles in possibilistic logic, *Int. J. of Approximate Reasoning*, 4, 1-21, 1990.

D. Dubois, J. Lang and H. Prade (1994), Possibilistic logic. In: *Handbook of Logic in Artificial Intelligence and Logic Programming*, (D.M. Gabbay et al., eds.), Vol. 3, Oxford Univ. Press, Oxford, UK, 439-513.

D. Dubois, H. Prade (2004) Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144, 3-23.

R. Fagin, J. Y. Halpern (1988) Belief, awareness, and limited reasoning, *Artificial Intelligence*, 34, 39-76.

J. Y. Halpern (2001) Alternative semantics for unawareness, *Games and Economic Behavior*, 37, 321-339.

A. Heifetz, M. Meier, B. C. Schipper (2003) Multi-person unawareness. *Proc. of the 9th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2003),* J. Y. Halpern, M. Tennenholtz (eds.), Bloomington, Indiana, USA, June 20-22, 2003, 145-158.

S. Modica and A. Rustichini (1994) Awareness and partitional information structures, *Theory and Decision*, 37, 107-125.

S. Modica and A. Rustichini (1999) Unawareness and partitional information structures, *Games and Economic Behavior*, 27, 265-298.

2.13 On the Computation of Warranted Arguments within a Possibilistic Logic Framework with Fuzzy Unification

On the Computation of Warranted Arguments within a Possibilistic Logic Framework with Fuzzy Unification *

Teresa Alsinet Dept. of Computer Science, University of Lleida, Lleida, SPAIN, {tracy,cic}@eps.udl.es
Carlos Chesñevar Dept. of Computer Science, University of Lleida, Lleida, SPAIN, {tracy,cic}@eps.udl.es
Lluís Godo AI Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, SPAIN, {godo,sandri}@iiia.csic.es
Sandra Sandri AI Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, SPAIN, {godo,sandri}@iiia.csic.es
Guillermo Simari Dept. of Computer Science and Eng., Universidad Nacional del Sur, Bahía Blanca, ARGENTINA, grs@cs.uns.edu.ar

Abstract

Possibilistic Defeasible Logic Programming (P-DeLP) is a logic programming language which combines features from argumentation theory and logic programming, incorporating the treatment of possibilistic uncertainty at object-language level. The aim of this paper is twofold: first to present an approach towards extending P-DeLP in order to incorporate fuzzy constants and fuzzy unification, and after to propose a way to handle conflicting arguments in the context of the extended framework.

Keywords: Possibilistic logic, fuzzy constants, fuzzy unification, defeasible argumentation.

Introduction

In the last decade, defeasible argumentation has emerged as a very powerful paradigm to model commonsense reasoning in the presence of incomplete and potentially inconsistent information (Chesñevar, Maguitman, & Loui 2000). Recent developments have been oriented towards integrating argumentation as part of logic programming languages. In this context, Possibilistic Defeasible Logic Programming (P-DeLP) (Chesñevar *et al.* 2004) is a logic programming language which combines features from argumentation theory and logic programming, incorporating the treatment of possibilistic uncertainty at object-language level. Roughly speaking, in P-DeLP degrees of uncertainty help in determining which arguments prevail in case of conflict.

In spite of its expressive power, an important limitation in P-DeLP (as defined in (Chesñevar *et al.* 2004)) is that the treatment of imprecise, fuzzy information was not formalized. One interesting alternative for such formalization is the use of PGL$^+$, a Possibilistic logic over Gödel logic extended with fuzzy constants. Fuzzy constants in PGL$^+$ allow expressing imprecise information about the possibly unknown value of a variable (in the sense of magnitude) modeled as a (unary) predicate. For instance, an imprecise statement like "John's salary is low" can be expressed PGL$^+$ by the formula $John\_salary(low)$ where $John\_salary$ is a predicate and low a fuzzy constant, which will be mapped under a given PGL$^+$ interpretation to a fuzzy set rather to a single domain element as usually in predicate logics. Notice that this kind of statements express *disjunctive* knowledge (mutually exclusive), in the sense that in each interpretation it is natural to require that the predicate $John\_salary(x)$ be true for one and only one variable assignment to x, say u_0. Then, in such an interpretation it is also natural to evaluate to what extent $John\_salary(low)$ is true as the degree in which the salary u_0 is considered to be low. Hence, allowing fuzzy constants in the language leads to treat formulas in a many-valued logical setting (that of Gödel many-valued logic in our framework), as opposed to the bivalued setting within classical possibilistic logic, with the unit interval $[0, 1]$ as a set of truth-values.

The aim of this paper is twofold: first to define DePGL$^+$, a possibilistic defeasible logic programming language that extends P-DeLP through the use of PGL$^+$, instead of (classical) possibilistic logic, in order to incorporate fuzzy constants and fuzzy unification, and after to propose a way to handle conflicting arguments in the context of the extended framework. To this end, the rest of the paper is structured as follows. First, we present the fundamentals of PGL$^+$. Then we define the DePGL$^+$ programming language. The next two sections focus on the characterization of arguments in DePGL$^+$ and the analysis of the notion of conflict among arguments in the context of our proposal. Next we discuss some problematic situations that may arise when trying to define the notion of warranted arguments in DePGL$^+$, and propose some solutions. Finally we discuss some related work and present the main conclusions we have obtained.

PGL$^+$: Overview

Possibilistic logic (Dubois, Lang, & Prade 1994) is a logic of uncertainty where a certainty degree between 0 and 1, interpreted as a lower bound of a necessity measure, is attached to each classical formula. In the propositional version, possibilistic formulas are pairs (φ, α) where φ is a proposition of classical logic and interpreted as specifying a constraint $N(\varphi) \geq \alpha$ on the necessity measure of φ. Possibilistic models are possibility distributions $\pi : \Omega \rightarrow [0, 1]$ on the set of classical (bivalued) interpretations Ω which rank them in terms of plausibility: w is at least as plausible as w' when $\pi(w) \geq \pi(w')$. If $\pi(w) = 1$ then w is considered as fully plausible, while if $\pi(w) = 0$ w is

*This is a proper extension of the paper "Modeling Defeasible Argumentation within a Possibilistic Logic Framework with Fuzzy Unification" to appear in the 11th IPMU International Conference 2006 (Paris, France).

considered as totally impossible. Then (φ, α) is satisfied by π, written $\pi \models (\varphi, \alpha)$ whenever $N_\pi(\varphi) \geq \alpha$, where $N_\pi(\varphi) = \inf\{1 - \pi(w) \mid w(\varphi) = 0\}$.

In (Alsinet & Godo 2000; 2001) the authors introduce PGL$^+$, an extension of possibilistic logic allowing to deal with some form of fuzzy knowledge and with an efficient and complete proof procedure for atomic deduction when clauses fulfill two kinds of constraints. Technically speaking, PGL$^+$ is a possibilistic logic defined on top of (a fragment of) Gödel infinitely-valued logic, allowing uncertainty qualification of predicates with imprecise, fuzzy constants, and allowing as well a form of graded unification between them. Next we provide some details.

The *basic components* of PGL$^+$ formulas are: a set of primitive propositions (fuzzy propositional variables) *Var*; a set S of *sorts* of constants; a set $\mathcal{C}$ of object *constants*, each having its sort; a set *Pred* of unary regular predicates, each one having a *type*; and *connectives* $\wedge$, $\rightarrow$. An *atomic formula* is either a primitive proposition from *Var* or of the form $p(A)$, where p is a predicate symbol from *Pred*, A is an object constant from $\mathcal{C}$ and the sort of A corresponds to the type of p. *Formulas* are Horn-rules of the form $p_1 \wedge \cdots \wedge p_k \rightarrow q$ with $k \geq 0$, where $p_1, \ldots, p_k, q$ are atomic formulas. A (weighted) *clause* is a pair of the form (φ, α), where φ is a Horn-rule and $\alpha \in [0, 1]$.

Remark *Since variables, quantifiers and function symbols are not allowed, the language of PGL$^+$ so defined remains in fact propositional. This allows us to consider only unary predicates since statements involving multiple (fuzzy) properties can be always represented in PGL$^+$ as a conjunction of atomic formulas. For instance, the statement "Mary is young and tall" can be represented in PGL$^+$ as* $age\_Mary(young) \wedge height\_Mary(tall)$ *instead of using a binary predicate involving two fuzzy constants like* $age\&height\_Mary(young, tall)$.

A many-valued *interpretation* for the language is a structure $w = (U, i, m)$, where: $U = \cup_{\sigma \in \mathcal{S}} U_\sigma$ is a collection of non-empty domains U_σ, one for each basic sort $\sigma \in \mathcal{S}$; $i = (i_{prop}, i_{pred})$, where $i_{prop} : Var \rightarrow [0, 1]$ maps each primitive proposition q into a value $i_{prop}(q) \in [0, 1]$ and $i_{pred} : Pred \rightarrow U$ maps a predicate p of type (σ) into a value $i_{pred}(p) \in U_\sigma$; and $m : \mathcal{C} \rightarrow [0, 1]^U$ maps an object constant A of sort σ into a normalized fuzzy set $m(A)$ on U_σ, with membership function $\mu_{m(A)} : U_\sigma \rightarrow [0, 1]$. [1]

The *truth value* of an atomic formula φ under an interpretation $w = (U, i, m)$, denoted by $w(\varphi) \in [0, 1]$, is defined as $w(q) = i_{prop}(q)$ for primitive propositions, and $w(p(A)) = \mu_{m(A)}(i_{pred}(p))$ for atomic predicates. The truth evaluation is extended to rules by means of interpreting the $\wedge$ connective by the min-conjunction and the $\rightarrow$ connective by the so-called Gödel's many-valued implication: $w(p_1 \wedge \cdots \wedge p_k \rightarrow q) = 1$ if $\min(w(p_1), \ldots, w(p_k)) \leq w(q)$, and $w(p_1 \wedge \cdots \wedge p_k \rightarrow q) = w(q)$ otherwise.

Note that the truth value $w(\varphi)$ will depend not only on the interpretation i_{pred} of predicate symbols that φ may contain, but also on the fuzzy sets assigned to fuzzy constants by m. Then, in order to define the possibilistic semantics, we need to fix a meaning for the fuzzy constants and to consider some extension of the standard notion of necessity measure for fuzzy events. The first is achieved by fixing a *context*. Basically a context is the set of interpretations sharing a common domain U and an interpretation of object constants m. So, given U and m, its associated context is just the set of interpretations $\mathcal{I}_{U,m} = \{w \mid w = (U, i, m)\}$ and, once fixed the context, $[\varphi]$ denotes the fuzzy set of models for a formula φ defining $\mu_{[\varphi]}(w) = w(\varphi)$, for all $w \in \mathcal{I}_{U,m}$.

Now, in a fixed context $\mathcal{I}_{U,m}$, a belief state (or *possibilistic model*) is determined by a normalized possibility distribution on $\mathcal{I}_{U,m}$, $\pi : \mathcal{I}_{U,m} \rightarrow [0, 1]$. Then, we say that π *satisfies* a clause (φ, α), written $\pi \models (\varphi, \alpha)$, iff the (suitable) necessity measure of the fuzzy set of models of φ with respect to π, denoted $N([\varphi] \mid \pi)$, is indeed at least α. Here, for the sake of soundness preservation, we take

$$N([\varphi] \mid \pi) = \inf_{w \in \mathcal{I}_{U,m}} \pi(w) \Rightarrow \mu_{[\varphi]}(w),$$

where $\Rightarrow$ is the reciprocal of Gödel's many-valued implication, defined as $x \Rightarrow y = 1$ if $x \leq y$ and $x \Rightarrow y = 1 - x$, otherwise. This necessity measure for fuzzy sets was proposed and discussed by Dubois and Prade (cf. (Dubois, Lang, & Prade 1994)). For example, according to this semantics, given a context $\mathcal{I}_{U,m}$ the formula

$$(age\_Peter(about\_35), 0.9)$$

is to be interpreted in PGL$^+$ as the following set of clauses with imprecise but non-fuzzy constants

$$\{(age\_Peter([about\_35]_\beta), \min(0.9, 1 - \beta)) : \beta \in [0, 1]\},$$

where $[about\_35]_\beta$ denotes the β-cut of the fuzzy set $m(about\_35)$. As usual, a set of clauses P is said to *entail* another clause (φ, α), written $P \models (\varphi, \alpha)$, iff every possibilistic model π satisfying all the clauses in P also satisfies (φ, α), and we say that a set of clauses P is *satisfiable* in the context determined by U and m if there exists a normalized possibility distribution $\pi : \mathcal{I}_{U,m} \rightarrow [0, 1]$ that satisfies all the clauses in P. Satisfiable clauses enjoy the following result (Alsinet 2003): If P is satisfiable and $P \models (\varphi, \alpha)$, with $\alpha > 0$, there exists at least an interpretation $w \in \mathcal{I}_{U,m}$ such that $w(\varphi) = 1$.

Finally, still in a context $\mathcal{I}_{U,m}$, the *degree of possibilistic entailment* of an atomic formula (or goal) φ by a set of clauses P, denoted by $\|\varphi\|_P$, is the greatest $\alpha \in [0, 1]$ such that $P \models (\varphi, \alpha)$. In (Alsinet 2003), it is proved that $\|\varphi\|_P = \inf\{N([\varphi] \mid \pi) \mid \pi \models P\}$.

The calculus for PGL$^+$ in a given context $\mathcal{I}_{U,m}$ is defined by the following set of inference rules:

Generalized resolution:

$$\frac{(p \wedge s \rightarrow q(A), \alpha), \quad (q(B) \wedge t \rightarrow r, \beta)}{(p \wedge s \wedge t \rightarrow r, \min(\alpha, \beta))} \text{ [GR], if } A \subseteq B$$

Fusion:

$$\frac{(p(A) \wedge s \rightarrow q(D), \alpha), \quad (p(B) \wedge t \rightarrow q(E), \beta)}{(p(A \cup B) \wedge s \wedge t \rightarrow q(D \cup E), \min(\alpha, \beta))} \text{ [FU]}$$

[1] Note that for each predicate symbol p, $i_{pred}(p)$ is the one and only value of the domain which satisfies p in that interpretation and that m prescribes for each constant A at least one value u_0 of the domain U_σ as fully compatible, i.e. such that $\mu_{m(A)}(u_0) = 1$.

Intersection:

$$\frac{(p(A),\alpha),(p(B),\beta)}{(p(A\cap B),\min(\alpha,\beta))}\ [\text{IN}]$$

Resolving uncertainty:

$$\frac{(p(A),\alpha)}{(p(A'),1)}\ [\text{UN}],\ \text{where } A'=\max(1-\alpha,A)$$

Semantical unification:

$$\frac{(p(A),\alpha)}{(p(B),\min(\alpha,N(B\mid A)))}\ [\text{SU}]$$

For each context $\mathcal{I}_{U,m}$, the above GR, FU, SU, IN and UN inference rules can be proved to be *sound* with respect to the possibilistic entailment of clauses. Moreover we shall also refer to the following weighted **modus ponens** rule, which can be seen as a particular case of the GR rule

$$\frac{(p_1\wedge ...\wedge p_n\rightarrow q,\alpha),\ (p_1,\beta_1),\ldots,(p_n,\beta_n)}{(q,\min(\alpha,\beta_1,\ldots,\beta_n))}\ [\text{MP}]$$

The notion of *proof* in PGL$^+$, denoted by $\vdash$, is that of deduction by means of the triviality axiom and the PGL$^+$ inference rules. Given a context $\mathcal{I}_{U,m}$, the *degree of deduction* of a goal φ from a set of clauses P, denoted $|\varphi|_P$, is the greatest $\alpha\in[0,1]$ for which $P\vdash(\varphi,\alpha)$. Actually this notion of proof is complete for determining the degree of possibilistic entailment of a goal, i.e. $|\varphi|_P=\|\varphi\|_P$, for non-recursive and satisfiable programs P, called PGL$^+$ programs, under certain further conditions. Details can be found in (Alsinet & Godo 2001; Alsinet 2003).

The DePGL$^+$ programming language

As already pointed out our objective is to extend the P-DeLP programming language through the use of PGL$^+$ in order to incorporate fuzzy constants and fuzzy propositional variables; we will refer to this extension as *Defeasible* PGL$^+$, DePGL$^+$ for short. To this end, the base language of P-DeLP (Chesñevar *et al.* 2004) will be extended with fuzzy constants and fuzzy propositional variables, and arguments will have an attached necessity measure associated with the supported conclusion.

The DePGL$^+$ language $\mathcal{L}$ is defined over PGL$^+$ atomic formulas together with the connectives $\{\sim,\wedge,\leftarrow\}$. The symbol $\sim$ stands for *negation*. A *literal* $L\in\mathcal{L}$ is a PGL$^+$ atomic formula or its negation. A *rule* in $\mathcal{L}$ is a formula of the form $Q\leftarrow L_1\wedge\ldots\wedge L_n$, where $Q,L_1,\ldots,L_n$ are literals in $\mathcal{L}$. When $n=0$, the formula $Q\leftarrow$ is called a *fact* and simply written as Q. In the following, capital and lower case letters will denote literals and atoms in $\mathcal{L}$, respectively.

In argumentation frameworks, the negation connective allows to represent conflicts among pieces of information. In the frame of DePGL$^+$, the handling of negation deserves some explanation. In what regards negated propositional variables $\sim p$, the negation connective $\sim$ will not be considered as a proper Gödel negation. Rather, $\sim p$ will be treated as another propositional variable p', with a particular status with respect to p, since it will be only used to detect contradictions at the syntactical level. On the other hand, negated literals of the form $\sim p(A)$, where A is a fuzzy constant, will be handled in the following way.

As previously mentioned, fuzzy constants are *disjunctively* interpreted in PGL$^+$. For instance, consider the formula $speed(low)$. In each interpretation $I=(U,i,m)$, the predicate $speed$ is assigned a *unique* element $i(speed)$ of the corresponding domain. If low denotes a crisp interval of rpm's, say $[0,2000]$, then $speed(low)$ will be true iff such element belongs to this interval, i.e. iff $i(speed)\in[0,2000]$. Now, since the negated formula $\sim speed(low)$ is to be interpreted as "$\neg[\exists x\in low$ such that the engine speed is x]", which (under PGL$^+$ interpretations) amounts to "$[\exists x\notin low$ such that the engine speed is x]", it turns out that $\sim speed(low)$ is true iff $speed(\neg low)$ is true, where $\neg low$ denotes the complement of the interval $[0,2000]$ in the corresponding domain. Then, given a context $\mathcal{I}_{U,m}$, this leads us to understand a negated literal $\sim p(A)$ as another positive literal $p(\neg A)$, where the fuzzy constant $\neg A$ denotes the (fuzzy) complement of A, that is, where $\mu_{m(\neg A)}(u)=n(\mu_{m(A)}(u))$, for some suitable negation function n (usually $n(x)=1-x$).

Therefore, given a context $\mathcal{I}_{U,m}$, using the above interpretations of the negation, and interpreting the DePGL$^+$ arrow $\leftarrow$ as the PGL$^+$ implication $\rightarrow$, we can actually transform a DePGL$^+$ program P into a PGL$^+$ program, denoted as $\tau(P)$, and then, we can apply the deduction machinery of PGL$^+$ on $\tau(P)$ for automated proof purposes. From now on and for the sake of a simpler notation, we shall write $\Gamma\vdash_\tau(\varphi,\alpha)$ to denote $\tau(\Gamma)\vdash\tau((\varphi,\alpha))$, being Γ and (φ,α) DePGL$^+$ clauses. Moreover, we shall consider that the negation function n is implicitly determined by each context $\mathcal{I}_{U,m}$, i.e. the function m will interpret both fuzzy constants A and their complement (negation) $\neg A$.

Arguments in DePGL$^+$

In the last sections we formalized the many-valued and the possibilistic semantics of the underlying logic of DePGL$^+$. In this section we formalize the procedural mechanism for building arguments in DePGL$^+$.

We distinguish between *certain* and *uncertain* DePGL$^+$ clauses. A DePGL$^+$ clause (φ,α) will be referred as certain when $\alpha=1$ and uncertain, otherwise. Given a context $\mathcal{I}_{U,m}$, a set of DePGL$^+$ clauses Γ will be deemed as *contradictory*, denoted $\Gamma\vdash_\tau\bot$, when

(i) either $\Gamma\vdash_\tau(q,\alpha)$ and $\Gamma\vdash_\tau(\sim q,\beta)$, with $\alpha>0$ and $\beta>0$, for some atom q in $\mathcal{L}$,

(ii) or $\Gamma\vdash_\tau(p(A),\alpha)$ with $\alpha>0$, for some predicate p and some fuzzy constant A such that $m(A)$ is non-normalized.

Notice that in the latter case, $\tau(\Gamma)$ is not satisfiable and there exist $\Gamma_1\subset\tau(\Gamma)$ and $\Gamma_2\subset\tau(\Gamma)$ such that Γ_1 and Γ_2 are satisfiable and $|p(B)|_{\Gamma_1}>0$ and $|p(C)|_{\Gamma_2}>0$, with $A=B\cap C$.

Example 1 *Consider the set of clauses* $\Gamma=\{(q,0.8),(r,1),(p(A)\leftarrow q,0.5),(p(B)\leftarrow q\wedge r,0.3)\}$. *Then,* $\Gamma\vdash_\tau$

$(p(A), 0.5)$ *and* $\Gamma \vdash_\tau (p(B), 0.3)$*, and, by the IN inference rule,* $\Gamma \vdash_\tau (p(A \cap B), 0.3)$*. Hence, in a particular context* $\mathcal{I}_{U,m}$*,* Γ *is contradictory as soon as* $m(A) \cap m(B)$ *is a non-normalized fuzzy set whereas, for instance,* $\Gamma \backslash \{(r, 1)\}$ *is satisfiable.*

A DePGL$^+$ program is a set of clauses in $\mathcal{L}$ in which we distinguish certain from uncertain information. As additional requirement, certain knowledge is required to be non-contradictory and the corresponding PGL$^+$ program (by means of transformation τ) is required to satisfy the modularity constraint (Alsinet & Godo 2001; Alsinet 2003). Formally: Given a context $\mathcal{I}_{U,m}$, a *DePGL$^+$ program* $\mathcal{P}$ is a pair (Π, Δ), where Π is a non-contradictory finite set of certain clauses, Δ is a finite set of uncertain clauses, and $\tau(\Pi \cup \Delta)$ satisfies the modularity constraint.

The requirement of the modularity constraint of a DePGL$^+$ program ensures that all (explicit and hidden) clauses of programs are considered. Indeed, since fuzzy constants are interpreted as (flexible) restrictions on an existential quantifier, atomic formulas clearly express disjunctive information. For instance, when $A = \{a_1, \ldots, a_n\}$, $p(A)$ is equivalent to the disjunction $p(a_1) \vee \cdots \vee p(a_n)$. Then, when parts of this (hidden) disjunctive information occur in the body of several program formulas we also have to consider all those new formulas that can be obtained through a completion process of the program which is based on the RE and FU inference rules.

Example 2 *(Adapted from (Chesñevar* et al. *2004)) Consider an intelligent agent controlling an engine with three switches* $sw1$*,* $sw2$ *and* $sw3$*. These switches regulate different features of the engine, such as pumping system, speed, etc. The agent's generic (and incomplete) knowledge about how this engine works is the following:*

- *If the pump is clogged, then the engine gets no fuel.*
- *When* $sw1$ *is on, apparently fuel is pumped properly.*
- *When fuel is pumped, fuel seems to work ok.*
- *When* $sw2$ *is on, usually oil is pumped.*
- *When oil is pumped, usually it works ok.*
- *When there is oil and fuel, normally the engine is ok.*
- *When there is heat, the engine is almost sure not ok.*
- *When there is heat, normally there are oil problems.*
- *When fuel is pumped and speed is low, there are reasons to believe that the pump is clogged.*
- *When* $sw2$ *is on, usually speed is low.*
- *When* $sw2$ *and* $sw3$ *are on, usually speed is not low.*
- *When* $sw3$ *is on, normally fuel is ok.*

Suppose also that the agent knows some particular facts about the current state of the engine:

- $sw1$*,* $sw2$ *and* $sw3$ *are on, and*
- *the temperature is around 31^{o}C.*

This knowledge can be modelled by the program $\mathcal{P}_{engine}$ *shown in Fig. 1. Note that uncertainty is assessed in terms of different necessity degrees and vague knowledge is represented by means of fuzzy constants (*low*,* $around\_31$*,* $high$*).*

Next we introduce the notion of *argument* in DePGL$^+$. Informally, an argument for a literal (goal) Q with necessity degree α is a tentative (as it relies to some extent on uncertain, possibilistic information) proof for (Q, α) .

(1)	$(\sim fuel\_ok \leftarrow pump\_clog, 1)$
(2)	$(pump\_fuel \leftarrow sw1, 0.6)$
(3)	$(fuel\_ok \leftarrow pump\_fuel, 0.85)$
(4)	$(pump\_oil \leftarrow sw2, 0.8)$
(5)	$(oil\_ok \leftarrow pump\_oil, 0.8)$
(6)	$(engine\_ok \leftarrow fuel\_ok \wedge oil\_ok, 0.6)$
(7)	$(\sim engine\_ok \leftarrow temp(high), 0.95)$
(8)	$(\sim oil\_ok \leftarrow temp(high), 0.9)$
(9)	$(pump\_clog \leftarrow pump\_fuel \wedge speed(low), 0.7)$
(10)	$(speed(low) \leftarrow sw2, 0.8)$
(11)	$(\sim speed(low) \leftarrow sw2, sw3, 0.8)$
(12)	$(fuel\_ok \leftarrow sw3, 0.9)$
(13)	$(sw1, 1)$
(14)	$(sw2, 1)$
(15)	$(sw3, 1)$
(16)	$(temp(around\_31), 0.85)$

Figure 1: DePGL$^+$ program $\mathcal{P}_{eng}$ (example 2)

Definition 3 (Argument) *Given a context* $\mathcal{I}_{U,m}$ *and a DePGL$^+$ program* $\mathcal{P} = (\Pi, \Delta)$*, a set* $\mathcal{A} \subseteq \Delta$ *of uncertain clauses is an* argument *for a goal* Q *with necessity degree* $\alpha > 0$*, denoted* $\langle \mathcal{A}, Q, \alpha \rangle$*, iff:*
(1) $\Pi \cup \mathcal{A} \vdash_\tau (Q, \alpha)$*;*
(2) $\Pi \cup \mathcal{A}$ *is non contradictory; and*
(3) $\mathcal{A}$ *is minimal wrt set inclusion, i.e. there is no* $\mathcal{A}_1 \subset \mathcal{A}$ *satisfying (1) and (2).*

Let $\langle \mathcal{A}, Q, \alpha \rangle$ and $\langle \mathcal{S}, R, \beta \rangle$ be two arguments. We will say that $\langle \mathcal{S}, R, \beta \rangle$ is a *subargument* of $\langle \mathcal{A}, Q, \alpha \rangle$ iff $\mathcal{S} \subseteq \mathcal{A}$. Notice that the goal R may be a subgoal associated with the goal Q in the argument $\mathcal{A}$.

Given a context $\mathcal{I}_{U,m}$, the set of arguments for a DePGL$^+$ program $\mathcal{P} = (\Pi, \Delta)$ can be found by the iterative application of the following construction rules:

1) Building arguments from facts (INTF):

$$\frac{(Q, 1)}{\langle \emptyset, Q, 1 \rangle} \qquad \frac{(Q, \alpha), \Pi \cup \{(Q, \alpha)\} \nvdash_\tau \bot, \alpha < 1}{\langle \{(Q, \alpha)\}, Q, \alpha \rangle}$$

for any $(Q, 1) \in \Pi$ and any $(Q, \alpha) \in \Delta$.

2) Building Arguments by SU (SUA):

$$\frac{\langle \mathcal{A}, p(A), \alpha \rangle}{\langle \mathcal{A}, p(B), \min(\alpha, N(m(B) \mid m(A))) \rangle}$$

if $N(m(B) \mid m(A)) \neq 0$.

3) Building Arguments by UN (UNA):

$$\frac{\langle \mathcal{A}, p(A), \alpha \rangle}{\langle \mathcal{A}, p(A'), 1 \rangle}$$

where $m(A') = \max(1 - \alpha, m(A))$.

4) Building Arguments by IN (INA):

$$\frac{\langle \mathcal{A}_1, p(A), \alpha \rangle,\ \langle \mathcal{A}_2, p(B), \beta \rangle,\ \Pi \cup \mathcal{A}_1 \cup \mathcal{A}_2 \nvdash_\tau \bot}{\langle \mathcal{A}_1 \cup \mathcal{A}_2, p(A \cap B), \min(\alpha, \beta) \rangle}$$

5) Building Arguments by MP (MPA):

$$\frac{\langle \mathcal{A}_1, L_1, \alpha_1\rangle \;\; \langle \mathcal{A}_2, L_2, \alpha_2\rangle \;\; \ldots \;\; \langle \mathcal{A}_k, L_k, \alpha_k\rangle \quad (L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, 1) \quad \Pi \cup \bigcup_{i=1}^{k} \mathcal{A}_i \not\vdash_\tau \bot}{\langle \bigcup_{i=1}^{k} \mathcal{A}_i, L_0, \beta\rangle}$$

for any certain rule $(L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, 1) \in \Pi$, with $\beta = min(\alpha_1, \ldots, \alpha_k)$.

$$\frac{\langle \mathcal{A}_1, L_1, \alpha_1\rangle \;\; \langle \mathcal{A}_2, L_2, \alpha_2\rangle \;\; \ldots \;\; \langle \mathcal{A}_k, L_k, \alpha_k\rangle \quad (L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma), \text{ with } \gamma < 1 \quad \Pi \cup \{(L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma)\} \cup \bigcup_{i=1}^{k} \mathcal{A}_i \not\vdash_\tau \bot}{\langle \bigcup_{i=1}^{k} \mathcal{A}_i \cup \{(L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma)\}, L_0, \beta\rangle}$$

for any weighted rule $(L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma) \in \Delta$, with $\beta = min(\alpha_1, \ldots, \alpha_k, \gamma)$.

The basic idea with the argument construction procedure is to keep a trace of the set $\mathcal{A} \subseteq \Delta$ of all uncertain information in the program $\mathcal{P}$ used to derive a given goal Q with necessity degree α. Appropriate preconditions ensure that the proof obtained always ensures the non-contradictory constraint of arguments wrt the certain knowledge Π of the program. Given a context $\mathcal{I}_{U,m}$ and a DePGL$^+$ program $\mathcal{P}$, rule INTF allows to construct arguments from facts. An empty argument can be obtained for any certain fact in $\mathcal{P}$. An argument concluding an uncertain fact (Q, α) in $\mathcal{P}$ can be derived whenever assuming (Q, α) is not contradictory wrt the set Π in $\mathcal{P}$. Rules SUA and UNA accounts for semantical unification and resolving uncertainty, respectively. As both rules do not combine new uncertain knowledge, we do not need to check the non-contradictory constraint. Rule INA applies intersection between previously argumented goals provided that the resulting intersection is non contradictory wrt Π. Rules MPA account for the use of modus ponens, both with certain and defeasible rules. Note they assume the existence of an argument for every literal in the antecedent of the rule. Then, in a such a case, the MPA rule is applicable whenever no contradiction results when putting together Π, the sets $\mathcal{A}_1, \ldots, \mathcal{A}_k$ corresponding to the arguments for the antecedents of the rule and the rule $(L_0 \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma)$ when $\gamma < 1$.

Example 4 *Consider the program $\mathcal{P}_{eng}$ in Example 2, where $temp(\cdot)$ is a unary predicate of type* (degrees)*, $speed(\cdot)$ is a unary predicate of type* (rpm)*,* heat *and* around_31 *are two object constants of type* degrees*, and* low *is an object constant of type* rpm*. Further, consider the context $\mathcal{I}_{U,m}$ such that:*

- $U = \{U_{\texttt{degrees}} = [-100, 100]\ ^oC,\ U_{\texttt{rpm}} = [0, 200]\}$;
- $m(high) = [28, 30, 100, 100]$[2]*,*
  $m(around\_31) = [26, 31, 31, 36]$*,*
  $m(low) = [10, 15, 25, 30]$*, and*
  $m(\neg low) = 1 - m(low)$.

Then the following arguments can be derived from $\mathcal{P}_{eng}$:

1. *The argument $\langle \mathcal{B}, fuel\_ok, 0.6\rangle$ can be derived as follows:*
   i) $\langle \emptyset, sw1, 1\rangle$ *from (13) via INTF.*
   ii) $\langle \mathcal{B}', pump\_fuel, 0.6\rangle$ *from (2) and i) via MPA.*
   iii) $\langle \mathcal{B}, fuel\_ok, 0.6\rangle$ *from (3) and ii) via MPA.*
   where $\mathcal{B}'$=$\{(pump\_fuel \leftarrow sw1, 0.6)\}$ and $\mathcal{B} = \mathcal{B}' \cup \{(fuel\_ok \leftarrow pump\_fuel, 0.85)\}$.
2. *Similarly, the argument $\langle \mathcal{C}_1, oil\_ok, 0.8\rangle$ can be derived using the rules (15), (4) and (5) via INTC, MPA, and MPA respectively, with: $\mathcal{C}_1$ = $\{(pump\_oil \leftarrow sw2, 0.8)$; $(oil\_ok \leftarrow pump\_oil, 0.8)\}$.*
3. *The argument $\langle \mathcal{A}_1, engine\_ok, 0.6\rangle$ can be derived as follows:*
   i) $\langle \mathcal{B}, fuel\_ok, 0.6\rangle$ *as shown above.*
   ii) $\langle \mathcal{C}_1, oil\_ok, 0.8\rangle$ *as shown above.*
   iii) $\langle \mathcal{A}_1, engine\_ok, 0.6\rangle$ *from i), ii), (6) via MPA.*
   with $\mathcal{A}_1$=$\{(engine\_ok \leftarrow fuel\_ok \wedge oil\_ok, 0.6)\} \cup \mathcal{B} \cup \mathcal{C}_1$. Note that $\langle \mathcal{C}_1, oil\_ok, 0.8\rangle$ and $\langle \mathcal{B}, fuel\_ok, 0.6\rangle$ are subarguments of $\langle \mathcal{A}_1, engine\_ok, 0.6\rangle$.
4. *One can also derive the argument $\langle \mathcal{C}_2, \sim oil\_ok, 0.8\rangle$, where $\mathcal{C}_2$ = $\{(temp(around\_31), 0.85), (\sim oil\_ok \leftarrow temp(high), 0.9)\}$, as follows:*
   i) $\langle \{(temp(around\_31),\ 0.85)\}, temp(around\_31),\ 0.85\rangle$ *from (16) via INTF.*
   ii) $\langle \{(temp(around\_31),\ 0.85)\}, temp(high), 0.8\rangle$ *from i) via SUA, where $N(high \mid around\_31) = 0.8$ and $0.8 = \min(0.85, 0.8)$.*
   iii) $\langle \mathcal{C}_2, \sim oil\_ok, 0.8\rangle$ *from i), ii), (6) via MPA.*
5. *Similarly, an argument $\langle \mathcal{A}_2, \sim engine\_ok, 0.8\rangle$ can be derived using the rules (16) and (7) via INTF, SUA, and MPA, with*
   $\mathcal{A}_2$ = {(temp(around_31), 0.85);
   $(\sim engine\_ok \leftarrow temp(high), 0.95)\}$.

Counter-argumentation and defeat in DePGL$^+$

Given a program and a particular context, it can be the case that there exist conflicting arguments for one literal and its negation. For instance, in the above example, $\langle \mathcal{A}_1, engine\_ok, 0.6\rangle$ and $\langle \mathcal{A}_2, \sim engine\_ok, 0.8\rangle$, and $\langle \mathcal{C}_1, oil\_ok, 0.8\rangle$ and $\langle \mathcal{C}_2, \sim oil\_ok, 0.8\rangle$, and thus, the program $\mathcal{P}_{eng}$ considering the context $\mathcal{I}_{U,m}$ is contradictory. Therefore, it is necessary to define a formal framework for solving conflicts among arguments in DePGL$^+$. This is formalized next by the notions of counterargument and defeat, based on the same ideas used in P-DeLP (Chesñevar *et al.* 2004) but incorporating the treatment of fuzzy constants.

Definition 5 (Counterargument) *Let $\mathcal{P}$ be a DePGL$^+$ program, let $\mathcal{I}_{U,m}$ be a context, and let $\langle \mathcal{A}_1, Q_1, \alpha_1\rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2\rangle$ be two arguments wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1\rangle$ counterargues $\langle \mathcal{A}_2, Q_2, \alpha_2\rangle$ iff there exists a subargument (called* disagreement subargument*) $\langle \mathcal{S}, Q, \beta\rangle$ of $\langle \mathcal{A}_2, Q_2, \alpha_2\rangle$ such that $Q = \sim Q_1$*[3].

[2] We represent a trapezoidal fuzzy set as $[t_1; t_2; t_3; t_4]$, where the interval $[t_1, t_4]$ is the support and the interval $[t_2, t_3]$ is the core.

[3] For a given goal Q, we write $\sim Q$ as an abbreviation to denote "$\sim q$" if $Q \equiv q$ (resp.,"$\sim q(A)$" if $Q \equiv q(A)$) and "q" if $Q \equiv \sim q$ (resp., "$q(A)$" if $Q \equiv \sim q(A)$).

Since arguments rely on uncertain and hence defeasible information, conflicts among arguments may be resolved by comparing their strength and deciding which argument is defeated by which one. Therefore, a notion of defeat amounts to establish a *preference criterion* on conflicting arguments. In our framework, following (Chesñevar *et al.* 2004), it seems natural to define it on the basis of necessity degrees associated with arguments.

Definition 6 (Defeat) *Let $\mathcal{P}$ be a DePGL$^+$ program, let $\mathcal{I}_{U,m}$ be a context, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$* defeats *$\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ (or equivalently $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a* defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$) iff:*

(1) the argument $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counterargues the argument $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ with disagreement subargument $\langle \mathcal{A}, Q, \alpha \rangle$; and

(2) either it holds that $\alpha_1 > \alpha$, in which case $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ will be called a proper defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$, or $\alpha_1 = \alpha$, in which case $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ will be called a* blocking defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$.*

Following Examples 2 and 4, we have that argument $\langle \mathcal{A}_2, \sim engine\_ok, 0.8 \rangle$ is a defeater of argument $\langle \mathcal{A}_1, engine\_ok, 0.6 \rangle$ while $\langle \mathcal{C}_2, \sim oil\_ok, 0.8 \rangle$ is a blocking defeater of $\langle \mathcal{C}_1, oil\_ok, 0.8 \rangle$.

Computing warranted arguments in DePGL$^+$

As in most argumentation systems, a main goal in DePGL$^+$ is to devise a procedure to determine whether a given argument $\langle \mathcal{A}, Q, \alpha \rangle$ is *warranted* (or ultimately accepted) wrt a program $\mathcal{P}$. Intuitively, an argument $\langle \mathcal{A}, Q, \alpha \rangle$ is warranted when

1. it has no defeaters, or
2. every defeater for $\langle \mathcal{A}, Q, \alpha \rangle$ is on its turn defeated by another argument which is warranted.

In P-DeLP this is done by an exhaustive dialectical analysis of all argumentation lines rooted in a given argument (see (Chesñevar *et al.* 2004) for details) which can be efficiently performed by means of a top-down algorithm, as described in (Chesñevar, Simari, & Godo 2005). For instance, given the following simple P-DeLP program $\mathcal{P} = \{(p, 0.45), (\sim p, 0.7)\}$, a short dialectical analysis would conclude that the argument $A = \langle \{(\sim p, 0.7)\}, \sim p, 0.7 \rangle$ is warranted.

However, even with similar simple programs, the situation DePGL$^+$ gets more involved. Indeed, in order to provide DePGL$^+$ with a similar dialectical analysis, due to the disjunctive interpretation of fuzzy constants and their associated fuzzy unification mechanism, new blocking situations between arguments have to be considered as we show in the following example.

Example 7 *Consider the DePGL$^+$ program*

$$\mathcal{P} = \{(temp(around\_31), 0.45), \\ (temp(between\_25\_30), 0.7)\}$$

where $temp(\cdot)$ is a unary predicate of type (degrees)*, and the context $\mathcal{I}_{U,m}$ with $U = \{U_{\texttt{degrees}} = [-100, 100]\ {}^oC\}$ and*

$m(around\_31) = [26, 31, 31, 36]$,
$m(between\_25\_30) = [20, 25, 30, 35]$,
$m(\neg around\_31) = 1 - m(around\_31)$, *and*
$m(\neg between\_25\_30) = 1 - m(between\_25\_30)$.

Consider the following sets of clauses:

$\mathcal{A}_1 = \{(temp(around\_31), 0.45)\}$
$\mathcal{A}_2 = \{(temp(between\_25\_30), 0.7)\}$.

Within the context $\mathcal{I}_{U,m}$, the arguments

$A_1 = \langle \mathcal{A}_1, temp(around\_31), 0.45 \rangle$,
$A_2 = \langle \mathcal{A}_2, temp(between\_25\_30), 0.7 \rangle$,

can be derived from $\mathcal{P}$, but notice that $m(around\_31) \cap m(between\_25\_30)$ is a non-normalized fuzzy set. However, since we have

$N(m(\neg around\_31) \mid m(between\_25\_30)) = 0$
$N(m(\neg between\_25\_30) \mid m(around\_31)) = 0$,

using the SUA procedural rule, one can only derive arguments for the negated literals $\sim temp(around\_31)$ and $\sim temp(between\_25\_30)$ with necessity degree 0. Hence, neither A_1 nor A_2 has a proper defeater. Then, in this particular context, neither A_1 nor A_2 can be warranted, and thus A_1 acts as a blocking argument for A_2, and viceversa.

Remark that the unification degree, or the partial matching, between fuzzy constants depends on the context we are considering. For instance, if for the above context $\mathcal{I}_{U,m}$ we consider the Gödel negation instead of the involutive negation; i.e.,

$$m(\neg A)(t) \quad = \quad \begin{cases} 1, & \text{if } m(A)(t) = 0 \\ 0, & \text{otherwise} \end{cases}$$

for any fuzzy constant A, we get that

$$N(m(\neg around\_31) \mid m(between\_25\_30)) \quad = \quad 0.2$$
$$N(m(\neg between\_25\_30) \mid m(around\_31)) \quad = \quad 0.2$$

However, as $0.2 < 0.45$ and $0.2 < 0.7$, in this new particular context neither A_1 nor A_2 can be warranted as well.

Therefore we introduce the following notion of pair of blocking arguments.

Definition 8 (Blocking arguments) *Let $\mathcal{P}$ be a DePGL$^+$ program, let $\mathcal{I}_{U,m}$ be a context, and let $\langle \mathcal{A}_1, q(A), \alpha_1 \rangle$ and $\langle \mathcal{A}_2, q(B), \alpha_2 \rangle$ be two arguments wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$. We will say that $\langle \mathcal{A}_1, q(A), \alpha_1 \rangle$* blocks *$\langle \mathcal{A}_2, q(B), \alpha_2 \rangle$, and viceversa, when*

1. *$m(A) \cap m(B)$ is a non-normalized fuzzy set; and*
2. *$N(m(\neg A) \mid m(B)) < \alpha_1$ and $N(m(\neg B) \mid m(A)) < \alpha_2$.*

By extension, if $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a subargument of $\langle \mathcal{A}, Q, \alpha \rangle$ and $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ are a pair of blocking arguments, argument $\langle \mathcal{A}, Q, \alpha \rangle$ cannot be warranted and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ is a *blocking argument* for $\langle \mathcal{A}, Q, \alpha \rangle$.

Given a DePGL$^+$ program and a particular context, there may exist both multiple blocking arguments and multiple proper defeaters for a same argument, all of them derived from a same set of clauses by applying the semantical unification procedural rule SUA as we show in the following example.

Example 9 *Consider the DePGL$^+$ program $\mathcal{P}$ and the context $\mathcal{I}_{U,m}$ of Example 7. Let*

$$\mathcal{A}_3 = \{(temp(about\_25), 0.9)\},$$

and let $\mathcal{P}' = \mathcal{P} \cup \mathcal{A}_3$ be a new program. Further, consider two new fuzzy constants "$between\_31\_32$" and "$about\_25\_ext$". The three new fuzzy constants are interpreted in the context $\mathcal{I}_{U,m}$ as

$m(about\_25) = [24, 25, 25, 26]$,
$m(\neg about\_25) = 1 - m(about\_25)$,
$m(between\_31\_32) = [26, 31, 32, 37]$, *and*
$m(about\_25\_ext) = [24, 25, 25, 32]$.

Notice that arguments A_1 and A_2 from Example 7 are still arguments with respect the new program $\mathcal{P}'$. Now, in the frame of the program $\mathcal{P}'$, from the argument A_1 and by applying the SUA procedural rule, we can build the argument

$$A_3 = \langle \mathcal{A}_1, temp(between\_31\_32), 0.45 \rangle,$$

since $N(m(between\_31\_32) \mid m(around\_31)) = 1$. One can easily check that A_3 and A_2 are a pair of blocking arguments. Moreover, as $m(around\_31) \leq m(between\_31\_32)$, i.e. "$around\_31$" is more specific *than "$between\_31\_32$", we have $N(m(\neg between\_25\_30) \mid m(around\_31)) \geq N(m(\neg between\_25\_30) \mid m(between\_31\_32))$, and thus, the argument A_3 can be considered as a redundant blocking argument for the argument A_2.*

On the other hand, the argument

$$A_4 = \langle \mathcal{A}_3, temp(about\_25), 0.9 \rangle,$$

can be derived from $\mathcal{P}'$. Then, from the argument A_4 and by applying the SUA procedural rule, we can build the argument

$$A_5 = \langle \mathcal{A}_3, \sim temp(around\_31), 0.9 \rangle,$$

since $N(m(\neg around\_31) \mid m(about\_25)) = 1$, and thus, the argument A_5 is a proper defeater for the argument A_1. Now, from the argument A_4 and by applying the SUA procedural rule, we can build the argument

$$A_6 = \langle \mathcal{A}_3, temp(about\_25\_ext), 0.9 \rangle,$$

since $N(m(about\_25\_ext) \mid m(about\_25)) = 1$. Finally, from the argument A_6 and by applying the SUA procedural rule, we can build the argument

$$A_7 = \langle \mathcal{A}_3, \sim temp(around\_31), 0.5 \rangle,$$

since $N(m(\neg around\_31) \mid m(about\_25\_ext)) = 0.5$, and thus, the argument A_7 is a proper defeater for the argument A_1. However, as arguments A_5 and A_7 have been computed both from the same specific information of the program and $0.9 > 0.5$, the argument A_7 can be considered as a redundant proper defeater for the argument A_1.

Therefore, if we aim at an efficient procedure for computing warrants (based on an exhaustive dialectical analysis of all argumentation lines), we have to avoid for a given argument both redundant blocking arguments and redundant proper defeaters . According to the above discussion, we introduce the following definitions of redundant blocking arguments and defeaters.

Definition 10 (Redundant blocking arguments) *Let $\mathcal{P}$ be a DePGL$^+$ program, let $\mathcal{I}_{U,m}$ be a context, and let $\langle \mathcal{A}_1, p(A), \alpha_1 \rangle$ and $\langle \mathcal{A}_2, p(B), \alpha_2 \rangle$ be a pair of blocking arguments wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$. We will say that $\langle \mathcal{A}_2, p(B), \alpha_2 \rangle$ is a* redundant blocking argument *for $\langle \mathcal{A}_1, p(A), \alpha_1 \rangle$ iff there exists an argument $\langle \mathcal{A}_2, p(C), 1 \rangle$ such that:*

1. *$\langle \mathcal{A}_1, p(A), \alpha_1 \rangle$ and $\langle \mathcal{A}_2, p(C), 1 \rangle$ are a pair of blocking arguments; and*
2. *$m(C) \leq \max(1 - \alpha_2, m(B))$.*

Definition 11 (Redundant defeater) *Let $\mathcal{P}$ be a DePGL$^+$ program, let $\mathcal{I}_{U,m}$ be a context, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$ such that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a proper defeater for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a* redundant defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff there exists an argument $\langle \mathcal{A}_1, Q_1, \alpha \rangle$ such that:*

1. *$\langle \mathcal{A}_1, Q_1, \alpha \rangle$ is is a proper defeater for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$); and*
2. *$\alpha_1 < \alpha$.*

At this point we are ready to formalize the notion of argumentation line in the framework of DePGL$^+$. An *argumentation line* starting in an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ is a sequence of arguments

$$\lambda = [\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle, \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle, \ldots, \langle \mathcal{A}_n, Q_n, \alpha_n \rangle, \ldots]$$

that can be thought of as an exchange of arguments between two parties, a *proponent* (evenly-indexed arguments) and an *opponent* (oddly-indexed arguments). Each $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ is *either a defeater or a blocking argument* for the previous argument $\langle \mathcal{A}_{i-1}, Q_{i-1}, \alpha_{i-1} \rangle$ in the sequence, $i > 0$. In order to avoid *fallacious* reasoning, argumentation theory imposes additional constraints on such an argument exchange to be considered rationally acceptable wrt a DePGL$^+$ program $\mathcal{P}$ and a context $\mathcal{I}_{U,m}$, namely:

1. **Non-contradiction:** given an argumentation line λ, the set of arguments of the proponent (resp. opponent) should be *non-contradictory* wrt $\mathcal{P}$ and $\mathcal{I}_{U,m}$.
2. **Progressive argumentation:** every[4] blocking defeater and blocking argument $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ in λ, $i > 0$, is defeated by a proper defeater $\langle \mathcal{A}_{i+1}, Q_{i+1}, \alpha_{i+1} \rangle$ in λ.
3. **Non-redundancy:** every proper defeater and blocking argument $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ in λ, $i > 0$, is a non-redundant defeater, resp. a non-redundant blocking argument, for the previous argument $\langle \mathcal{A}_{i-1}, Q_{i-1}, \alpha_{i-1} \rangle$ in λ; i.e. $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ is the best proper defeater or the most specific blocking argument one can consider from a given set of clauses.

The first condition disallows the use of contradictory information on either side (proponent or opponent). The second condition enforces the use of a proper defeater to defeat an argument which acts as a blocking defeater or as a blocking argument. An argumentation line satisfying restrictions

[4] Remark that the last argument in an argumentation line is allowed to be a blocking defeater and a blocking argument for the previous one.

(1) and (2) is called *acceptable*, and can be proven to be finite. Finally, since we consider programs with a finite set of clauses, the last condition ensures that we have a computable number of argumentations lines.

Given a program $\mathcal{P}$, a context $\mathcal{I}_{U,m}$ and an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$, the set of all acceptable argumentation lines starting in $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ accounts for a whole dialectical analysis for $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ (i.e. all possible dialogues rooted in $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$, formalized as a *dialectical tree*[5]).

Definition 12 (Warrant) *Given a program $\mathcal{P} = (\Pi, \Delta)$, a context $\mathcal{I}_{U,m}$, and a goal Q, we will say that Q is* warranted *wrt $\mathcal{P}$ in the context $\mathcal{I}_{U,m}$ with a* maximum necessity degree *α iff there exists an argument of the form $\langle \mathcal{A}, Q, \alpha \rangle$, for some $\mathcal{A} \subseteq \Delta$, such that:*

1. *every acceptable argumentation line starting with $\langle \mathcal{A}, Q, \alpha \rangle$ has an odd number of arguments; i.e. every argumentation line starting with $\langle \mathcal{A}, Q, \alpha \rangle$ finishes with an argument proposed by the proponent which is in favor of Q with at least a necessity degree α; and*
2. *for each argument of the form $\langle \mathcal{A}_1, Q, \beta \rangle$, with $\beta > \alpha$, there exists at least an acceptable argumentation line starting with $\langle \mathcal{A}_1, Q, \beta \rangle$ that has an even number of arguments.*

Note that we will generalize the use of the term "warranted" for applying it to both goals and arguments: whenever a goal Q is warranted on the basis of a given argument $\langle \mathcal{A}, Q, \alpha \rangle$ as specified in Def. 12, we will also say that the argument $\langle \mathcal{A}, Q, \alpha \rangle$ is warranted. Continuing with Examples 7 and 9, we will next show how to determine, according to the above definition, whether some arguments appearing there (arguments A_4, A_1 and A_2) are warranted.

Example 13 *Let us recall the following arguments:*

$A_1 = \langle \mathcal{A}_1, temp(around\_31), 0.45 \rangle$,
$A_2 = \langle \mathcal{A}_2, temp(between\_25\_30), 0.7 \rangle$,
$A_4 = \langle \mathcal{A}_3, temp(about\_25), 0.9 \rangle$,
$A_5 = \langle \mathcal{A}_3, \sim temp(around\_31), 0.9 \rangle$.

Consider first the argument A_4. It has neither a proper defeater nor a blocking argument, hence there exists an acceptable argumentation line starting with A_4 with just one argument. Indeed, the only possible argumentation line rooted in A_4 that can be obtained is $[A_4]$. Since this line has odd length, according to Definition 12 the goal "$temp(about\_25)$" can be warranted wrt $\mathcal{P}'$ in the context $\mathcal{I}_{U,m}$ with a necessity of 0.9.

Consider now the case of argument A_1. In this case, the argument A_5 is a non-redundant proper defeater for A_1 and A_5 has no defeater, since "$temp(about\_25)$" is a warranted goal with a necessity of 0.9. Similarly, the argument A_2 is a non-redundant blocking argument for A_1, but A_2 has a proper defeater, namely A_4. However, the line $[A_1, A_2, A_4]$ is not allowed because A_1 and A_4 are contradictory since $m(around\_31) \cap m(about\_25)$ is not normalized. Therefore two acceptable argumentation lines rooted at A_1 can be built: $[A_1, A_5]$ and $[A_1, A_2]$. Since it is not the case that every argumentation line rooted in A_1 has odd length, the argument A_1 cannot be warranted.

Finally, following a similar discussion for A_2, we can conclude that the argument A_2 is not warranted either.

It must be noted that to decide whether a given goal Q is warranted (on the basis of a given argument A_0 for Q) it may be not necessary to compute *every* possible argumentation line rooted in A_0, e.g. in the case of A_1 in the previous example, it sufficed to detect just one even-length argumentation line to determine that is not warranted. Some aspects concerning computing warrant efficiently by means of a top-down procedure in P-DeLP can be found in (Chesñevar, Simari, & Godo 2005).

Related work

To the best of our knowledge, in the literature there have been not many approaches that aim at combining argumentation and fuzziness, except for the work of Schroeder & Schweimeier (Schweimeier & Schroeder 2001; Schroeder & Schweimeier 2002; Schweimeier & Schroeder 2004). The argumentation framework is also defined for a logic programming framework based on extended logic programming with well-founded semantics, and providing a declarative bottom-up fixpoint semantics along with an equivalent top-down proof procedure. In contrast with our approach, this argumentation framework defines fuzzy unification on the basis of the notion of edit distance, based on string comparison (Schweimeier & Schroeder 2004). Their proposal, on the other hand, does not include an explicit treatment of possibilistic uncertainty as in our case.

There has been generic approaches connecting defeasible reasoning and possibilistic logic (e.g.(Benferhat, Dubois, & Prade 2002)). Including possibilistic logic as part of an argumentation framework for modelling preference handling and information merging has recently been treated by Amgoud & Kaci (Amgoud & Kaci 2005) and Amgoud & Cayrol (Amgoud & Cayrol 2002). Such formulations are based on using a possibilistic logic framework to handle merging of prioritized information, obtaining an aggregated knowledge base. Arguments are then analyzed on the basis of the resulting aggregated knowledge base. An important difference of these proposals with our formulation is that our framework introduces explicit representation of fuzziness along with handling possibilistic logic. Besides, in the proposed framework we attach necessity degrees to object level formulas, propagating such necessity degrees according to suitable inference rules, which differs from the approach used in the proposals above mentioned.

Besides of considering possibilistic logic and fuzziness, a number of hybrid approaches connecting argumentation and uncertainty have been developed, such as Probabilistic Argumentation Systems (Haenni, Kohlas, & Lehmann 2000; Haenni & Lehmann 2003), which use probabilities to compute degrees of support and plausibility of goals, related to Dempster-Shafer belief and plausibility functions. However

[5] It must be remarked that the definition of dialectical tree as well as the characterization of constraints to avoid fallacies in argumentation lines can be traced back to (Simari, Chesñevar, & García 1994). Similar formalizations were also used in other argumentation frameworks (e.g. (Prakken & Sartor 1997)).

this approach is not based on a dialectical theory (with arguments, defeaters, etc.) nor includes fuzziness as presented in this paper.

Conclusions and future work

PGL^+ constitutes a powerful formalism that can be integrated into an argument-based framework like P-DeLP, allowing to combine uncertainty expressed in possibilistic logic and fuzziness characterized in terms of fuzzy constants and fuzzy propositional variables.

In this paper we have focused on characterizing $DePGL^+$, a formal language that combines features from PGL^+ along with elements which are present in most argumentative frameworks (like the notions of argument, counterargument, and defeat). As stated in Sections 5 and 6, part of our current work is focused on providing a formal characterization of warrant in the context of the proposed framework. In particular, we are interested in studying formal properties for warrant that should hold in the context of argumentation frameworks, as proposed in (Caminada & Amgoud 2005). In this paper, Caminada & Amgoud identify anomalies in several argumentation formalisms and provide an interesting solution in terms of rationality postulates which –the authors claim– should hold in any well-defined argumentative system. In (Chesñevar *et al.* 2005) we started a preliminary analysis for this problem in the context of P-DeLP (Chesñevar *et al.* 2004), and currently part of our research is focused on this issue. We are also analyzing how to characterize an alternative conceptualization of warrant in which different warrant degrees can be attached to formulas on the basis of necessity degrees, extending some concepts suggested in (Pollock 2001). Research in these directions is currently being pursued.

Acknowledgments

We thank anonymous reviewers for their comments and suggestions to improve the final version of this paper. This work was supported by Spanish Projects TIC2003-00950, TIN2004-07933-C03-01/03, by Ramón y Cajal Program (MCyT, Spain), by CONICET (Argentina), by the Secretaría General de Ciencia y Tecnología de la Universidad Nacional del Sur and by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13096).

References

Alsinet, T., and Godo, L. 2000. A complete calculus for possibilistic logic programming with fuzzy propositional variables. In *Proc. of UAI-2000 Conf.*, 1–10.

Alsinet, T., and Godo, L. 2001. A proof procedure for possibilistic logic programming with fuzzy constants. In *Proc. of the ECSQARU-2001 Conf.*, 760–771.

Alsinet, T. 2003. *Logic Programming with Fuzzy Unification and Imprecise Constants: Possibilistic Semantics and Automated Deduction*. Number 15. IIIA-CSIC. Bellaterra, Spain.

Amgoud, L., and Cayrol, C. 2002. Inferring from inconsistency in preference-based argumentation frameworks. *J. Autom. Reasoning* 29(2):125–169.

Amgoud, L., and Kaci, S. 2005. An argumentation framework for merging conflicting knowledge bases: The prioritized case. In *Proc. of the ECSQARU-2005 Conf.*, LNAI 3571, 527–538.

Benferhat, S.; Dubois, D.; and Prade, H. 2002. The possibilistic handling of irrelevance in exception-tolerant reasoning. *Annals of Math. and AI* 35:29–61.

Caminada, M., and Amgoud, L. 2005. An axiomatic account of formal argumentation. In *Proc. of the AAAI-2005 Conf.*, 608–613.

Chesñevar, C. I.; Simari, G.; Alsinet, T.; and Godo, L. 2004. A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge. In *Proc. of the UAI-2004 Conf.*, 76–84.

Chesñevar, C.; Simari, G.; Godo, L.; and Alsinet, T. 2005. On warranted inference in possibilistic defeasible logic programming. In *Proc. of CCIA-2005*. IOS Press, 265–272.

Chesñevar, C.; Maguitman, A.; and Loui, R. 2000. Logical Models of Argument. *ACM Computing Surveys* 32(4):337–383.

Chesñevar, C.; Simari, G.; and Godo, L. 2005. Computing dialectical trees efficiently in possibilistic defeasible logic programming. In *Proc. of LPNMR-2005 Conf.*, 158–171.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic. In D.Gabbay et al. eds., *Handbook of Logic in Art. Int. and Logic Prog. (Nonmonotonic Reasoning and Uncertain Reasoning)*. Oxford Univ. Press. 439–513.

Haenni, R., and Lehmann, N. 2003. Probabilistic Argumentation Systems: a New Perspective on Dempster-Shafer Theory. *Int. J. of Intelligent Systems* 1(18):93–106.

Haenni, R.; Kohlas, J.; and Lehmann, N. 2000. Probabilistic argumentation systems. *Handbook of Defeasible Reasoning and Uncertainty Management Systems*.

Pollock, J. L. 2001. Defeasible reasoning with variable degrees of justification. *Artif. Intell.* 133(1-2):233–282.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics* 7:25–75.

Schroeder, M., and Schweimeier, R. 2002. Fuzzy argumentation for negotiating agents. In *Proc. of the AAMAS-2002 Conf.*, 942–943.

Schweimeier, R., and Schroeder, M. 2001. Fuzzy argumentation and extended logic programming. In *Proceedings of ECSQARU Workshop Adventures in Argumentation (Toulouse, France)*.

Schweimeier, R., and Schroeder, M. 2004. Fuzzy unification and argumentation for well-founded semantics. In *proc. of SOFSEM 2004*, LNCS 2932, 102–121.

Simari, G.; Chesñevar, C.; and García, A. 1994. The role of dialectics in defeasible argumentation. In *Proc. of the XIV Intl. Conf. of the Chilean Society for Computer Science*, 260–281. Universidad de Concepción, Concepción (Chile).

2.14 Preference reasoning for argumentation: Non-monotonicity and algorithms

Preference Reasoning for Argumentation: Non-monotonicity and Algorithms

Souhila Kaci
CRIL
Rue de l'Université SP 16
62307 Lens France
kaci@cril.univ-artois.fr

Leendert van der Torre
ILIAS
University of Luxembourg
Luxembourg
leon.vandertorre@uni.lu

Abstract

In this paper we are interested in the role of preferences in argumentation theory. To promote a higher impact of preference reasoning in argumentation, we introduce a novel preference-based argumentation theory. Using non-monotonic preference reasoning we derive a Dung-style attack relation from a preference specification together with a defeat relation. In particular, our theory uses efficient algorithms computing acceptable arguments via a unique preference relation among arguments from a preference relation among sets of arguments.

Introduction

Dung's theory of abstract argumentation (Dung 1995) is based on a set of arguments and a binary attack relation defined over the arguments. Due to this abstract representation, it can and has been used in several ways, which may explain its popularity in artificial intelligence. It has been used as a general framework for non-monotonic reasoning, as a framework for argumentation, and as a component in agent communication, dialogue, decision making, *etc*. Dung's abstract theory has been used mainly in combination with more detailed notions of arguments and attack, for example arguments consisting of rules, arguments consisting of a justification and a conclusion, or attack relations distinguishing rebutting and undercutting. However, there have also been several attempts to modify or generalize Dung's theory, for example by introducing preferences (Amgoud & Cayrol 2002; Kaci, van der Torre, & Weydert 2006), defeasible priorities (Prakken & Sartor 1997; Poole 1985; Simari & Loui 1992; Stolzenburg *et al.* 2003), values (Bench-Capon 2003), or collective arguments (Bochman 2005).

In this paper we are interested in the role of preference reasoning in Dung's argumentation theory. An example from political debate has been discussed by Bench-Capon *et al.* (Atkinson, Bench-Capon, & McBurney 2005), where several arguments to invade Iraq are related to values such as respect for life, human rights, good world relations, and so on. In this paper we use a less controversial example to illustrate our theory where several arguments used in a debate between parents and their children are used to promote values like staying healthy, doing well at school, and so on.

In our theory, we integrate two existing approaches (though our approach differs both conceptually and technically from these approaches in several significant ways, as explained in the related work).

- We consider a preference based argumentation theory consisting of a set of arguments, an attack relation, and a preference relation over arguments. Then, like Amgoud and Cayrol (Amgoud & Cayrol 2002), we transform this preference based argumentation theory to Dung's theory, by stating that an argument A attacks another argument B in Dung's theory, when A attacks B in the preference-based theory, and B is not preferred to A. To distinguish the two notions of attack, we call the notion of attack in the preference-based theory *defeat*. The defeat and preference relation may be considered as an alternative representation of Dung's attack relation.
- Like Bench-Capon (Bench-Capon 2003), we consider value based argumentation, in which arguments are used to promote a value, and in which values are ordered by a preference relation. Moreover, in contrast to Bench-Capon, we use non-monotonic preference reasoning to reduce the ordered values to a preference relation over arguments. In analogy with the above, we say that the ordered values represent the preference relation over arguments.

Summarizing, starting with a set of arguments, a defeat relation, and an ordered set of values, we use the ordered values to compute a preference relation over arguments, and we combine this preference relation with the defeat relation to compute Dung's attack relation. Then we use any of Dung's semantics to define the acceptable set of arguments. In contrast to most other approaches (Amgoud & Cayrol 2002; Prakken & Sartor 1997; Poole 1985; Simari & Loui 1992; Stolzenburg *et al.* 2003) (but see (Amgoud, Parsons, & Perrussel 2000) for an exception), our approach to reason about preferences in argumentation does not refer to the internal structure of the arguments. We study the following research questions:

1. How to reason about ordered values and to derive a preference relation over arguments?
2. How to combine the two steps of our approach to directly define the acceptable set of arguments from a defeat relation and an ordered set of values?

To reason about ordered values and to compute the preference relation over arguments, we are inspired by insights from the non-monotonic logic of preference (Kaci & van der Torre 2005). When value v_1 is promoted by the arguments $A_1, \ldots, A_n$, and value v_2 is promoted by arguments $B_1, \ldots, B_m$, then the statement that value v_1 is preferred to value v_2 means that the set of arguments $A_1, \ldots, A_n$ is preferred to the set of arguments $B_1, \ldots, B_m$. In other words, the problem of reducing ordered values to a preference relation comes down to reducing a preference relation over sets of arguments to a preference relation over single arguments. We use both so-called optimistic and pessimistic reasoning to define the preference relation.

For the combined approach, we restrict ourselves to Dung's grounded semantics. For this semantics, we introduce an algorithm that shows how the computation of set of acceptable arguments can be combined with the optimistic reasoning to incrementally define the set of acceptable arguments, and we show why this works less well for pessimistic reasoning.

The layout of this paper is as follows. After presenting Dung's abstract theory of argumentation, and its extension to the preference-based argumentation framework, we introduce our value based argumentation theory, and show how to reduce a value based argumentation theory to a preference-based argumentation theory using optimistic or pessimistic reasoning. Then we introduce an algorithm for directly computing the set of acceptable arguments using grounded semantics. We also present an algorithm for ordering the arguments following the pessimistic reasoning. Lastly we discuss related work and conclude.

Abstract argumentation

Argumentation is a reasoning model based on constructing arguments, determining potential conflicts between arguments and determining acceptable arguments.

Dung's argumentation framework

Dung's framework (Dung 1995) is based on a binary attack relation among arguments.

Definition 1 (Argumentation framework) *An argumentation framework is a tuple $\langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R}$ is a binary attack relation defined on $\mathcal{A} \times \mathcal{A}$.*

We restrict ourselves to *finite* argumentation frameworks, i.e., when the set of arguments $\mathcal{A}$ is *finite*.

Definition 2 (Defence) *A set of arguments $\mathcal{S}$ defends A if for each argument B of $\mathcal{A}$ which attacks A, there is an argument C in $\mathcal{S}$ which attacks B.*

Definition 3 (Conflict-free) *Let $\mathcal{S} \subseteq \mathcal{A}$. The set $\mathcal{S}$ is conflict-free iff there are no $A, B \in \mathcal{S}$ such that $A\mathcal{R}B$.*

The following definition summarizes different acceptable semantics of arguments proposed in the literature:

Definition 4 (Acceptability semantics) *Let $\mathcal{S} \subseteq \mathcal{A}$.*

- *$\mathcal{S}$ is* admissible *iff it is conflict-free and defends all its elements.*
- *A conflict-free $\mathcal{S}$ is a* complete extension *iff $\mathcal{S} = \{A \mid \mathcal{S}$ defends $A\}$.*
- *$\mathcal{S}$ is a* grounded extension *iff it is the smallest (for set inclusion) complete extension.*
- *$\mathcal{S}$ is a* preferred extension *iff it is the largest (for set inclusion) complete extension.*
- *$\mathcal{S}$ is a* stable extension *iff it is a preferred extension that attacks all arguments in $\mathcal{A} \backslash \mathcal{S}$.*

The output of the argumentation framework is derived from the set of selected acceptable arguments w.r.t. an acceptability semantics.

Preference-based argumentation framework

An extended version of Dung's framework (Dung 1995) has been proposed in (Amgoud & Cayrol 2002) where a preference relation is defined on the set of arguments on the basis of the evaluation of arguments. We start with some definitions concerning preferences.

Definition 5 *A pre-order on a set $\mathcal{A}$, denoted $\succeq$, is a reflexive and transitive relation. $\succeq$ is total if it is complete and it is partial if it is not. The notation $A_1 \succeq A_2$ stands for A_1 is at least as preferred as A_2. $\succ$ denotes the order associated with $\succeq$. We write $\max(\succeq, \mathcal{A})$ for $\{B \in \mathcal{A}, \nexists B' \in \mathcal{A}$ s.t. $B' \succ B\}$ and we write $\min(\succeq, \mathcal{A})$ as $\{B \in \mathcal{A}, \nexists B' \in \mathcal{A}$ s.t. $B \succ B'\}$.*

Definition 6 illustrates how a total pre-order on $\mathcal{A}$ can also be represented by a well ordered partition of $\mathcal{A}$. This is an equivalent representation, in the sense that each total pre-order corresponds to one ordered partition and vice versa. This equivalent representation as an ordered partition makes some definitions easier to read.

Definition 6 (Ordered partition) *A sequence of sets of arguments of the form $(E_1, \cdots, E_n)$ is the ordered partition of $\mathcal{A}$ w.r.t. $\succeq$ iff*

- *$E_1 \cup \cdots \cup E_n = \mathcal{A}$,*
- *$E_i \cap E_j = \emptyset$ for $i \neq j$,*
- *$\forall A, B \in \mathcal{A}$, $A \in E_i$ and $B \in E_j$ with $i < j$ iff $A \succ B$.*

An ordered partition of $\mathcal{A}$ is associated with pre-order $\succeq$ on $\mathcal{A}$ iff $\forall A, B \in \mathcal{A}$ with $A \in E_i, B \in E_j$ we have $i \leq j$ iff $A \succeq B$.

Definition 7 (Preference-based argumentation framework) *A preference-based argumentation framework is a triplet $\langle \mathcal{A}, \mathcal{D}, \succeq \rangle$ where $\mathcal{A}$ is a set of arguments, $\mathcal{D}$ is a binary defeat relation defined on $\mathcal{A} \times \mathcal{A}$ and $\succeq$ is a (total or partial) pre-order (preference relation) defined on $\mathcal{A} \times \mathcal{A}$.*

The attack relation is defined on the basis of defeat $\mathcal{D}$ and preference relation $\succeq$, and therefore also the other relations defined by Dung are reused by the preference-based argumentation framework.

Definition 8 *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\langle \mathcal{A}, \mathcal{D}, \succeq \rangle$ a preference-based argumentation framework. We say that $\langle \mathcal{A}, \mathcal{D}, \succeq \rangle$ represents $\langle \mathcal{A}, \mathcal{R} \rangle$ iff for all arguments A and B of $\mathcal{A}$, we have $A \mathcal{R} B$ iff $A \mathcal{D} B$ and it is not the case that $B \succ A$. We also say that $\mathcal{R}$ is represented by $\mathcal{D}$ and $\succeq$.*

From this definition follows immediately that when $\succeq$ is a total pre-order, we have: $A \mathcal{R} B$ iff $A \mathcal{D} B$ and $A \succeq B$.

Preference reasoning

In most preference-based argumentation frameworks, the preference order on arguments is based on an evaluation of single arguments (Amgoud, Cayrol, & LeBerre 1996). It consists in computing the strength of the argument on the basis of knowledge from which it is built, knowledge being pervaded with implicit or explicit priorities. Note however that knowledge is not always pervaded with priorities which makes it difficult to use this way to evaluate arguments. Moreover one may also need to express more sophisticated preferences such as preferences among *sets* of *abstract* arguments without referring to their internal structure. We adapt in this paper a preference logic of non-monotonic reasoning (Kaci & van der Torre 2005) to the context of argumentation framework. Let p and q be two values. A preference of p over q, denoted $p >> q$, is interpreted as a preference of arguments promoting p over arguments promoting q.

Definition 9 (Value based argumentation framework) *A value based argumentation framework (VAF) is a 5-tuple $\langle \mathcal{A}, \mathcal{D}, V, >>, arg \rangle$ where $\mathcal{A}$ is a set of arguments, $\mathcal{D}$ is a defeat relation, V is a set of values, $>>$ is a total or partial order on V, called a preference specification, and arg is a function from V to $2^{\mathcal{A}}$ s.t. $arg(v)$ is the set of arguments supporting the value v.*

Given a preference specification the logic allows to compute a total pre-order over the set of all arguments. We are interested here in computing a *unique* total pre-order that satisfies the preference specification. Let $\succeq$ be the total pre-order that we intend to compute. A preference of p over q may be interpreted in two ways:

(1) either we compare the *best* arguments in favor of p and the *best* arguments in favor of q w.r.t. $\succeq$. In this case we say that $\succeq$ satisfies $p >> q$ iff $\forall A \in \max(arg(p), \succeq), \forall B \in \max(arg(q), \succeq)$ we have $A \succ B$.

(2) or we compare the *worst* arguments in favor of p and the *worst* arguments in favor of q w.r.t. $\succeq$. In this case we say that $\succeq$ satisfies $p >> q$ iff $\forall A \in \min(arg(p), \succeq), \forall B \in \min(arg(q), \succeq)$ we have $A \succ B$.

Comparing the worst arguments of $arg(p)$ and the best arguments of $arg(q)$ w.r.t. $\succeq$ can be reduced to comparing single arguments (see the related work). So they can be used in both above items. Comparing the best arguments of $arg(p)$ and the worst arguments of $arg(q)$ w.r.t. $\succeq$ is irrelevant (Kaci & van der Torre 2005).

Definition 10 (Model of a preference specification) *$\succeq$ satisfies (or is a model of) a preference specification $\mathcal{P} = \{p_i >> q_i : i = 1, \cdots, n\}$ iff $\succeq$ satisfies each $p_i >> q_i$ in $\mathcal{P}$.*

The above two cases correspond to two different reasonings: an *optimistic* reasoning which applies to the first case since we compare the best arguments w.r.t. $\succeq$, and a *pessimistic* reasoning which applies to the second case since we compare the worst arguments w.r.t. $\succeq$.

The optimistic reasoning corresponds to the minimal specificity principle in non-monotonic reasoning (Pearl 1990). Following this principle there is a unique model of $\mathcal{P}$. This model, called the least specific model of $\mathcal{P}$, is characterized as gravitating towards the ideal since arguments are put in the highest possible rank in the pre-order $\succeq$. The pessimistic reasoning behaves in an opposite way and corresponds to the maximal specificity principle in non-monotonic reasoning. Following this principle there is also a unique model of $\mathcal{P}$ (Benferhat *et al.* 2002). This pre-order, called the most specific model of $\mathcal{P}$, is characterized as gravitating towards the worst since arguments are put in the lowest possible rank in the pre-order $\succeq$.

Definition 11 (Minimal/Maximal specificity principle) *Let $\succeq$ and $\succeq'$ be two total pre-orders on a set of arguments $\mathcal{A}$ represented by ordered partitions $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_m)$ respectively. We say that $\succeq$ is at least as specific as $\succeq'$, written as $\succeq \sqsubseteq \succeq'$, iff $\forall A \in \mathcal{A}$, if $A \in E_i$ and $A \in E'_j$ then $i \leq j$.*
$\succeq$ belongs to the set of the least (resp. most) specific pre-orders among a set of pre-orders $\mathcal{O}$ if there is no $\succeq'$ in $\mathcal{O}$ such that $\succeq' \sqsubset \succeq$, i.e., $\succeq' \sqsubseteq \succeq$ holds but $\succeq \sqsubseteq \succeq'$ (resp. $\succeq \sqsubset \succeq'$) does not.

Since the preference-based argumentation framework is mainly based on the preference relation among arguments, it is worth noticing that the choice of the reasoning attitude is predominant in the output of the argumentation system.

Example 1 *Let $\mathcal{A} = \{A, B, C\}$ be a set of arguments and $V = \{p, q\}$ be the set of values. Let $\mathcal{D}$ be a defeat relation defined by $C \mathcal{D} B$ and $B \mathcal{D} C$. Let $p >> q$ with $arg(p) = \{A\}$ and $arg(q) = \{B\}$. Following the optimistic reasoning the total pre-order satisfying $p >> q$ is $\succeq_o = (\{A, C\}, \{B\})$. We can check that each argument is put in the highest possible rank in $\succeq_o$ s.t. $p >> q$ is satisfied. So we have C attacks B. The grounded extension is composed of A and C. Now following the pessimistic reasoning the total pre-order satisfying $p >> q$ is $\succeq_p = (\{A\}, \{B, C\})$. Here also we can check that each argument is put in the lowest possible rank in $\succeq_p$ s.t. $p >> q$ is satisfied. In this case we have B attacks C and C attacks B. The grounded extension is composed of A only.*

Note that in this example pessimistic reasoning returns less acceptable arguments than optimistic reasoning, however this is not always the case. In addition to the defeat relations given in Example 1 we give $A \mathcal{D} C$ and $C \mathcal{D} A$. Then following the optimistic reasoning the grounded extension is empty while following the pessimistic reasoning the grounded extension is $\{A\}$.
Let us now consider the same example but with the following defeat relations $A \mathcal{D} C$ and $C \mathcal{D} A$ only. Then the grounded extension following the optimistic reasoning is $\{B\}$ while the grounded extension following the pessimistic reasoning is $\{A, B\}$.
Indeed the two kinds of reasoning are incomparable. It is important to notice that the optimistic/pessimistic adjectives refer to the way the arguments are ranked in the total pre-order $\succeq$.

Grounded extension in optimistic reasoning

Algorithms of optimistic reasoning compute the total pre-order $\succeq$ starting from the *best* arguments w.r.t. $\succeq$. Indeed this property makes it possible to compute incrementally the grounded extension when computing this pre-order. Informally this consists in first computing the set of the best arguments w.r.t. $\succeq$. Let us say E_0. Then arguments in E_0 which are not defeated in E_0 belong to the grounded extension. Also all arguments in E_0 defeated only by arguments in $\mathcal{A}\backslash E_0$ belong to the grounded extension. Belong also to the grounded extension arguments in E_0 which are defeated by arguments in E_0 but defended by acceptable arguments, i.e., arguments already put in the grounded extension. Lastly all arguments in $\mathcal{A}\backslash E_0$ defeated by arguments in the current grounded extension will certainly not belong to the grounded extension and can be removed from $\mathcal{A}$. Once $\mathcal{A}$ updated we compute the set of immediately preferred arguments, let's say E_1. At this stage non defeated arguments from E_1 are added to the current grounded extension. Also belong to the grounded extension arguments in E_1 which are defeated by arguments in E_1 but their defeaters are themselves defeated by the current grounded extension. This means that these arguments are defended by the grounded extension. Lastly all arguments in $\mathcal{A}\backslash E_1$ defeated by selected arguments (in the grounded extension) are discarded. This reasoning is repeated until the set of arguments is empty. Algorithm 1 gives a formal description of our procedure to compute progressively the grounded extension. Let

- $Safe(E_l) = \{B : B \in E_l$ s.t. $\nexists B' \in (E_l \cup R)$ with $B'\mathcal{D}B\}$,
- $Acceptable_{\mathcal{GE}}(E_l) = \{B : B \in E_l$ s.t. for each $B' \in (E_l \cup R)$ s.t. $B'\mathcal{D}B, \exists C \in \mathcal{GE}$ s.t. $C\mathcal{D}B'\}$,
- $non\text{-}Safe(\mathcal{A}) = \{B : B \in \mathcal{A}$ s.t. $\exists B' \in \mathcal{GE}$ with $B'\mathcal{D}B\}$.

Algorithm 1: Computing the grounded extension in optimistic reasoning.

Data: $\langle \mathcal{A}, \mathcal{D}, V, >>, arg \rangle$.
Result: The grounded extension.
begin
  $l = 0, \mathcal{GE} = \emptyset, R = \emptyset$;
  while $\mathcal{A} \neq \emptyset$ **do**
    - $E_l = \{B : B \in \mathcal{A}, \forall p_i >> q_i, B \notin arg(q_i)\}$;
    if $E_l = \emptyset$ **then** Stop (inconsistent preferences);
    – $\mathcal{GE} = \mathcal{GE} \cup Safe(E_l)$;
    – $\mathcal{GE} = \mathcal{GE} \cup Acceptable_{\mathcal{GE}}(E_l)$;
    – $\mathcal{A} = \mathcal{A}\backslash E_l$;
    – $\mathcal{A} = \mathcal{A}\backslash\ non\text{-}Safe(\mathcal{A})$;
    – $R = R \cup (E_l \backslash \mathcal{GE})$;
    /** remove satisfied preferences **/;
    – remove $p_i >> q_i$ where $arg(p_i) \cap E_l \neq \emptyset$;
    – $l = l + 1$.
  return $\mathcal{GE}$
end

Example 2 *Tom and Mary discuss with their children about their education. Several arguments are given concerning the plans of the children to spend the day. In an attempt to structure the discussion, the arguments are grouped according to several values they promote, and modeled as follows. Tom and Marry give the following set of preferences* $\{Health >> Unhealth, Education >> Enjoy, Social >> Alone\}$.
Let $\mathcal{A} = \{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ *be a set of arguments where* $arg(Health) = \{A_4, A_5\}$, $arg(Unhealth) = \{A_6, A_7\}$, $arg(Education) = \{A_3, A_5, A_7\}$, $arg(Enjoy) = \{A_2, A_4, A_6\}$, $arg(Social) = \{A_0, A_4\}$ *and* $arg(Alone) = \{A_1, A_5\}$.
Let the following defeat relations $A_6\mathcal{D}A_0$, $A_0\mathcal{D}A_6$, $A_3\mathcal{D}A_4$, $A_3\mathcal{D}A_2$, $A_2\mathcal{D}A_5$, $A_5\mathcal{D}A_2$, $A_4\mathcal{D}A_5$ *and* $A_5\mathcal{D}A_4$. *Figure 1 summarizes defeat relations among the arguments. An arrow from A to B stands for "A defeats B".*

We first put in E_0 *arguments which are not in* $arg(Unhealth)$, $arg(Enjoy)$ *and* $arg(Alone)$. *We get* $E_0 = \{A_0, A_3\}$. *R is the empty set and there is no defeat relation among arguments of* E_0 *so both* A_0 *and* A_3 *are safe. They belong to* $\mathcal{GE}$. *The set* $Acceptable_{\mathcal{GE}}(E_0)$ *returns the empty set since there is no defeat relation in* E_0. *Now we first remove arguments of* E_0 *from* $\mathcal{A}$ *since they have been treated. We get* $\mathcal{A} = \{A_1, A_2, A_4, A_5, A_6, A_7\}$. *Then we remove from* $\mathcal{A}$ *arguments which are defeated by arguments in* $\mathcal{GE}$ *(i.e. which are already accepted). We remove* A_2, A_4 *and* A_6. *So* $\mathcal{A} = \{A_1, A_5, A_7\}$. $R = \emptyset$ *since* $E_0 = \mathcal{GE}$. *Lastly we remove* $Education >> Enjoy$ *and* $Social >> Alone$ *since they are satisfied. We run the second iteration of the algorithm. We have* $E_1 = \{A_1, A_5\}$. A_1 *and* A_5 *are safe so they are added to* $\mathcal{GE}$, *i.e.,* $\mathcal{GE} = \{A_0, A_3, A_1, A_5\}$. $Acceptable_{\mathcal{GE}}(E_1)$ *is empty. We remove* A_1 *and* A_5 *from* $\mathcal{A}$. *There are no non-safe arguments in* $\mathcal{A}$ *and* $R = \emptyset$. *In the third iteration of the algorithm we have* $E_2 = \{A_7\}$. A_7 *is safe so* $\mathcal{GE} = \{A_0, A_3, A_1, A_5, A_7\}$.

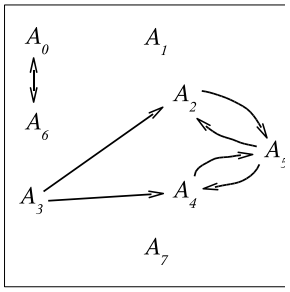


Figure 1: Defeat relations among the arguments.

The role of the set R does not appear in this example however it is important to define such a set to compute incrementally the grounded extension. Let $\mathcal{A} = \{A, B, C, D\}$ be a set of arguments such that $B\mathcal{D}C$, $C\mathcal{D}B$ and $B\mathcal{D}D$. Suppose that the first iteration gives $E_0 = \{A, B, C\}$. So A belongs to the grounded extension while B and C do not (since they attack each other). Following the algorithm we update $\mathcal{A}$ and get $\mathcal{A} = \{D\}$. At this stage it is important to keep B and C in a set, let's say R. The reason is that in the second iteration of the algorithm we should not put D in the grounded extension just because it is not defeated by A. In fact D is attacked by B and not defended by A. This justifies why we consider $E_l \cup R$ when computing $Safe(E_l)$ and $Acceptable_{\mathcal{GE}}(E_l)$.

Let us now first compute the pre-order and then compute the grounded extension. We compute this pre-order from Algorithm 1 by replacing *while* loop by

while $\mathcal{A} \neq \emptyset$ **do**

- $E_l = \{B : B \in \mathcal{A}, \forall p_i >> q_i, B \notin arg(q_i)\}$;
- remove $p_i >> q_i$ where $arg(p_i) \cap E_l \neq \emptyset$.

We have $\succeq_o = (E_0, E_1, E_2)$ where $E_0 = \{A_0, A_3\}$, $E_1 = \{A_1, A_2, A_4, A_5\}$ and $E_2 = \{A_6, A_7\}$.
Let us now compute the grounded extension. Following Definition 8 the attack relations are $A_3\mathcal{R}A_2$, $A_3\mathcal{R}A_4$, $A_4\mathcal{R}A_5$, $A_5\mathcal{R}A_4$, $A_2\mathcal{R}A_5$, $A_5\mathcal{R}A_2$ and $A_0\mathcal{R}A_6$. We first put in the grounded extension arguments which are not attacked, so $\mathcal{GE} = \{A_0, A_1, A_3, A_7\}$. Then we add to $\mathcal{GE}$ arguments which are attacked but defended by arguments in $\mathcal{GE}$. We add A_5. So $\mathcal{GE} = \{A_0, A_1, A_3, A_7, A_5\}$.

The following theorem shows that Algorithm 1 computes the grounded extension.

Theorem 1 *Let $\mathcal{F} = \langle \mathcal{A}, \mathcal{D}, V, >>, arg \rangle$ be a VAF. Algorithm 1 computes the grounded extension of $\mathcal{F}$.*

Grounded extension in pessimistic reasoning

A particularity of pessimistic reasoning is that it computes the total pre-order starting from the lowest ranked arguments in this pre-order. Indeed it is no longer possible to compute progressively the grounded extension. Let us consider our running example. Following the pessimistic reasoning (we will give the formal algorithm later in this section), the worst arguments are A_1, A_2 and A_6. At this stage we can only conclude that A_1 belongs to $\mathcal{GE}$ since it is not defeated. However the status of A_2 and A_6 cannot be determined since they are attacked by A_3 and A_0 respectively. Since higher ranks in $\succeq$ are not computed yet we cannot check whether A_3 and A_0 are attacked or not. The only case where the status of A_2 and A_6 can be determined is when at least one of their defeaters is not defeated. In this case we can conclude that they do not belong to $\mathcal{GE}$. Algorithm 2 gives the total pre-order following the pessimistic reasoning. Each argument is put in the lowest possible rank in the computed pre-order.

Example 3 *(cont'd)*
We put in E_0 arguments which do not appear in any $arg(Health)$, $arg(Education)$ and $arg(Social)$. We get $E_0 = \{A_1, A_2, A_6\}$. We remove all preferences $p_i >> q_i$ s.t. $arg(q_i) \cap E_0 \neq \emptyset$. All preferences are removed. Then $E_1 = \{A_0, A_3, A_4, A_5, A_7\}$.
So we have $\succeq_p = (\{A_0, A_3, A_4, A_5, A_7\}, \{A_1, A_2, A_6\})$.

Algorithm 2: Pessimistic reasoning.

Data: $\langle \mathcal{A}, \mathcal{D}, V, >>, arg \rangle$.
Result: A total pre-order $\succeq_p$ on $\mathcal{A}$.

begin
  $l = 0$;
  while $\mathcal{A} \neq \emptyset$ **do**
    $E_l = \{B : B \in \mathcal{A}, \forall p_i >> q_i, B \notin arg(p_i)\}$;
    if $E_l = \emptyset$ **then** Stop (inconsistent preferences);
    – Remove from $\mathcal{A}$ elements of E_l;
    /** remove satisfied preferences **/
    – Remove $p_i >> q_i$ where $arg(q_i) \cap E_l \neq \emptyset$;
    – $l = l + 1$.
  return $(E'_1, \cdots, E'_{l-1})$ *s.t.* $\forall 1 \leq h \leq l, E'_h = E_{l-h-1}$
end

In this example we get the same grounded extension as in the optimistic reasoning. However if we add for example the defeat relations $A_3\mathcal{D}A_7$ and $A_7\mathcal{D}A_3$ then the grounded extension following the optimistic reasoning is $\{A_0, A_1, A_3, A_5\}$ while following the pessimistic reasoning the grounded extension is $\{A_0, A_1\}$.

Related Work

The preference-based argumentation theory introduced in this paper integrates several existing approaches, most notable the preference based framework of Amgoud and Cayrol (Amgoud & Cayrol 2002), and the value based argumentation theory of Bench-Capon (Bench-Capon 2003). However, there are also substantial conceptual and technical distinctions.

Maybe the main conceptual distinction is that the above authors present their argumentation theory as an extension of Dung's framework, which has the technical consequence that they also define new notions of, for example, defence and acceptance. We, in contrast, consider our preference-based argumentation theory as an alternative representation of Dung's theory, that is, as a kind of front end to it, which has the technical consequence that we do not have to introduce such notions. Reductions of the other preference-based argumentation theories to Dung's theory may be derived from some of the results presented by these authors.

Another conceptual distinction is that in our theory, there seems to be a higher impact of preference reasoning in argumentation. The preference ordering on arguments is not given, but has to be derived from a more abstract preference specification. Technically, this leads to our use of non-monotonic preference reasoning to derive a Dung-style attack relation from a preference specification together with a defeat relation. None of the existing approaches studies the use of non-monotonic reasoning techniques to reason with the preferences. Another conceptual distinction with

the work of Bench-Capon is that he, following Perelman, is concerned with an audience.

Concerning the extensive work of Amgoud and colleagues on preference-based argumentation theory, our preference based argumentation theory seems closest to the argumentation framework based on contextual preferences of Amgoud, Parsons and Perrussel (Amgoud, Parsons, & Perrussel 2000). A context may be an agent, a criterion, a viewpoint, *etc.*, and they are ordered. For example, in law earlier arguments are preferred to later ones, arguments of a higher authority are preferred to arguments of a lower authority, more specific arguments are preferred over more general arguments, and these three rules are ordered themselves too. However our approach is more general since we compare sets of arguments instead of single arguments as it is the case in their approach. Bench-Capon (Bench-Capon 2003) develops a value-based argumentation framework, where arguments promote some value. No ordering is required among arguments promoting the same value. If a value V is prioritized over another value W then this is interpreted as "each argument promoting the value V is preferred to all arguments promoting the value W". In our framework we can add such preferences, or encode them as $p_i >> q_j$ where p_i is an argument in favor of V and q_j is an argument in favor of W. Note that in our example there is no ordering which satisfies these strong preferences.

Specificity principle we used in this paper has been also used in many other works (Prakken & Sartor 1997; Poole 1985; Simari & Loui 1992; Stolzenburg *et al.* 2003)[1] however in that works preference relation over arguments is defined on the basis of specificity of their internal structure. In fact arguments are built from default and strict knowledge. Then an argument is preferred to another if its internal structure is more specific. In our work specificity concerns *abstract* arguments without referring to their internal structure.

There are numerous works on non-monotonic logic and in particular the non-monotonic logic of preference which is related to the work in this paper, and which can be used to further generalize the reasoning about preferences in argumentation. Interestingly, as argumentation theory is itself a framework of non-monotonic reasoning, due to our non-monotonic reasoning about preferences two kinds of non-monotonicity seems to be present in our system; we leave a further analysis of this phenomena for further research.

Summary

To promote a higher impact of preference reasoning in argumentation, we introduce a novel preference-based argumentation theory. Starting with a set of arguments, a defeat relation, and an ordered set of values, we use the ordered values to compute a preference relation over arguments, and we combine this preference relation with the defeat relation to compute Dung's attack relation. Then we use any of Dung's semantics to define the acceptable set of arguments. In contrast to most other approaches, our approach to reason about preferences in argumentation does not refer to the internal structure of the arguments.

The problem of reducing ordered values to a preference relation comes down to reducing a preference relation over sets of arguments to a preference relation over single arguments. To reason about ordered values and to compute the preference relation over arguments, we are inspired by insights from the non-monotonic logic of preference known as minimal specificity, System Z, gravitation to normality, and otherwise, and we use both so-called optimistic and pessimistic ways to define the preference relation.

For the combined approach, we introduce an algorithm for Dung's grounded semantics. It shows that the computation of the set of acceptable arguments can be combined with the optimistic reasoning to incrementally define the set of acceptable arguments, because in this construction for each equivalence class we can deduce which arguments are not attacked by other arguments. This property does not hold for pessimistic reasoning.

In future work, we study other ways to use reasoning about preferences in argumentation theory. For example, Bochman (2005) develops a generalization of Dung's theory, called collective argumentation, where the attack relation is defined over sets of arguments instead of single arguments. It seems natural to develop a unified framework where both attack and preference relations are defined over sets of arguments. Another future work is to study the reinforcement among different arguments promoting the same value as advocated in (Bench-Capon 2003).

[1] Note that Poole (Poole 1985) uses specificity of arguments without studying interaction among arguments.

References

Amgoud, L., and Cayrol, C. 2002. A reasoning model based on the production of acceptable arguments. *AMAI Journal* 34:197–216.

Amgoud, L.; Cayrol, C.; and LeBerre, D. 1996. Comparing arguments using preference orderings for argument-based reasoning. In *8th International Conf. on Tools with Artificial Intelligence*, 400–403.

Amgoud, L.; Parsons, S.; and Perrussel, L. 2000. An argumentation framework based on contextual preferences. Technical report, Department of Electronic Engineering, Queen Mary and Westfield College.

Atkinson, K.; Bench-Capon, T.; and McBurney, P. 2005. Persuasive Political Argument. In *Computational Models of Natural Argument*, 44–51.

Bench-Capon, T. 2003. Persuasion in practical argument using value based argumentation framework. *Journal of Logic and Computation* 13(3):429–448.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2002. Bipolar possibilistic representations. In *18th International Conference on Uncertainty in Artifcial Intelligence (UAI'02)*, 45–52.

Bochman, A. 2005. Propositional Argumentation and

Causal Reasoning. In *11th Int. Joint Conf. on Artificial Intelligence*, 388–393.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77:321–357.

Kaci, S., and van der Torre, L. 2005. Algorithms for a Nonmonotonic Logic of Preferences. In *8th Eur. Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 281–292.

Kaci, S.; van der Torre, L.; and Weydert, E. 2006. Acyclic argumentation: Attack = conflict + preference. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, to appear.

Pearl, J. 1990. System z: A natural ordering of defaults with tractable applications to default reasoning. In *3rd Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'90)*, 121–135.

Poole, D. L. 1985. On the comparison of theories: Preferring the most specific explanation. In *Proceedings of the 9th IJCAI*, 144–147. IJCAI.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorties. *Journal of Applied Non-Classical Logics* 7:25–75.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53:125–157.

Stolzenburg, F.; García, A. J.; Chesñevar, C. I.; and Simari, G. R. 2003. Computing generalized specificity. *Journal of Aplied Non-Classical Logics* 13(1):87–113.

3 NMR Systems and Applications

In recent years, a number of systems implementing non-monotonic reasoning, or making extensive use of nonmonotonic reasoning approaches, have emerged. These systems have ranged from implementations of systems in traditional nonmonotonic reasoning areas (best exemplified by answer set program implementations) to those in less traditional areas (including belief change and causality). Areas of application have similarly ranged from implementations of systems in traditional areas (best exemplified by planning) to less traditional or emerging areas (including bioinformatics, configuration, and the semantic web). Given the increasing performance of computer hardware along with advances in algorithm design, the performance of existing systems is already sufficient to enable industrial applications of non-monotonic reasoning.

This special session was intended to attract researchers interested in systems and applications of non-monotonic reasoning. Seven papers were presented at the session. Grigoris Antoniou and Antonis Bikakis describe a system, DR-Prolog, for defeasible reasoning on the web. Martin Brain, Tom Crick, Marina De Vos and John Fitch describe a large-scale implementation using answer set programming to generate optimal machine code for simple, acyclic functions, with encouraging results. In the paper by James Delgrande, Daphne H. Liu, Torsten Schaub, and Sven Thiele, an implementation for a general consistency-based approach to belief change, including revision, contraction and merging with integrity constraints is presented. Susanne Grell, Torsten Schaub, and Joachim Selbig propose a new action language for modelling biological networks, building on earlier work by Baral et al. and providing a translation into answer set programs. Efstratios Kontopoulos, Nick Bassiliades, Grigoris Antoniou present in their paper a system for non-monotonic reasoning on the Semantic Web called VDR-Device, which is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules. Frederick Maier and Donald Nute relating Defeasible Logic to the well-founded semantics for normal logic programs. Oliver Ray and Antonis Kakas's paper presents a new system, called ProLogICA, for abductive logic programming in the presence of negation as failure and integrity constraints.

Session chairs

James Delgrande
(jim@cs.sfu.ca)

Torsten Schaub
(torsten@cs.uni-potsdam.de)

Program committee

Chitta Baral
(chitta@asu.edu)

Dirk Vermeir
(dvermeir@tinf.vub.ac.be)

Gerald Pfeifer
(gerald@pfeifer.com)

Gianluigi Greco
(ggreco@mat.unical.it)

Joohyung Lee
(joolee@asu.edu)

Leopoldo Bertossi
(bertossi@scs.carleton.ca)

Paolo Liberatore
(liberato@dis.uniroma1.it)

Pascal Nicolas
(pn@info.univ-angers.fr)

Odile Papini
(papini@univ-tln.fr)

Trao Can Son
(tson@cs.nmsu.edu)

Yannis Dimopoulos
(yannis@cs.ucy.ac.cy)

Yan Zhang
(yan@cit.uws.edu.au)

3.1 DR-Prolog: A System for Reasoning with Rules and Ontologies on the Semantic Web

DR-Prolog: A System for Reasoning with Rules and Ontologies on the Semantic Web

Grigoris Antoniou and Antonis Bikakis

Institute of Computer Science, FO.R.T.H
Vassilika Vouton, P.O. Box 1385, GR 71110, Heraklion, Greece
{antoniou,bikakis}@ics.forth.gr

Abstract

Defeasible reasoning is a rule-based approach for efficient reasoning with incomplete and inconsistent information. Such reasoning is, among others, useful for ontology integration, where conflicting information arises naturally; and for the modeling of business rules and policies, where rules with exceptions are often used. This paper describes these scenarios in more detail, and reports on the implementation of a system for defeasible reasoning on the Web. The system (a) is syntactically compatible with RuleML; (b) features strict and defeasible rules, priorities and two kinds of negation; (c) is based on a translation to logic programming with declarative semantics; (d) is flexible and adaptable to different intuitions within defeasible reasoning; and (e) can reason with rules, RDF, RDF Schema and (parts of) OWL ontologies.

Introduction

The development of the Semantic Web (Berners Lee *et al.*, 2001) proceeds in layers, each layer being on top of other layers. At present, the highest layer that has reached sufficient maturity is the ontology layer in the form of the description logic based languages of DAML+OIL (Connolly *et al.*, 2001) and OWL (Dean and Schreiber, 2004).

The next step in the development of the Semantic Web will be the logic and proof layers that will offer enhanced representation and reasoning capabilities. *Rule systems* appear to lie in the mainstream of such activities. Moreover, rule systems can also be utilized in ontology languages. So, in general rule systems can play a twofold role in the Semantic Web initiative: (a) they can serve as extensions of, or alternatives to, description logic based ontology languages; and (b) they can be used to develop declarative systems on top of (using) ontologies. Reasons why rule systems are expected to play a key role in the further development of the Semantic Web include the following:

- Seen as subsets of predicate logic, monotonic rule systems (Horn logic) and description logics are orthogonal; thus they provide additional expressive power to ontology languages.
- Efficient reasoning support exists to support rule languages.
- Rules are well known in practice, and are reasonably well integrated in mainstream information technology.

Possible interactions between description logics and monotonic rule systems were studied in (Grosof *et al.*, 2003). Based on that work and on previous work on hybrid reasoning (Levy and Rousset, 1998) it appears that the best one can do at present is to take the intersection of the expressive power of Horn logic and description logics; one way to view this intersection is the Horn-definable subset of OWL.

This paper is devoted to a different problem, namely *conflicts among rules*. Here we just mention the main sources of such conflicts, which are further expanded in the next section. At the ontology layer:

- Default inheritance within ontologies
- Ontology merging

And at the logic and reasoning layers:

- Rules with exceptions as a natural representation of business rules
- Reasoning with incomplete information

Defeasible reasoning is a simple rule-based approach to reasoning with incomplete and inconsistent information. It can represent facts, rules, and priorities among rules. This reasoning family comprises defeasible logics (Nute, 1994; Antoniou *et al.*, 2001) and Courteous Logic Programs (Grosof 1997). The main advantage of this approach is the combination of two desirable features: enhanced representational capabilities allowing one to reason with incomplete and contradictory information, coupled with low computational complexity compared to mainstream nonmonotonic reasoning.

In this paper we report on the implementation of a defeasible reasoning system for reasoning on the Web. Its main characteristics are the following:

- Its user interface is compatible with RuleML (RuleML), the main standardization effort for rules on the Semantic Web.
- It is based on Prolog. The core of the system consists of a well-studied translation (Antoniou *et. al.*, 2001) of defeasible knowledge into logic programs under Well-

Founded Semantics (van Gelder *et al.*, 1991). This declarative translation distinguishes our work from other implementations (Grosof *et al.*, 2002; Maher *et al.*, 2001).

- The main focus is on flexibility. Strict and defeasible rules and priorities are part of the interface and the implementation. Also, a number of variants were implemented (ambiguity blocking, ambiguity propagating, conflicting literals; see below for further details).
- The system can reason with rules and ontological knowledge written in RDF Schema (RDFS) or OWL.

As a result of the above, DR-Prolog is a powerful declarative system supporting:

- rules, facts and ontologies
- all major Semantic Web standards: RDF, RDFS, OWL, RuleML
- monotonic and nonmonotonic rules, open and closed world assumption, reasoning with inconsistencies.

The paper is organized as follows. The next section describes the main motivations for conflicting rules on the Semantic Web. The third section describes the basic ideas of defeasible reasoning, and the forth one describes the translation of defeasible logic, and of RDF, RDFS and (parts of) OWL into logic programs. The fifth section reports on the implemented system. The sixth section discusses related work, and the last section concludes with a summary and some ideas for future work.

Motivation for Nonmonotonic Rules on the Semantic Web

We believe that we have to distinguish between two types of knowledge on the Semantic Web. One is static knowledge, such as factual and ontological knowledge which contains general truths that do not change often. And the other is dynamic knowledge, such as business rules, security policies etc. that change often according to business and strategic needs. The first type of knowledge requires monotonic reasoning based on an open world assumption to guarantee correct propagation of truths. But for dynamic knowledge flexible, context-dependent and inconsistency tolerant nonmonotonic reasoning is more appropriate for drawing practical conclusions.

Obviously, a combination of both types of knowledge is required for practical systems. Defeasible logic, as described in the next section, supports both kinds of knowledge. Before presenting its technical details, we motivate the use of nonmonotonic rules in more detail.

Reasoning with Incomplete Information*:* Antoniou and Arief (2002) describe a scenario where business rules have to deal with incomplete information: in the absence of certain information some assumptions have to be made which lead to conclusions that are not supported by classical predicate logic. In many applications on the Web such assumptions must be made because other players may not be able (e.g. due to communication problems) or willing (e.g. because of privacy or security concerns) to provide information. This is the classical case for the use of nonmonotonic knowledge representation and reasoning (Marek and Truszczynski, 1993).

Rules with Exceptions*:* Rules with exceptions are a natural representation for policies and business rules (Antoniou *et. al*, 1999). And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies (Ashri *et al.*, 2004; Li *et al*., 2003), business rules (Antoniou and Arief 2002), personalization, brokering, bargaining, and automated agent negotiations (Governatori *et al.*, 2001).

Default Inheritance in Ontologies*:* Default inheritance is a well-known feature of certain knowledge representation formalisms. Thus it may play a role in ontology languages, which currently do not support this feature. Grosof and Poon (2003) present some ideas for possible uses of default inheritance in ontologies. A natural way of representing default inheritance is rules with exceptions, plus priority information. Thus, nonmonotonic rule systems can be utilized in ontology languages.

Ontology Merging*:* When ontologies from different authors and/or sources are merged, contradictions arise naturally. Predicate logic based formalisms, including all current Semantic Web languages, cannot cope with inconsistencies.

If rule-based ontology languages are used and if rules are interpreted as defeasible (that is, they may be prevented from being applied even if they can fire) then we arrive at nonmonotonic rule systems. A skeptical approach, as adopted by defeasible reasoning, is sensible because it does not allow for contradictory conclusions to be drawn. Moreover, priorities may be used to resolve some conflicts among rules, based on knowledge about the reliability of sources or on user input. Thus, nonmonotonic rule systems can support ontology integration.

Defeasible Logics

Basic Characteristics

The root of defeasible logics lies on research in knowledge representation, and in particular on inheritance networks. Defeasible logics can be seen as inheritance networks expressed in a logical rules language. In fact, they are the first nonmonotonic reasoning approach designed from its beginning to be implementable.

Being nonmonotonic, defeasible logics deal with potential conflicts (inconsistencies) among knowledge items. Thus they contain classical negation, contrary to usual logic programming systems. They can also deal with negation as failure (NAF), the other type of negation typical of nonmonotonic logic programming systems; in fact, Wagner (2003) argues that the Semantic Web requires

both types of negation. In defeasible logics, often it is assumed that NAF is not included in the object language. However, as Antoniou *et al.* (2000a) show, it can be easily simulated when necessary. Thus, we may use NAF in the object language and transform the original knowledge to logical rules without NAF exhibiting the same behavior.
Conflicts among rules are indicated by a conflict between their conclusions. These conflicts are of local nature. The simpler case is that one conclusion is the negation of the other. The more complex case arises when the conclusions have been declared to be mutually exclusive, a very useful representation feature in practical applications.
Defeasible logics are skeptical in the sense that conflicting rules do not fire. Thus consistency of drawn conclusions is preserved.
Priorities on rules may be used to resolve some conflicts among rules. Priority information is often found in practice, and constitutes another representational feature of defeasible logics.
The logics take a pragmatic view and have low computational complexity. This is, among others, achieved through the absence of disjunction and the local nature of priorities: only priorities between conflicting rules are used, as opposed to systems of formal argumentation where often more complex kinds of priorities (e.g. comparing the strength of reasoning chains) are incorporated.
Generally speaking, defeasible logics are closely related to Courteous Logic Programs (Grosof, 1997); the latter were developed much later than defeasible logics. DLs have the following advantages:

- They have more general semantic capabilities, e.g. in terms of loops, ambiguity propagation etc.
- They have been studied much more deeply, with strong results in terms of proof theory (Antoniou *et al.*, 2001), semantics (Maher, 2002) and computational complexity (Maher, 2001). As a consequence, its translation into logic programs, a cornerstone of DR-Prolog, has also been studied thoroughly (Maher *et al.*, 2001; Antoniou and Maher, 2002).

However, Courteous Logic Programs have also had some advantages:

- They were the first to adopt the idea of mutually exclusive literals, an idea incorporated in DR-Prolog.
- They allow access to procedural attachments, something we have chosen not to follow in our work so far.

Syntax

A *defeasible theory D* is a triple (*F*,*R*,>) where *F* is a finite set of facts, *R* a finite set of rules, and > a superiority relation on R. In expressing the proof theory we consider only propositional rules. Rules containing free variables are interpreted as the set of their variable-free instances.
There are two kinds of rules (fuller versions of defeasible logics include also defeaters): *Strict rules* are denoted by

$A \rightarrow p$,

and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "Professors are faculty members". Written formally:

$professor(X) \rightarrow faculty(X)$.

Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.
Defeasible rules are denoted by

$A \Rightarrow p$,

and can be defeated by contrary evidence. An example of such a rule is

$faculty(X) \Rightarrow tenured(X)$

which reads as follows: "Professors are typically tenured".
A *superiority relation* on R is an acyclic relation > on R (that is, the transitive closure of > is irreflexive). When

$r_1 > r_2$,

then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

$r: professor(X) \Rightarrow tenured(X)$
$r': visiting(X) \Rightarrow \neg tenured(X)$

which contradict one another: no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation > with

$r' > r$,

then we can indeed conclude that a visiting professor cannot be tenured.
A formal definition of the proof theory is found in (Antoniou *et al.*, 2001).

Simulation of Negation As Failure in the Object Language

We follow a technique based on auxiliary predicates first presented in (Antoniou *et al.*, 2000a), but which is often used in logic programming. According to this technique, a defeasible theory with NAF can be modularly transformed into an equivalent one without NAF. Every rule

$r: L_1,\ldots,L_n,\ \sim M_1,\ldots,\ \sim M_k \Rightarrow L$

where $L_1,\ldots,L_n,\ M_1,\ldots,M_k$ are atoms and $\sim M_i$ denotes the weak negation of M_i, is replaced by the rules:

$r: L_1,\ldots,L_n,\ neg(M_1),\ldots,neg(M_k) \Rightarrow L$
$\Rightarrow neg(M_1)$
…
$\Rightarrow neg(M_k)$
$M_1 \Rightarrow \neg neg(M_1)$
…
$M_k \Rightarrow \neg neg(M_k)$

where $neg(M1),\ldots,neg(Mk)$ are new auxiliary atoms and $\neg neg(M_i)$ denotes the strong negation of M_i. If we restrict attention to the original language, the set of conclusions remains the same.

Ambiguity Blocking and Ambiguity Propagating Behavior

A literal is *ambiguous* if there is a chain of reasoning that supports a conclusion that *p* is true, another that supports that ¬*p* (where ¬*p* denotes strong negation of *p*) is true, and the superiority relation does not resolve this conflict. We can illustrate the concept of ambiguity propagation through the following example.

```
r1: quaker(X) ⇒ pacifist(X)
r2: republican(X) ⇒ ¬pacifist(X)
r3: pacifist(X) ⇒ ¬hasGun(X)
r4: livesInChicago(X) ⇒ hasGun(X)
quaker(a)
republican(a)
livesInChicago(a)
r3 > r4
```

Here `pacifist(a)` is ambiguous. The question is whether this ambiguity should be propagated to the dependent literal `hasGun(a)`. In one defeasible logic variant it is detected that rule r_3 cannot fire, so rule r_4 is unopposed and gives the defeasible conclusion `hasGun(a)`. This behavior is called *ambiguity blocking*, since the ambiguity of `pacifist(a)` has been used to block r_3 and resulted in the unambiguous conclusion `hasGun(a)`.

On the other hand, in the ambiguity propagation variant, although rule r_3 cannot lead to the conclusion `hasGun(a)` (as `pacifist(a)` is not provable), it opposes rule r_4 and the conclusion `hasGun(a)` cannot also be drawn.

This question has been extensively studied in artificial intelligence, and in particular in the theory of inheritance networks. A preference for ambiguity blocking or ambiguity propagating behavior is one of the properties of nonmonotonic inheritance nets over which intuitions can clash. Ambiguity propagation results in fewer conclusions being drawn, which might make it preferable when the cost of an incorrect conclusion is high. For these reasons an ambiguity propagating variant of DL is of interest.

Conflicting Literals

Usually in Defeasible Logics only conflicts among rules with complementary heads are detected and used; all rules with head *L* are considered as *supportive* of *L*, and all rules with head ¬*L* as *conflicting*. However, in applications often literals are considered to be conflicting, and at most one of a certain set should be derived. For example, the risk an investor is willing to accept may be classified in one of the categories low, medium, and high. The way to solve this problem is to use a constraint rule of the form

```
conflict :: low, medium, high
```

Now if we try to derive the conclusion `high`, the conflicting rules are not just those with head ¬`high`, but also those with head `low` and `medium`. Similarly, if we are trying to prove ¬`high`, the supportive rules include those with head `low` or `medium`.

In general, given a `conflict::L,M`, we augment the defeasible theory by:

```
ri: q1,q2,…,qn → ¬L
  for all rules ri: q1,q2,…,qn → M
ri: q1,q2,…,qn → ¬M
  for all rules ri: q1,q2,…,qn → L
ri: q1,q2,…,qn ⇒ ¬L
  for all rules ri: q1,q2,…,qn⇒ M
ri: q1,q2,…,qn ⇒ ¬M
  for all rules ri: q1,q2,…,qn ⇒ L
```

The superiority relation among the rules of the defeasible theory is propagated to the "new" rules.

Translation into Logic Programs

Translation of Defeasible Theories

The translation of a defeasible theory *D* into a logic program *P(D)* has a certain goal: to show that

p is defeasibly provable in *D* ⇔
p is included in the Well-Founded Model of *P(D)*

Two different translations have so far been proposed, sharing the same basic structure:

The translation of (Antoniou *et al.*, 2000b; Maher *et al.*, 2001) where a meta-program was used.

The translation of (Antoniou and Maher, 2002), which makes use of control literals.

It is an open question which is better in terms of computational efficiency, although we conjecture that for large theories the meta-program approach is better, since in the other approach a large number of concrete program clauses is generated. Therefore, we have adopted this approach in our implementation.

Translation of Ambiguity Blocking Behavior. The metaprogram which corresponds to the ambiguity blocking behavior of the defeasible theories consists of the following program clauses:

The first three clauses define the class of rules used in a defeasible theory.

```
supportive_rule(Name,Head,Body):-
  strict(Name,Head,Body).
supportive_rule(Name,Head,Body):-
  defeasible(Name,Head,Body).
rule(Name,Head,Body):-

  supportive_rule (Name,Head,Body).
```

The following clauses define the definite provability: a literal is definitely provable if it is a fact or is supported by a strict rule, the premises of which are definitely provable.

```
definitely(X):- fact(X).
definitely(X):-strict(R,X,L),
  definitely_provable(L).
definitely_provable([]).
definitely_provable(X):- definitely(X).
definitely_provable([X1|X2]):-
  definitely_provable(X1),
  definitely_provable(X2).
```

The next clauses define the defeasible provability: a literal is defeasibly provable, either if it is definitely provable, or if its complementary is not definitely provable, and it is supported by a defeasible rule, the premises of which are defeasibly provable, and which is not overruled. The sk_not operator, which we use as the negation operator in the following clauses, is provided by XSB (the logic programming system that stands in the core of DR-Prolog), and allows for correct execution of programs according to the well-founded semantics.

```
defeasibly(X):- definitely(X).
defeasibly(X):- negation(X,X1),
  supportive_rule(R,X,L),
  defeasibly_provable(L),
  sk_not(definitely(X1)),
  sk_not(overruled(R,X)).
defeasibly_provable([]).
defeasibly_provable(X):- defeasibly(X).
defeasibly_provable([X1|X2]):-
  defeasibly_provable(X1),
  defeasibly_provable(X2).
```

The next clause defines that a rule is overruled when there is a conflicting rule, the premises of which are defeasible provable, and which is not defeated.

```
overruled(R,X):- negation(X,X1),
  supportive_rule(S,X1,U),
defeasibly_provable(U),
  sk_not(defeated(S,X1)).
```

The next clause defines that a rule is defeated when there is a superior conflict rule, the premises of which are defeasibly provable. The last two clauses are used to define the negation of a literal.

```
defeated(S,X):-sup(T,S), negation(X,X1),
  supportive_rule(T,X1,V),
  defeasibly_provable(V).
negation(~(X),X):- !.
negation(X,~(X)).
```

For a defeasible theory $D = (F,R,>)$, where F is the set of the facts, R is the set of the rules, and $>$ is the set of the superiority relations between the rules of the theory, we add facts according to the following guidelines:

```
fact(p).
```
  for each $p \in F$
```
strict(r_i,p,[q_1,…,q_n]).
```
  for each rule $r: q_1,q_2,\ldots,q_n \rightarrow p \in R$
```
defeasible(r_i,p,[q_1,…,q_n]).
```
  for each rule $r: q_1,q_2,\ldots,q_n \Rightarrow p \in R$
```
sup(r,s).
```
  for each pair of rules such that $r>s$

Translation of Ambiguity Propagating Behavior. In order to support the ambiguity propagation behavior of a defeasible theory, we only have to modify the program clauses which define when a rule is overruled. In particular, in this variant a rule is overruled when there is a conflicting rule, the premises of which are supported, and which is not defeated.

```
overruled(R,X):- negation(X,X1),
  supportive_rule(S,X1,U),
  supported_list(U),
  sk_not(defeated(S,X1)).
```

The next clauses define that a literal is supported, either if it is definitely provable, or if there is a supportive rule, the premises of which are supported, and which is not defeated.

```
supported(X):- definitely(X).
supported(X):-supportive_rule(R,X,L),
  supported_list(L),
  sk_not(defeated(R,X)).
supported_list([]).
supported_list(X):- supported(X).
supported_list([X1|X2]):-
  supported_list(X1),
  supported_list(X2).
```

Translation of RDF(S) and parts of OWL ontologies

In order to support reasoning with RDF/S and OWL ontologies, we translate RDF data into logical facts, and RDFS and OWL statements into logical facts and rules.
For RDF data, the SWI-Prolog RDF parser (SWI) is used to transform it into an intermediate format, representing triples as

```
rdf(Subject, Predicate, Object).
```

Some additional processing
(i) transforms the facts further into the format

```
Predicate(Subject, Object);
```

(ii) cuts the namespaces and the “comment” elements of the RDF files, except for resources which refer to the RDF or OWL Schema, for which namespace information is retained.
In addition, for processing RDF Schema information, the following rules capturing the semantics of RDF Schema constructs are created:

```
a: C(X):- rdf:type(X,C).
b: C(X):- rdfs:subClassOf(Sc,C),Sc(X).
c: P(X,Y):- rdfs:subPropertyOf(Sp,P),
              Sp(X,Y).
d: D(X):- rdfs:domain(P,D),P(X,Z).
e: R(Z):- rdfs:range(P,R),P(X,Z).
```

Parts of OWL ontologies can also be translated using logical rules, which capture the semantics of some of the OWL constructs.

Equality

```
o1: D(X):- C(X),owl:equivalentClass(C,D).
o2: C(X):- D(X),owl:equivalentClass(C,D).
o3: P(X,Y):- Q(X,Y),
      owl:equivalentProperty(P,Q).
o4: Q(X,Y):- P(X,Y),
      owl:equivalentProperty(P,Q).
o5: owl:equivalentClass(X,Y):-
      rdfs:subClassOf(X,Y),
      rdfs:subClassOf(Y,X).
o6 :owl:equivalentProperty(X,Y):-
      rdfs:subPropertyOf(X,Y),
      rdfs:subPropertyOf(Y,X)
o7 : C(X):- C(Y),
       owl:sameIndividualAs(X,Y).
o8 : P(X,Z):- P(X,Y),
      owl:sameIndividualAs(Y,Z).
o9 : P(Z,Y):- P(X,Y),
      owl:sameIndividualAs(X,Z).
o10: owl:sameIndividualAs(X,Y):-
      owl:sameIndividualAs(Y,X).
o11: owl:sameIndividualAs(X,Z):-
        owl:sameIndividualAs(X,Y),
        owl:sameIndividualAs(Y,Z).
o12: owl:sameAs(X,Y):-
        owl:equivalentClass(X,Y).
o13: owl:sameAs(X,Y):-
        owl:equivalentProperty(X,Y).
o14: owl:sameAs(X,Y):-
        owl:sameIndividualAs(X,Y).
```

Property Characteristics

```
o15: P(X,Z):- P(X,Y), P(Y,Z),
        rdf:type(P,owl:TransitiveProperty).
o16: P(X,Y):- P(Y,X),
        rdf:type(P,owl:SymmetricProperty).
o17: P(X,Y):- Q(Y,X),owl:Inverseof(P,Q).
o18: Q(X,Y):- P(Y,X),owl:Inverseof(P,Q).
o19: owl:sameIndividualAs(X,Y):-
      P(A,X),P(A,Y),
      rdf:type(P,owl:FunctionalProperty).
o20: owl:sameIndividualAs(X,Y):-
      P(X,A),P(Y,A),
rdf:type(P,owl:InverseFunctionalProperty)
```

Property Restrictions

```
o21: D(Y):- C(X),P(X,Y),
      rdfs:subClassOf(C,R),
      rdf:type(R,owl:Restriction),
      owl:onProperty(R,P),
      owl:allValuesFrom(R,D),
      rdf:type(D,owl:Class).
o22: C(X):- P(X,V),rdfs:subClassOf(C,R),
      rdf:type(R,owl:Restriction),
    owl:onProperty(R,P),owl:hasValue(R,V).
o23: P(X,V):- C(X),rdfs:subClassOf(C,R),
      rdf:type(R,owl:Restriction),
    owl:onProperty(R,P),owl:hasValue(R,V).
```

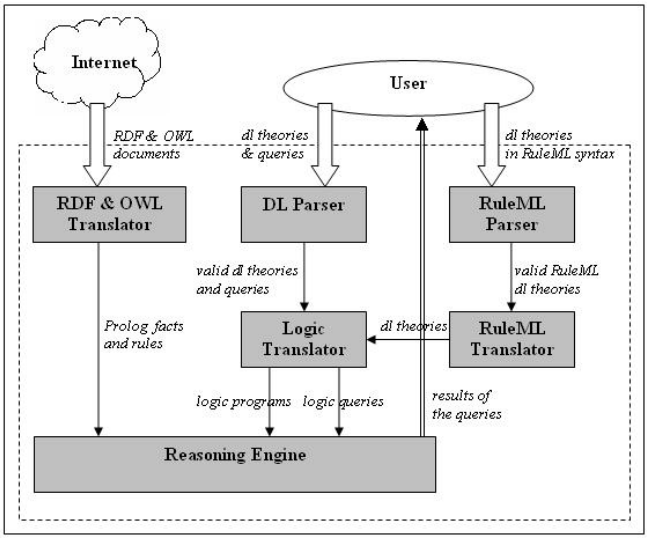


Figure 1: The overall architecture of DR-Prolog

Collections

```
o24: D(X):- C1(X), C2(X),
      owl:IntersectionOf(D,Collect),
      rdf:type(Collect,Collection),
      memberOf(C1,Collect),
      memberOf(C2,Collect).
o25: C1(X):- D(X),
      owl:IntersectionOf(D,Collect),
      rdf:type(Collect,Collection),
      memberOf(C1,Collect),
      memberOf(C2,Collect).
o26: C2(X):- D(X),
      owl:IntersectionOf(D,Collect),
      rdf:type(Collect,Collection),
      memberOf(C1,Collect),
      memberOf(C2,Collect).
o27: C(X):- owl:oneOf(C,Collect),
      rdf:type(Collect,Collection),
      memberOf(X,Collect).
```

Implementation

DR-Prolog, in accordance with the general philosophy of logic programming, is designed to answer queries. In fact, there are two kinds of queries, depending on which strength of proof we are interested in: definite or defeasible provability.

In Figure 1 we present the overall architecture of our system. The system works in the following way: The user imports defeasible theories, either using the syntax of defeasible logic, or in the RuleML syntax, that we describe below in this section. The former theories are checked by the DL Parser, and if they are syntactically correct, they are passed to the Logic Translator, which translates them

into logic programs. The RuleML defeasible theories are checked by the RuleML Parser and translated into defeasible theories, which are also passed to the Logic Translator and transformed into logic programs. The Reasoning Engine compiles the logic programs and the metaprogram which corresponds to the user's choice of the defeasible theory variants (ambiguity blocking / propagating), and evaluates the answers to the user's queries. The logic programming system that we use as the Reasoning Engine is XSB. The advantages of this system are two: (a) it supports the well-founded semantics of logic programs through the use of tabled predicates, and its *sk_not* negation operator; and (b) it offers an easy and efficient way to communicate with the other parts of the system. The RDF&OWL Translator is used to translate the RDF/S and OWL information into logical facts and rules, which can be processed by the rules, provided by the user.
The DTD that we have developed to represent defeasible theories in XML format, is in fact an extension of the RuleML DTDs (RuleML). The elements that we add / modify to support the defeasible theories are:

- The "rulebase" root element which uses strict and defeasible rules, fact assertions and superiority relations.
- The "imp" element, which consists of a "_head" and a "_body" element, accepts a "name" attribute, and refers to the strict rules.
- The "def" element, which consists of a "_head" and a "_body" element, accepts a "name" attribute, and refers to the defeasible rules.
- The "superiority" empty element, which accepts the name of two rules as its attributes ("sup" & "inf"), and refers to the superiority relation between these two rules.

Below, we present the modified DTD:

```
<!ELEMENT rulebase ((imp|def|fact|greater)*)>
<!ELEMENT imp ((head, body) | (body, head))>
<!ATTLIST imp
      name ID #IMPLIED>
<!ELEMENT def ((head, body) | (body, head))>
<!ATTLIST def
      name ID #IMPLIED>
<!ELEMENT fact (atom|neg) >
<!ELEMENT greater EMPTY>
<!ATTLIST greater
      sup IDREF #REQUIRED
      inf IDREF #REQUIRED>
<!ELEMENT head (atom|neg)>
<!ELEMENT body (atom|neg)*>
<!ELEMENT neg (atom)>
<!ELEMENT atom ((op,(ind | var)*) | ((ind | var)+,
op))>
<!ELEMENT ind (#PCDATA)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT op (#PCDATA)>
```

All the DR-Prolog files are available at:
http://www.csd.uoc.gr/~bikakis/DR-Prolog.

Related Work

There exist several previous implementations of defeasible logics. Conington *et al.* (2002) give the historically first implementation, *D-Prolog*, a Prolog-based implementation. It was not declarative in certain aspects (because it did not use a declarative semantic for the not operator), therefore it did not correspond fully to the abstract definition of the logic. Also, D-Prolog supported only one variation thus it lacked the flexibility of the implementation we report on. Finally it did not provide any means of integration with Semantic Web layers and concepts, a central objective of our work.
Deimos (Maher *et al.*, 2001) is a flexible, query processing system based on Haskell. It implements several variants, but not conflicting literals. Also, it does not integrate with Semantic Web (for example, there is no way to treat RDF data and RDFS/OWL ontologies; nor does it use an XML-based or RDF-based syntax for syntactic interoperability). Thus it is an isolated solution. Finally, it is propositional and does not support variables.
Delores (Maher *et al.*, 2001) is another implementation, which computes all conclusions from a defeasible theory. It is very efficient, exhibiting linear computational complexity. Delores only supports ambiguity blocking propositional defeasible logic; so, it does support ambiguity propagation, nor conflicting literals and variables. Also, it does integrate with other Semantic Web languages and systems, and is thus an isolated solution.
DR-DEVICE (Bassiliades, 2004) is another effort on implementing defeasible reasoning, albeit with a different approach. DR-DEVICE is implemented in Jess, and integrates well with RuleML and RDF. It is a system for query answering. Compared to the work of this paper, DR-DEVICE supports only one variant, ambiguity blocking, thus it does not offer the flexibility of this implementation. At present, it does not support RDFS and OWL ontologies.
SweetJess (Grosof *et al.*, 2002) is another implementation of a defeasible reasoning system (situated courteous logic programs) based on Jess. It integrates well with RuleML. Also, it allows for procedural attachments, a feature not supported by any of the above implementations, not by the system of this paper. However, SweetJess is more limited in flexibility, in that it implements only one reasoning variant (it corresponds to ambiguity blocking defeasible logic). Moreover, it imposes a number of restrictions on the programs it can map on Jess. In comparison, our system implements the full version of defeasible logic.

Conclusion

In this paper we described reasons why conflicts among rules arise naturally on the Semantic Web. To address this problem, we proposed to use defeasible reasoning which is known from the area of knowledge representation. And we reported on the implementation of a system for defeasible

reasoning on the Web. It is Prolog-based, supports RuleML syntax, and can reason with monotonic and nonmonotonic rules, RDF facts and RDFS and OWL ontologies..

Planned future work includes:

- Adding arithmetic capabilities to the rule language, and using appropriate constraint solvers in conjunction with logic programs.
- Implementing load/upload functionality in conjunction with an RDF repository, such as RDF Suite (Alexaki *et al*., 2001) and Sesame (Broekstra *et al*., 2003).
- Applications of defeasible reasoning and the developed implementation for brokering, bargaining, automated agent negotiation, and security policies.

References

Alexaki, S.; Christophides, V.; Karvounarakis, G.; Plexousakis, D.; and Trolle, K. 2001 The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. 2nd International Workshop on the Semantic Web (SemWeb'01).

Antoniou, G., and Arief, M. 2002. Executable Declarative Business rules and their use in Electronic Commerce. *In Proc. ACM Symposium on Applied Computing*

Antoniou, G.; Billington, D.; and Maher M. J. 1999. On the analysis of regulations using defeasible rules. *In Proc. 32nd Hawaii International Conference on Systems Science*

Antoniou G.; Billington D.; Governatori G.; and Maher M. J. 2001. Representation results for defeasible logic. *ACM Transactions on Computational Logic* 2, 2 (2001): 255 - 287

Antoniou G.; Maher M. J.; and Billington D. 2000a. Defeasible Logic versus Logic Programming without Negation as Failure. *Journal of Logic Programming* 41,1 (2000): 45-57

Antoniou G.; Billington, D.; Governatori G.; and Maher M. J. 2000b: A Flexible Framework for Defeasible Logics. In *Proc. AAAI' 2000*, 405-410

Antoniou G.; Maher M. J. 2002. Embedding Defeasible Logic into Logic Programs. In *Proc. ICLP 2002*, 393-404

Ashri, R.; Payne, T.; Marvin, D; Surridge, M.; and Taylor S. 2004. Towards a Semantic Web Security Infrastructure. In *Proc. of Semantic Web Services 2004 Spring Symposium Series*, Stanford University, California

Bassiliades, N; Antoniou, G; and Vlahavas, I. 2004. DR-DEVICE: A Defeasible Logic System for the Semantic Web. In Proc. 2nd Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR04), LNCS, Springer 2004 (accepted)

Berners-Lee, T; Hendler, J; and Lassila, O. 2001. The Semantic Web. *Scientific American*, 284, 5 (2001): 34-43

Broekstra, J; Kampman, A.; and van Harmelen, F. 2003 Sesame: An Architecture for Storin gand Querying RDF Data and Schema Information. In: D. Fensel, J. A. Hendler, H. Lieberman and W. Wahlster (Eds.), *Spinning the Semantic Web*, MIT Press, 197-222

Connolly, D; van Harmelen, F.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2001. *DAML+OIL Reference Description*. www.w3.org/TR/daml+oil-reference

Covington, M. A.; Nute, D.; and Vellino, A. 1997. *Prolog Programming in Depth*, 2nd ed. Prentice-Hall

Dean, M., and Schreiber, G. (Eds.) 2004. *OWL Web Ontology Language Reference*. www.w3.org/TR/2004/REC-owl-ref-20040210/

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38 (1991): 620—650

Governatori, G; Dumas, M.; ter Hofstede, A.; and Oaks, P. 2001. A formal approach to legal negotiation. In *Proc. ICAIL 2001*, 168-177

Grosof, B. N. 1997. Prioritized conflict handing for logic programs. In *Proc. of the 1997 International Symposium on Logic Programming*, 197-211

Grosof, B. N.; Gandhe, M. D.; and Finin T. W. 2002 SweetJess: Translating DAMLRuleML to JESS. RuleML 2002. In: *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*

Grosof, B. N.; Horrocks, I.; Volz, R; and Decker, S. 2003. Description Logic Programs: Combining Logic Programs with Description Logic". In: *Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003)*, ACM Press

Grosof, B. N., and Poon, T. C. 2003. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proc. 12th International Conference on World Wide Web*. ACM Press, 340 – 349

Levy, A., and Rousset M. C. 1998. Combining Horn rules and description logics in CARIN. *Artificial Intelligence* 104, 1-2 (1998):165 - 209

Li, N.; Grosof, B. N.; and Feigenbaum, J. 2003. Delegation Logic: A Logic-based Approach to Distributed Authorization. In: *ACM Transactions on Information Systems Security* 6,1 (2003)

Maher, M. J. 2002: A Model-Theoretic Semantics for Defeasible Logic. In *Proc. Paraconsistent Computational Logic 2002*, Datalogisker Srkifter 95 ,67-80

Maher, M. J. 2001. Propositional Defeasible Logic has Linear Complexity. *Logic Programming Theory and Practice* 1(6): 691-711 (2001)

Maher, M. J.; Rock, A.; Antoniou, G.; Billington, D.; and Miller, T. 2001. Efficient Defeasible Reasoning Systems. *International Journal of Tools with Artificial Intelligence* 10,4 (2001): 483--501

Marek, V. W., and Truszczynski, M. 1993. *Nonmonotonic Logics; Context Dependent Reasoning*. Springer Verlag

Nute, D. 1994. Defeasible logic. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*. Oxford University Press

RuleML. *The Rule Markup Language Initiative*. www.ruleml.org

SWI. SWI-Prolog, http://www.swi-prolog.org

Wagner, G. 2003. Web Rules Need Two Kinds of Negation. In Proc. First Workshop on Semantic Web Reasoning, LNCS 2901, Springer 2003, 33-50

XSB, Logic Programming and Deductive Database System for Unix and Windows. http://xsb.sourceforge.net

3.2 An Application of Answer Set Programming: Superoptimisation A Preliminary Report

An Application of Answer Set Programming: Superoptimisation A Preliminary Report

Martin Brain, Tom Crick, Marina De Vos and **John Fitch**
Department of Computer Science
University of Bath,
Bath BA2 7AY, UK
email: {mjb,tc,mdv,jpff}@cs.bath.ac.uk

Abstract

Answer set programming (ASP) is a declarative problem-solving technique that uses the computation of answer set semantics to provide solutions. Despite comprehensive implementations and a strong theoretical basis, ASP has yet to be used for more than a handful of large-scale applications. This paper describes such a large-scale application and presents some preliminary results. The TOAST (Total Optimisation using Answer Set Technology) project seeks to generate optimal machine code for simple, acyclic functions using a technique known as superoptimisation. ASP is used as a scalable computational engine for conducting searches over complex, non-regular domains. The experimental results suggest this is a viable approach to the optimisation problem and demonstrates the value of using parallel answer set solvers.

Introduction

Answer set programming (ASP) is a relatively new technology, with the first computation tools (referred to as answer set solvers) only appearing in the late 1990s (Niemelä & Simons 1997). Initial studies have demonstrated (WASP 2004) that it has great potential in many application areas, including automatic diagnostics (Eiter *et al.* 2000; Nogueira *et al.* 2001), agent behaviour and communication (De Vos *et al.* 2006), security engineering (P. Giorgini & Zannone 2004) and information integration (S. Costantini & Omodeo 2003). However, larger production scale applications are comparatively scarce. One of the few examples of such a system is the *USA-Advisor* decision support system for the NASA Space Shuttle (Nogueira *et al.* 2001). It modelled an extremely complex domain in a concise way; although of great significance to the field it is, in computational terms, relatively small. The only large and difficult programs most answer set solvers have been tested on are synthetic benchmarks. How well do the algorithms and implementations scale? How much memory and how much time is required? This paper makes an initial attempt to answer some of these questions.

This paper investigates the possibility of using ASP technology to generate optimal machine code for simple functions. Modern compilers apply a fixed set of code improvement techniques using a range of approximations rather than aiming to generate optimal code. None of the existing techniques, or approaches to creating new techniques, are likely to change the current state of play.

An approach to obtaining optimal code sequences is called superoptimisation (Massalin 1987). One of the main bottlenecks in this process is the size of the space of possible instruction sequences, with most superoptimising implementations relying on brute force searches to locate candidate sequences and approximate equivalence verification. The TOAST project presents a new approach to the search and verification problems using ASP.

From an ASP perspective, the TOAST project provides a large-scale, real-world application with some programs containing more than a million ground rules. From a compiler optimisation perspective, it might be a step towards tools that can generate truly optimal code, benefiting many areas, especially embedded systems and high performance computing.

This paper presents the results of the first phase of the TOAST project, with the overall infrastructure complete and three machine architectures implemented. We have used off-the-shelf solvers without any domain-specific optimisations, so the results we present also provide useful benchmarks for these answer set solvers.

The rest of this paper is structured as follows: in the next section, we provide a short introduction to modern compiler technology. In two subsections we explain the mechanisms of code optimisation, superoptimisation and verifiable code generation. In a third subsection we investigate the challenges of producing verifiable superoptimised sequences in terms of the length of input code sequences and word length of the target machine. We then give an overview of ASP from a programming language viewpoint. After these two background sections, we introduce the TOAST system and present the preliminary results. The analysis of these results leads to a section detailing the future work of the project.

The Problem Domain

Before describing the TOAST system and how it uses answer set technology, it is important to consider the problem that it seeks to solve and how this fits into the larger field of compiler design.

Compilers and Optimisation

Optimisation, as commonly used in the field of compiler research and implementation, is something of a misnomer.

A typical compiler targeting assembly language or machine code will include an array of code improvement techniques, from the relatively cheap and simple (identification of common sub-expressions and constant folding) (Aho, Sethi, & Ullmann 1986) to the costly and esoteric (auto-vectorisation and inter-function register allocation) (Appel 2004). However, none of these generate optimal code; the code that they output is only improved (though often to a significant degree). As all of these techniques identify and remove certain inefficiencies, it is impossible to guarantee that the code could not be further improved.

Further confusion is created by complications in defining optimality. In the linear case, a shorter instruction sequence is clearly better[1]. If the code branches but is not cyclic, a number of definitions are possible: shortest average path, shortest over all sequence, etc. However, for code including cycles, it is not possible to define optimality in the general case. To do so would require calculating how many time the body of loop would be executed – a problem equivalent to the halting problem. To avoid this, and problems with other areas such as equivalence of floating point operations, this paper only considers optimality in terms of the number of instructions used in acyclic, integer-based code.

Finally, it is important to consider the scale of the likely savings. The effect of improvements in code generation for an average program have been estimated as a 4% speed increase[2] per year (Proebsting 1998). In this context, saving just one or two instructions is significant, particularly if the technique is widely applicable, or can be used to target 'hot spots', CPU-intensive sections of code.

Superoptimisation

Superoptimisation is a radically different approach to code generation, first described in (Massalin 1987). Rather than starting with crudely generated code and improving it, a superoptimiser starts with the specification of a function and performs an exhaustive search for a sequence of instructions that meets this specification. Clearly, as the length of the sequence increases, the search space potentially rises at an exponential rate. This makes the technique unsuitable for use in normal compilers, but for improving the code generators of compilers and for targeting key sections of performance-critical functions, the results can be quite impressive.

A good example of superoptimisation is the sign function (Massalin 1987), which returns the sign of a binary integer, or zero if the input is zero:

```
int signum (int x) {
        if (x > 0)      return 1;
        else if (x < 0) return -1;
        else            return 0;
}
```

A naïve compilation of this function would produce approximately ten instructions, including at least two conditional branch instructions. A skilled assembly language programmer may manage to implement it in four instructions with one conditional branch. At the time of writing, this is the best that state of the art compilation can produce. However, superoptimisation (in this case for the SPARC-V7 architecture) gives the following:

```
! input in %i0
addcc   %i0 %i0 %l1
subxcc  %i0 %l1 %l2
addx    %l2 %i0 %o1
! output in %o1
```

Not only is this sequence only three instructions long, it does not require any conditional branches, a significant saving on modern pipelined processors. This example also demonstrates another interesting property of code produced by superoptimisation: it is not obvious that this computes the sign of a number or how it does so. The pattern of addition and subtraction essentially 'cancels out', with the actual computation done by how the carry flag is set and used by each instruction (instructions whose name includes `cc` set the carry flag, whereas instructions with `x` use the carry flag). Such inventive use of a processor's features are common in superoptimised sequences; when the GNU Superoptimizer (GSO) (Granlund & Kenner 1992) was first used to superoptimise sequences for the GCC port to the POWER architecture, it produced a number of sequences that were shorter than the processor's designers thought possible!

Despite significant potential, superoptimisation has received relatively little attention within the field of compiler research. Following Massalin's work, the next published superoptimiser was GSO, a portable superoptimiser developed to aid the development of GCC. It improved on Massalin's search strategy by attempting to apply constraints while generating elements of the search space, rather than generating all possible sequences and then skipping those that were marked as clearly redundant. The most recent work on superoptimisation have been from the Denali project (Joshi, Nelson, & Randall 2002; Joshi, Nelson, & Zhou 2003). Their approach was much closer to that of the TOAST system, using automatic theorem-proving technology to handle the large search spaces.

Analysis of Problem Domain

Superoptimisation naturally breaks into two sub-problems: searching for sequences that meet some limited criteria and verifying which of these candidates are fully equivalent to the input function.

The search space of possible sequences of a given length is very large, at least the number of instructions available to the power of the length of the sequences (thus growing at least exponentially as the length rises). However, a number of complex constraints exist that reduce the space that has

[1] Although the TOAST approach could be generalised to handle them, this paper ignores complications such as pipelining, caching, pre-fetching, variable-instruction latency and super-scalar execution.

[2] This may seem very low in comparison with the increase in processing power created by advances in microprocessor manufacturing. However, it is wise to consider the vast disparity in research spending in the two areas, as well as the link between them: most modern processors would not achieve such drastic improvements without advanced compilers to generate effi cient code for them.

to be searched. For example, if a sub-sequence is known to be non-optimal then anything that includes it will also be non-optimal and thus can be discarded. Managing the size and complexity of this space is the current limit on superoptimiser performance.

Verifying that two code sequences are equivalent also involves a large space of possibilities (for single input sequences it is 2^w where w is the word length (number of bits per register) of the processor). However, it is a space that has a number of unusual properties. Firstly, verification of two sequences is a reasonably simple task for human experts, suggesting there may be a strong set of heuristics. Secondly, sequences of instructions that are equivalent on a reasonably small subset of the space of possible inputs tend to be equivalent on all of it. Both GSO and Massalin's original superoptimiser handled verification by testing the new sequence for correctness of a small number of inputs and declaring it equivalent if it passed. Although non-rigorous, this approach seemed to work in practise (Granlund & Kenner 1992).

Answer Set Programming

Answer set programming is a declarative problem solving technique based on research on the semantics of logic programming languages and non-monotonic reasoning (Gelfond & Lifschitz 1988; 1991). For reasons of compactness, this paper only includes a brief summary of answer set semantics; a more in-depth discussion can be found in (Baral 2003).

Answer set semantics are defined with respect to *programs*, sets of Horn clause-style rules composed of literals. Two forms of negation are described, negation as failure and explicit (or classical) negation. The first (denoted as not) is interpreted as not knowing that the literal is true, while the second (denoted as $\neg$) is knowing that the literal is not true. For example:

$$a \leftarrow b, not\ c.$$
$$\neg b \leftarrow not\ a.$$

is interpreted as "a is known to be true if b is known to be true and c is not known to be true. b is known to be not true if a is not known to be true" (the precise declarative meaning is an area of ongoing work, see (Denecker 2004)). Constraints are also supported, which allow conjunctions of literals to be ruled as inconsistent. Answer sets are sets of literals that are consistent (do not contain both a and $\neg a$ or the bodies of any constraints) and supported (every literal has at least one, acyclic way of concluding its truth). A given program may have zero or more answer sets.

Answer set programming is describing a problem as a program under answer set semantics in such a way that the answer sets of the program correspond to the solutions of the problem. In many cases, this is simply a case of encoding the description of the problem domain and the description of what constitutes a solution. Thus solving the problem is reduced to computing the answer sets of the program.

Computing an answer set of a program is an NP-complete task, but there are a number of sophisticated tools, known as answer set solvers, that can perform this computation. The first generation of efficient solvers (such as SMODELS (Niemelä & Simons 1997) and DLV (Leone *et al.* 2006)) use a DPLL-style algorithm (Davis, Logemann, & Loveland 1962). Before computation, the answer set program is *grounded* (an instantiation process that creates copies of the rules for each usable value of each variable) by using tools such as LPARSE (Syrjänen 2000), to remove variables. The answer sets are then computed using a backtracking algorithm; at each stage the sets of literals that are known to be true and known to be false are expanded according to a set of simple rules (similar to unit propagation in DPLL), then a branching literal is chosen according to heuristics and both possible branches (asserting the literal to be true or false) are explored. An alternative approach is to use a SAT solver to generate candidate answer sets and then check whether these meet all criteria. This is the approach used by CMODELS (Giunchiglia, Lierler, & Maratea 2004). More recent work has investigated using 'Beowulf'-style parallel systems to explore possible models in parallel (Pontelli, Balduccini, & Bermudez 2003). One such system, PLATYPUS (Gressmann *et al.* 2005) is used in the TOAST system.

TOAST

The existence of a clear NP algorithm, as well as the causal nature of the problem and the need for high expressive and computational power, suggest ASP as a suitable approach to the superoptimisation problem. The TOAST system consists of a number of components that generate answer set programs and parse answer sets, with a 'front end' that uses these components to produce a superoptimised version of an input function. Data is passed between components either as fragments of answer set programs or in an architecture-independent, assembly language-like format. An answer set solver is used as a 'black box' tool, currently either SMODELS or PLATYPUS, although experiments with other solvers are ongoing. Although the grounding tool of DLV is stronger in some notable examples, it has not been tested yet due to syntax incompatibilities with many of the features required.

System Components

Four key components provide most of the functionality of the TOAST system:

pickVectors Given the specification of the input to an instruction sequence, *pickVectors* creates a representative set of inputs, known as input vectors, and outputs it as an ASP program fragment.

execute This component takes an ASP program fragment describing an input vector (as generated by *pickVector* or *verify*) and emulates running an instruction sequence with that input. The output is the given as another ASP program fragment containing constraints on the instruction sequence's outputs.

search Taking ASP fragments giving 'input' and 'output' values (from *pickVectors* / *verify* and *execute* respectively), this component searches for all instruction se-

quences of a given length that produce the required 'output' for the given 'input' values.

verify Takes two instruction sequences with the same input specification and tests if they are equivalent. If they are not, an input vector on which they differ can be generated, in the format used by *execute* and *search*.

The TOAST system is fully architecture-independent. Architecture-specific information is stored in a description file which provides meta-information about the architecture, as well as which operations from the library of instructions are available. At the time of writing, TOAST supports the MIPS R2000 and SPARC V7/V8 processors. Porting to a new architecture is simple and takes between a few hours and a week, depending on how many of the instructions have already been modelled.

System Architecture

The key observation underlying the design of the TOAST system is that any correct superoptimised sequence will be returned by running *search* for the appropriate instruction length; however, not everything that *search* returns is necessarily a correct answer. Thus to generate superoptimised sequences, the front end uses *pickVector* and *execute* on the input instruction sequence to create criteria for *search*. Instruction sequence lengths from one up to one less than the length of the original input sequence are then sequentially searched. If answers are generated, another set of criteria are created and the same length searched again. The two sets are then intersected, as any correct answer must appear in both sets. This process is repeated until either the intersection becomes empty, in which case the search moves on to the next sequence length, or until the intersection does not decrease in size. *verify* can then be used to check members of this set for equivalence to the original input program.

The Answer Set Programs

In the following section we give a brief overview of the basic categories of answer set programs generated within the system: flow control, flag control, instruction sequences, instruction definitions, input vectors and output constraints.

The flow control rules set which instruction will be 'executed' at a given time step by controlling the `pc` (program counter) literal. An example set of flow control rules are given in Figure 1. The rules are simple, such as an instruction that asserts `jump(C,T,J)` would move the program's execution on `J` instructions, otherwise it will just move on by one. As the ASP programs may need to simultaneously model multiple independent code streams (for example, when trying to verify their equivalence), all literals are tagged with a abstract entity called 'colour'. The inclusion of the `colour(C)` literal in each rule allows copies to be created for each separate code stream during instantiation. In most cases, when only one code stream is used, only one value of `colour` is defined and only one copy of each set of rules is produced; the overhead involved is negligible.

Flag control rules control the setting and maintenance of processor flags such as carry, overflow, zero and negative. Although generally only used for controlling conditional

```
haveJumped(C,T) :- jump(C,T,J), jumpSize(C,J),
                   time(C,T), colour(C).

pc(C,PCV+J,T+1) :- pc(C,PCV,T), jump(C,T,J), jumpSize(C,J),
                   time(C,T), colour(C), position(C,PCV).

pc(C,PCV+1,T+1) :- pc(C,PCV,T), not haveJumped(C,T),
                   time(C,T), colour(C), position(C,PCV).

pc(C,1,1).
```

Figure 1: Flow Control Rules in ASP

```
value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                value(C,R1,B), -value(C,R2,B),
                register(R1), register(R2), colour(C),
                position(C,P), time(C,T), bit(B).

value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                -value(C,R1,B), value(C,R2,B),
                register(R1), register(R2), colour(C),
                position(C,P), time(C,T), bit(B).

-value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                 not value(C,T,B), register(R1), register(R2),
                 colour(C), position(C,P), time(C,T), bit(B).

symmetricInstruction(lxor).
```

Figure 2: Modelling of a Logical XOR Instruction in ASP

branches and multi-word arithmetic, the flags are a source of many superoptimised sequences and are thus of prime importance when modelling.

The instruction sequence itself is represented as a series of facts, or in the case of *search*, a set of choice rules (choice rules are a syntactic extension to ASP, see (Niemelä & Simons 1997)). The literals are then used by the instruction definitions to control the `value` literals that give the value of various registers within the processor. If the literal is in the answer set, the given bit is taken to be a 1, if the classically-negated version of the literal is in the answer set then it is a 0. An example instruction definition, for a logical XOR (exclusive or) between registers, is given in Figure 2. Note the use of negation as failure to reduce the number of rules required and the declaration that `lxor` is symmetric, which is used to reduce the search space.

The input vectors and output constraints are the program fragments created by *pickVectors* and *execute* respectively.

The ASP programs generated do not contain disjunction, aggregates or any other non-syntactic extensions to answer set semantics.

Results

Tests were run on a Beowulf-style cluster of 20 x 800MHz Intel Celeron, 512MB RAM machines connected by 100Mb Ethernet, running SuSE Linux 9.2. Results are given for SMODELS v2.28 (denoted s) and the initial MPI version of PLATYPUS running on n nodes (denoted p/n). LPARSE v1.0.17 was used in all cases to ground the programs. The timings displayed are from the SMODELS internal timing mechanism and the PLATYPUS MPI wall time respectively. Values for LPARSE are the user times given via the system `time` command.

Search Time

search was used to generate programs that searched the space of SPARC-V7 instructions for candidate superoptimisations for the following instruction sequence:

```
! input in %i0, %i1
and     %i0 %i1 %l1
add     %i0 %l1 %l2
add     %i0 %l2 %l3
sub     0   %l3 %o1
! output in %o1
```

This sequence was selected as a 'worst case', an example of a sequence that cannot be superoptimised, giving an approximate ceiling on the performance of the system.

Statistics on the programs used can be found in Figure 3, with the timing results are given in Figure 4.

Verification Time

verify was used to create a verification program for the following two code sequences:

```
! input in %i0       ! input in %i0
add %i0 %i0 %o1      umult %i0  2  %o1
! output in %o1      ! output in %o1
```

using the SPARC-V8[3] architecture but varying the processor word length (the number of bits per register). This pair of programs were chosen as, although they are clearly equivalent, the modelling and reasoning required to show this is non-trivial. Timing results for a variety of solver configurations and different word lengths can be found in Figure 7, program statistics can be found in Figure 5.

Analysis

The experimental results presented suggest a number of interesting points. Firstly, superoptimisation using ASP is feasible, but work is needed to make it more practical. Given that only a few constraints were used in the programs generated by *search*, increasing the length of the maximum practical search space seems eminently possible. The result from *verify* are less encouraging; although it shows it is possible using ASP, it also suggests that attempting to verify instruction sequences of more than 32 bits of input is likely to require significant resources.

The graph in Figure 6 also shows some interesting properties of the parallel solver. The overhead of the solver appears to be near constant, regardless of the number of processors used. For the simpler problems, the overhead of the parallel solver is greater than any advantages, but for the larger problems it makes a significant difference and the speed-up is approximately proportional to the number of processors used.

Finally, the figures suggest that the SMODELS algorithm does not scale linearly on some programs. The programs output by *verify* double in search space size for each increase in word length, but the time required by SMODELS rises by significantly more than a factor of two. Strangely, this additional overhead appears to be less significant as the number of processors used by PLATYPUS rises.

[3]SPARC-V8 is a later, minimal extension of SPARC-V7 with the addition of the `umult` instruction.

The simplified graph in Figure 6 shows these effects, with time graphs for SMODELS against PLATYPUS with 4, 8 and 16 processors.

Future Development

One of the key targets in the development of TOAST is to reduce the amount of time required in searching. Doing so will also increase the length of instruction sequence that can be found. This requires improvements to both the programs that are generated and the tools used to solve them.

A key improvement to the generated programs will be to remove all short sequences that are known to be non-optimal. *search* can be used to generate all possible instruction sequences of a given length. By superoptimising each one of these for the smaller lengths, it is then possible to build a set of equivalence categories of instructions. Only the shortest member of each category needs to be in the search space and thus a set of constraints can be added to the programs that *search* generates. This process only ever needs to be done once for each processor architecture and will give significant improvements in terms of search times. The equivalence classes generated may also be useful to improve verification.

The other developments needed to reduce the *search* time are in the tools used. Addressing the amount of memory consumed by LPARSE and attempting to improve the scaling of the SMODELS algorithm are both high priorities.

The performance of *verify* also raises some interesting questions. At present, is possible to verify programs for some of the smaller, embedded processors. However, in its current form it is unlikely to scale to high-end, 64 bit processors. A number of alternative approaches are being considered, such as attempting to prove equivalence results about the generated ASP programs, reducing the instructions to a minimal/pseudo-normal form (an approach first used by Massalin), using some form of algebraic theorem-proving (as in the Denali project) or attempting to formalise and prove the observation that sequences equivalent on a small set of points tend to be equivalent on all of them.

Using the TOAST system to improve the code generated by tools such as GCC is also a key target for the project. By implementing tools that translate between the TOAST internal assembly-like format and processor-specific assembly, it will be possible to check the output of GCC for sequences that can be superoptimised. Patterns that occur regularly can then be added to the instruction generation phases of GCC. Performance-critical system libraries, such as the GNU Multiple Precision Arithmetic Library (GMP) (Granlund 2006) and code generators used by Just In Time (JIT) compilers could also be interesting application areas.

It is hoped that it will not only prove useful as a tool for optimising sections of performance critical code, but that the ASP programs could be used as benchmarks for solver performance and the basis of other applications which reason about machine code.

Length of Sequence	No. rules	Grounding time	No. ground rules	No. of atoms
1	530	20.100	95938	1018
2	534	65.740	298312	1993
3	538	142.22	643070	3428
4	542	-	1197182	6873

Figure 3: Search Program Sizes

Length of Sequence	s	$p/2$	$p/4$	$p/6$	$p/8$	$p/10$	$p/12$	$p/14$	$p/16$	$p/18$	$p/20$
1	3.057	10.4647	10.4813	10.4761	10.5232	10.5023	10.4674	10.4782	10.4833	10.4915	10.5040
2	99.908	104.710	123.312	120.984	135.733	136.057	139.944	139.000	135.539	139.271	138.288
3	81763.9	63644.4	19433.4	12641.0	6008.20	7972.73	9097.83	6608.64	6063.08	4629.90	5419.08
4	$>$ 237337.35	-	-	-	-	-	-	-	-	-	-

Figure 4: Search Space Size v Compute Time (secs)

Word Length	No. rules	Grounding time	No. ground rules	No. of atoms
8	779	1.220	1755	975
9	780	1.320	2063	1099
10	781	1.430	2402	1235
11	782	1.480	2772	1383
12	783	1.330	3173	1543
13	784	1.350	3605	1715
14	785	1.450	4068	1899
15	786	1.480	4562	2095
16	787	1.480	5087	2303
17	788	1.640	5645	2527
18	789	1.680	6234	2763
19	790	1.690	6854	3011
20	791	1.550	7505	3271
21	792	1.590	8187	3543
22	793	1.670	8900	3827
23	794	1.900	9644	4123
24	795	1.830	10419	4431

Figure 5: Verification Program Sizes

Conclusion

This paper suggests that ASP can be used to solve large-scale, real-world problems. Future work will hopefully show this is also a powerful approach to the superoptimisation problem and perhaps even a 'killer application' for ASP.

However, it is not without challenges. Although savings to both size of the ASP programs used and their search spaces are possible, this will remain a high-end application for answer set solvers. Some of the features required, such as the handling of large, sparse search spaces and efficiency in producing all possible answer sets (or traversing the search space of programs without answer sets) are unfortunately not key targets of current solver development.

The TOAST project demonstrates that answer set technology is ready to be used in large-scale applications, although more work is required to make it competitive.

References

Aho, A. V.; Sethi, R.; and Ullmann, J. D. 1986. *Compilers: Principles, Techniques and Tools*. Addison-Wesley.

Appel, A. W. 2004. *Modern Compiler Implementation in C*. Cambridge University Press.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem-Proving. *Communications of the ACM* 5(7):394–397.

De Vos, M.; Crick, T.; Padget, J.; Brain, M.; Cliffe, O.; and Needham, J. 2006. A Multi-agent Platform using Ordered Choice Logic Programming. In *Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT'05)*, volume 3904 of *LNAI*, 72–88. Springer.

Denecker, M. 2004. What's in a Model? Epistemological Analysis of Logic Programming. In *Proceedings of the 9th*

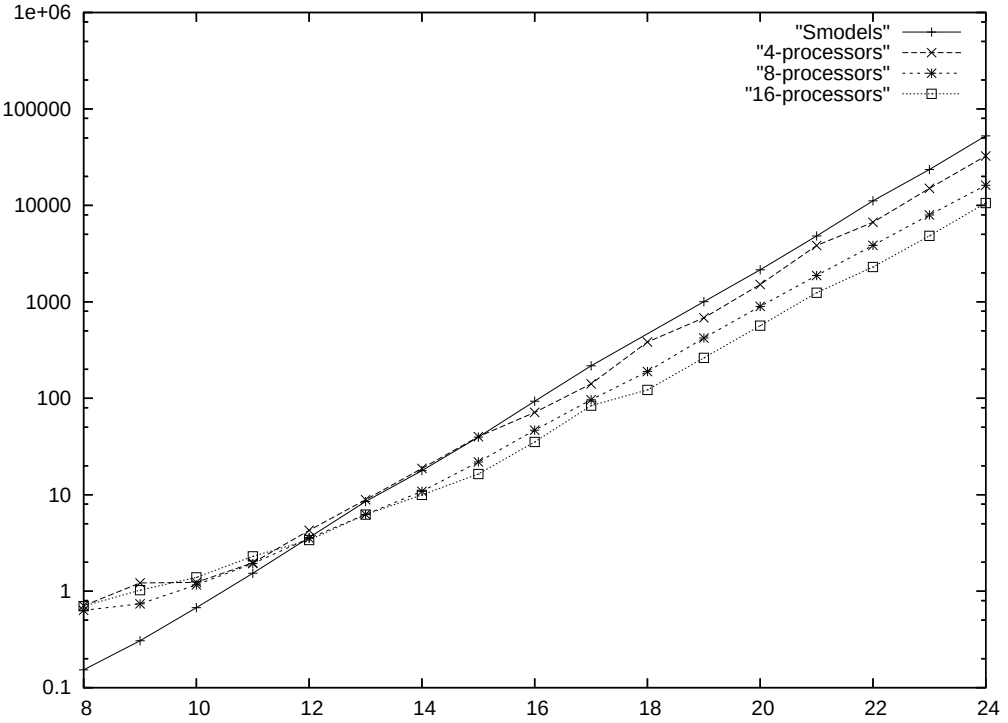


Figure 6: Simplified Timings (Log Scale)

Word Length	s	$p/2$	$p/4$	$p/6$	$p/8$	$p/10$	$p/12$	$p/14$	$p/16$	$p/18$	$p/20$
8 bit	0.153	0.495074	1.21623	0.581861	0.632791	0.662914	0.706752	1.21751	0.698032	0.723088	0.740474
9 bit	0.306	0.863785	0.705636	0.777568	0.740043	1.02031	0.918548	0.864449	1.02644	1.03752	1.09627
10 bit	0.675	1.61512	1.2337	1.23213	1.16333	1.23683	1.28347	1.28118	1.39326	1.29568	1.31185
11 bit	1.537	3.42153	1.97181	1.84315	1.93191	2.01146	1.9929	2.34911	2.2948	2.28081	2.18609
12 bit	3.597	7.46042	4.28284	3.43396	3.53243	3.33475	3.27878	3.16487	3.38788	3.21397	3.94176
13 bit	8.505	15.8174	8.86814	6.51479	6.25371	5.55683	5.1507	5.3369	6.22179	5.61428	5.06376
14 bit	17.795	34.2229	18.7478	15.9874	10.8228	9.57001	9.3808	8.6161	9.97594	9.41512	8.16737
15 bit	39.651	76.018	39.9688	25.9992	21.8607	19.382	17.6372	18.0614	16.3806	15.6143	15.6043
16 bit	93.141	167.222	71.3785	52.7732	46.6144	36.5995	31.9568	33.2825	35.3159	27.2188	29.5464
17 bit	217.162	373.258	141.108	110.65	96.6821	85.1217	77.4811	78.7892	83.9177	56.1338	58.4057
18 bit	463.025	815.373	384.237	222.826	189.690	162.318	144.840	136.126	122.038	118.658	133.579
19 bit	1002.696	1738.02	681.673	456.607	421.681	430.879	299.870	290.456	262.611	229.802	217.998
20 bit	2146.941	3790.84	1514.80	994.849	896.705	726.629	625.820	610.117	566.040	523.700	426.004
21 bit	4826.837	8206.4	3438.71	2279.3	1874.36	1544.74	1461.4	1199.96	1244.95	932.877	1128.53
22 bit	11168.818	17974.8	6683.06	4375.12	3850.71	3017.14	3206.33	2492.00	2296.87	2245.3	1869.17
23 bit	23547.807	38870.5	15047	9217.82	7947.95	7123.56	6111.6	6089.38	4833.66	4610.92	4020.37
24 bit	52681.498	83405.1	32561.2	20789.1	16165.4	14453.8	12800.7	11213.2	10580.4	9199.8	8685.47

Figure 7: Word Length v Compute Time (secs)

International Conference on the Principles of Knowledge Representation and Reasoning (KR2004), 106–113.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2000. Using the dlv system for planning and diagnostic reasoning. In *Proceedings of the 14th Workshop on Logic Programming (WLP'99)*, 125–134.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In Kowalski, R. A., and Bowen, K., eds., *Proceedings of the 5th International Conference on Logic Programming (ICLP'88)*, 1070–1080. The MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3-4):365–386.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2004. SAT-Based Answer Set Programming. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-04)*, 61–66.

Granlund, T., and Kenner, R. 1992. Eliminating Branches using a Superoptimizer and the GNU C Compiler. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'92)*,

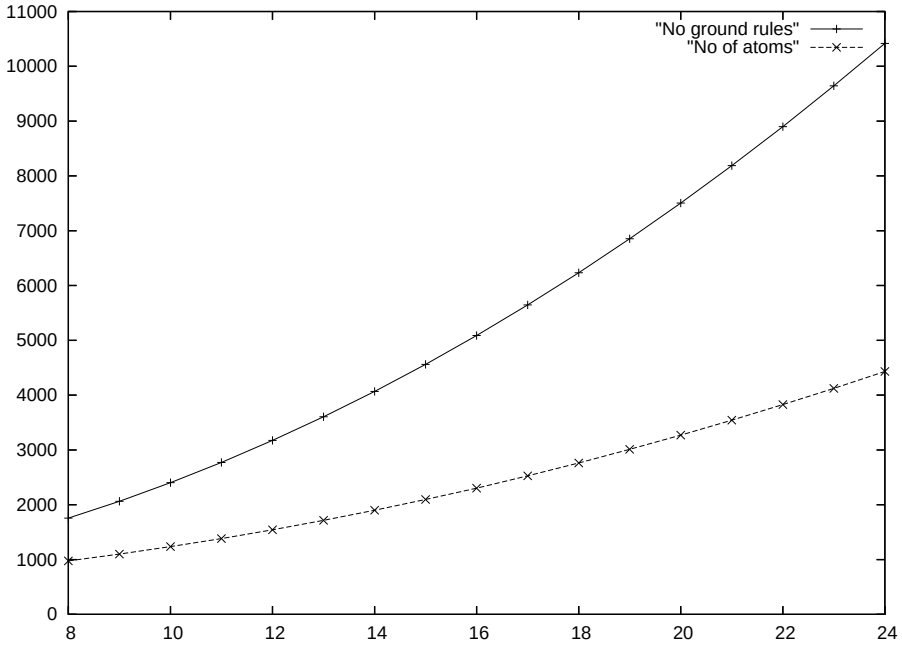


Figure 8: Number of Rules/Atoms v Word Length

341–352. ACM Press.

Granlund, T. 2006. GMP : GNU Multiple Precision Arithmetic Library. `http://www.swox.com/gmp/`.

Gressmann, J.; Janhunen, T.; Mercer, R.; Schaub, T.; Thiele, S.; and Tichy, R. 2005. Platypus: A Platform for Distributed Answer Set Solving. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, 227–239.

Joshi, R.; Nelson, G.; and Randall, K. 2002. Denali: A Goal-Directed Superoptimizer. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*, 304–314. ACM Press.

Joshi, R.; Nelson, G.; and Zhou, Y. 2003. The Straight-Line Automatic Programming Problem. Technical Report HPL-2003-236, HP Labs.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for Knowledge Representation and Reasoning. *to appear in ACM Transactions on Computational Logic (TOCL)*.

Massalin, H. 1987. Superoptimizer: A Look at the Smallest Program. In *Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'87)*, 122–126. IEEE Computer Society Press.

Niemelä, I., and Simons, P. 1997. Smodels: An Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *LNAI*, 420–429. Springer.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An A-Prolog Decision Support System for the Space Shuttle. In *Proceedings of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, 169–183. Springer-Verlag.

P. Giorgini, F. Massacci, J. M., and Zannone, N. 2004. Requirements Engineering Meets Trust Management: Model, Methodology, and Reasoning. In *Proceedings of the 2nd International Conference on Trust Management (iTrust 2004)*, volume 2995 of *LNCS*, 176–190. Springer.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Non-Monotonic Reasoning on Beowulf Platforms. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages (PADL'03)*, 37–57. Springer-Verlag.

Proebsting, T. 1998. Proebsting's Law: Compiler Advances Double Computing Power Every 18 Years. `http://research.microsoft.com/~toddpro/papers/law.htm`.

S. Costantini, A. F., and Omodeo, E. 2003. Mapping Between Domain Models in Answer Set Programming. *Proceedings of Answer Set Programming: Advances in Theory and Implementation (ASP'03)*.

Syrjänen, T. 2000. *Lparse 1.0 User's Manual*. Helsinki University of Technology.

WASP. 2004. WP5 Report: Model Applications and Proofs-of-Concept. `http://www.kr.tuwien.ac.at/projects/WASP/wasp-wp5-web.html`.

3.3 COBA 2.0: A Consistency-Based Belief Change System

COBA 2.0: A Consistency-Based Belief Change System

James P. Delgrande and **Daphne H. Liu**
School of Computing Science
Simon Fraser University
Burnaby, B.C.
Canada V5A 1S6
{jim, daphnel}@cs.sfu.ca

Torsten Schaub* and **Sven Thiele**
Institut für Informatik
Universität Potsdam
Postfach 60 15 53, D-14415
Potsdam, Germany
{torsten@cs, sthiele@rz}.uni-potsdam.de

Abstract

We describe COBA 2.0, an implementation of a consistency-based framework for expressing belief change, focusing here on revision and contraction, with the possible incorporation of integrity constraints. This general framework was first proposed in (Delgrande & Schaub 2003); following a review of this work, we present COBA 2.0's high-level algorithm, work through several examples, and describe our experiments. A distinguishing feature of COBA 2.0 is that it builds on SAT-technology by using a module comprising a state-of-the-art SAT-solver for consistency checking. As well, it allows for the simultaneous specification of revision, multiple contractions, along with integrity constraints, with respect to a given knowledge base.

Introduction

Given a knowledge base and a sentence for revision or contraction, the fundamental problem of belief change is to determine what the resulting knowledge base contains. The ability to change one's knowledge is essential for an intelligent agent. Such change in response to new information is not arbitrary, but rather is typically guided by various rationality principles. The best known of these sets of principles was proposed by Alchourron, Gardenfors, and Makinson (Alchourrón, Gärdenfors, & Makinson 1985), and has come to be known as the AGM approach.

In this paper, we describe COBA 2.0, an implementation of a consistency-based approach to belief revision and contraction. The general methodology was first proposed in (Delgrande & Schaub 2003). In this approach, the AGM postulates for revision are effectively satisfied, with the exception of one of the "extended" postulates. Similarly the contraction postulates are satisfied with the exception of the controversial recovery postulate and one of the extended postulates. Notably the approach is syntax independent, and so independent of how a knowledge base and sentence for belief change is represented. COBA 2.0 the implements this approach, and in a more general form. Thus a single belief change opertaion will involve a single knowledge base and (possibly) a sentence for revision, but along with (possibly) a set of sentences for contraction; as well integrity constraints are handled, and in a straightforward fashion.

*Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, B.C., Canada.

In Section 2, we give background terminology, notations, and implementation considerations. Section 3 presents COBA 2.0's high-level algorithm, in addition to working through two examples. Section 4 discusses COBA 2.0's features, syntax, and input checks, while Section 5 describes our experiments evaluating COBA 2.0 against a comparable solver. Lastly, Section 6 concludes with a summary.

Preliminaries

To set the stage, we informally motivate our original approach to belief revision; contraction is motivated similarly, and is omitted here given space considerations. First, the syntactic form of a sentence doesn't give a clear indication as to which sentences should or should not be retained in a revision. Alternately, one can consider interpretations, and look at the models of K and α. The interesting case occurs when $K \cup \{\alpha\}$ is unsatisfiable because K and α share no models. Intuitively, a model of $K\dot{+}\alpha$ should then contain models of α, but incorporating "parts" of models of K that don't conflict with those of α. That is, we will have $Mod(K\dot{+}\alpha) \subseteq Mod(\alpha)$, and for $m \in Mod(K\dot{+}\alpha)$ we will want to incorporate whatever we can of models of K.

We accomplish this by expressing K and α in different languages, but such that there is an isomorphism between atomic sentences of the languages. In essence, we replace every occurrence of an atomic sentence p in K by a new atomic sentence p', yielding knowledge base K' and leaving α unchanged. Clearly, under this relabelling, the models of K' and α will be independent, and $K' \cup \{\alpha\}$ will be satisfiable (assuming that each of K, α are satisfiable). We now assert that the languages agree on the truth values of corresponding atoms wherever consistently possible. So, for every atomic sentence p, we assert that $p \equiv p'$ whenever this is consistent with $K' \cup \{\alpha\}$ along with the set of equivalences obtained so far. We obtain a maximal set of such equivalences, call it EQ, such that $K' \cup \{\alpha\} \cup EQ$ is consistent. A model of $K' \cup \{\alpha\} \cup EQ$ then will be a model of α in the original language, wherein the truth values of atomic sentences in K' and α are linked via the set EQ. A candidate "choice" revision of K by α consists of $K' \cup \{\alpha\} \cup EQ$ re-expressed in the original language. General revision corresponds to the intersection of all candidate choice revisions. The following section gives an example, once we have given a formal summary of the approach.

Formal Preliminaries

We deal with propositional languages and use the logical symbols $\top$, $\bot$, $\neg$, $\vee$, $\wedge$, $\supset$, and $\equiv$ to construct formulas in the standard way. We write $\mathcal{L}_\mathcal{P}$ to denote a language over an alphabet $\mathcal{P}$ of propositional letters or atomic propositions. Formulas are denoted by the Greek letters α, β, α_1, ... Knowledge bases, identified with belief sets or deductively-closed sets of formulas, are denoted by K, K_1, ... So $K = Cn(K)$, where $Cn(\cdot)$ is the deductive closure in classical propositional logic of the formula or set of formulas given as argument. Given an alphabet $\mathcal{P}$, we define a disjoint alphabet $\mathcal{P}'$ as $\mathcal{P}' = \{p' \mid p \in \mathcal{P}\}$. For $\alpha \in \mathcal{L}_\mathcal{P}$, α' is the result of replacing in α each proposition $p \in \mathcal{P}$ by the corresponding proposition $p' \in \mathcal{P}'$ (and hence an isomorphism between $\mathcal{P}$ and $\mathcal{P}'$). This definition applies analogously to sets of formulas.

A *belief change scenario* in $\mathcal{L}_\mathcal{P}$ is a triple $B = (K, R, C)$ where K, R, and C are sets of formulas in $\mathcal{L}_\mathcal{P}$. Informally, K is a belief set that is to be modified so that the formulas in R are contained in the result, and the formulas in C are not. An extension determined by a belief change scenario is defined as follows.

Definition 1 (Belief Change Extension) *Let $B = (K, R, C)$ be a belief change scenario in $\mathcal{L}_\mathcal{P}$, and a maximal set of equivalences $EQ \subseteq \{p \equiv p' \mid p \in \mathcal{P}\}$ be such that $Cn(K' \cup R \cup EQ) \cap (C \cup \{\bot\}) = \emptyset$.*

Then $Cn(K' \cup R \cup EQ) \cap \mathcal{L}_\mathcal{P}$ is a belief change extension *of B. If there is no such set EQ, then B is* inconsistent *and $\mathcal{L}_\mathcal{P}$ is defined to be the sole* (inconsistent) belief change extension *of B.*

In Definition 1, "maximal" is with respect to set containment, and the exclusive use of "$\{\bot\}$" is to take care of consistency if $C = \emptyset$. Definition 1 provides a very general framework for specifying belief change. Next, we can restrict the definition to obtain specific functions for belief revision and contraction.

Revision and Contraction. For a given belief change scenario, there may be more than one consistent belief change extension. We can thus use a *selection function* c that, for any set $I \neq \emptyset$, has as value some element of I.

Definition 2 (Revision) *Let K be a knowledge base, α a formula, and $(E_i)_{i \in I}$ the family of all belief change extensions of $(K, \{\alpha\}, \emptyset)$. Then, we define*

1. *$K \dot{+}_c \alpha = E_i$ as a* choice revision *of K by α with respect to some selection function c with $c(I) = i$.*
2. *$K \dot{+} \alpha = \bigcap_{i \in I} E_i$ as the* (skeptical) revision *of K by α.*

Definition 3 (Contraction) *Let K be a knowledge base, α a formula, and $(E_i)_{i \in I}$ the family of all belief change extensions of $(K, \emptyset, \{\alpha\})$. Then, we define*

1. *$K \dot{-}_c \alpha = E_i$ as a* choice contraction *of K by α with respect to some selection function c with $c(I) = i$.*
2. *$K \dot{-} \alpha = \bigcap_{i \in I} E_i$ as the* (skeptical) contraction *of K by α.*

K'	α	EQ	$K \dot{+} \alpha$
$p' \wedge q'$	$\neg q$	$\{p \equiv p'\}$	$p \wedge \neg q$
$\neg p' \equiv q'$	$\neg q$	$\{p \equiv p', q \equiv q'\}$	$p \wedge \neg q$
$p' \vee q'$	$\neg p \vee \neg q$	$\{p \equiv p', q \equiv q'\}$	$p \equiv \neg q$
$p' \wedge q'$	$\neg p \vee \neg q$	$\{p \equiv p'\}, \{q \equiv q'\}$	$p \equiv \neg q$

Table 1: Skeptical Revision Examples

K'	α	EQ	$K \dot{-} \alpha$
$p' \wedge q'$	q	$\{p \equiv p'\}$	p
$p' \wedge q' \wedge r'$	$p \vee q$	$\{r \equiv r'\}$	r
$p' \vee q'$	$p \wedge q$	$\{p \equiv p', q \equiv q'\}$	$p \vee q$
$p' \wedge q'$	$p \wedge q$	$\{p \equiv p'\}, \{q \equiv q'\}$	$p \vee q$

Table 2: Skeptical Contraction Examples

A *choice* change represents a feasible way in which a knowledge base can be revised or contracted to incorporate new information. On the other hand, the intersection of all choice changes represents a "safe," *skeptical* means of taking into account all choice changes.

Table 1 gives examples of skeptical revision. The knowledge base is in the first column, but with atoms already renamed. The second column gives the revision formula, while the next lists the maximal consistent EQ set(s); the last column gives the results of the revision, as a finite representation of $Cn(K \dot{+} \alpha)$. For $\{p \wedge q\} \dot{+} (\neg p \vee \neg q)$, there are two maximal consistent EQ sets $\{p \equiv p'\}$ and $\{q \equiv q'\}$ and thus two corresponding choice extensions $Cn(p \wedge \neg q)$ and $Cn(\neg p \wedge q)$, respectively. Table 2 lists four skeptical contraction examples.

The general approach, with $|C| > 1$, can be employed to express *multiple contraction* (Fuhrmann 1988), in which contraction is with respect to a set of (not necessarily mutually consistent) sentences. Therefore, we can use the belief change scenario $(K, \emptyset, \{\alpha, \neg\alpha\})$ to represent a *symmetric contraction* (Katsuno & Mendelzon 1992) of α from K. Refer to (Delgrande & Schaub 2003) for a discussion of the formal properties of these belief revision and contraction operators.

Integrity Constraints. Definition 1 allows for simultaneous revision and contraction by sets of formulas. This in turn leads to a natural and general treatment of integrity constraints. To specify a belief change incorporating a set of *consistency-based* integrity constraints (Kowalski 1978; Sadri & Kowalski 1987), IC_c, and a set of formulas as entailment-based constraints (Reiter 1984), IC_e, one can specify a belief change scenario by $(K, R \cup IC_e, C \cup \overline{IC_c})$, where K, R, and C are as in Defintion 1, and $\overline{IC_c} = \{\neg\phi \mid \phi \in IC_c\}$. See (Delgrande & Schaub 2003) for details.

Implementation Considerations

Finite Representation. Definitions 1–3 provide an abstract characterization of revision and contraction, yielding in each case a deductively-closed belief set. It is proven in (Delgrande & Schaub 2003) that the same (with respect to

logical equivalence) operators can be defined so that they yield a knowledge base consisting of a finite formula. Consider $K \dot{+} \alpha$. Via Definitions 1 and 2, we determine maximal sets EQ where $\{K'\} \cup \{\alpha\} \cup EQ$ is consistent. For each such EQ set, we carry out the substitutions:

- for $p \equiv p' \in EQ$, replace p' with p in K',
- for $p \equiv p' \notin EQ$, replace p' with $\neg p$ in K'.

It is shown that following this substitution, the resulting knowledge base and input formula is logically equivalent to some choice revision; the disjunction of all such resulting knowledge bases and input formula is equivalent to the skeptical revision.

For contraction (where $C \neq \emptyset$), we need to substitute into the resulting K all possible combinations of truth value assignments for all elements in $P_{\overline{EQ}}$. Again, see (Delgrande & Schaub 2003) for details.

Limiting Range of EQ. The range of EQ can be limited to "relevant" atoms. Intuitively, if an atomic sentence appears in a knowledge base K but not in the sentence for revision α, or vice versa, then that atomic sentence plays no part in the revision process. The same intuition extends to contraction. It was proven in (Delgrande & Schaub 2003) that for computing a belief change extension of a belief change extension $B = (K, R, C)$, we need consider only those atoms common to K and to $(R \cup C)$. That is, if $Atoms(X)$ is the set of atoms in set of formulas X, then in Definition 1 for forming K' and the set EQ we can limit ourselves to considering atoms in $Atoms(K) \cap (Atoms(R) \cup Atoms(C))$.

Algorithm

The results at the end of the last section lead to an algorithm for computing a belief change extension for an arbitrary belief change scenario. After presenting our algorithm, we will work through two example belief change scenarios.

Given a set K of formulas in $\mathcal{L}_\mathcal{P}$, and sets Rev, IC_e, Con, and IC_c of formulas in $\mathcal{L}_\mathcal{P}$ for revision, entailment-based integrity constraints, contraction, and consistency-based integrity constraints, respectively, algorithm $ComputeBCE$ returns a formula whose deductive closure is a belief change extension of the belief change scenario $B = (K, Rev \cup IC_e, Con \cup \overline{IC_c})$, where $\overline{IC_c} = \{\neg\phi \mid \phi \in IC_c\}$.

Algorithm $ComputeBCE$ invokes the following auxiliary functions:

$Atoms(S)$ Returns the set of atoms appearing in any formula in set of formulas S.

$Prime(K, CA)$ For set of formulas K and set of atoms CA, returns K but where every atom $p \in A$ is replaced by p'.

$Initialize(K', R, Con, IC_c)$ Given a formula K' and sets R, Con, IC_c of formulas, returns a set of formulas of form $(K' \wedge (\bigwedge R) \wedge \neg\phi \wedge \psi)$, for each $\phi \in (Con \cup \{\bot\})$ and $\psi \in (IC_c \cup \{\top\})$.

$Replace(K', p', p)$ Returns K' with every occurrence of atom p' replaced by p.

$ForgetOutEquiv(K', Out)$ Input: formula K' and a set Out of equivalences of atoms
Output: K' with every atom p such that $(p' \equiv p) \in Out$ is "forgotten":

1. If $Out = \emptyset$, then return K'.
2. $OutAtoms := \{p \mid (p' \equiv p) \in Out\}$.
3. $TA := PowerSet(OutAtoms)$.
   //TA is the set of all truth assignments to $OutAtoms$.
4. $KDisj := \bot$.
5. For each truth assignment $\pi \in TA$ {
   $TempK := K'$.
   $KDisj := KDisj \vee Substitute(TempK, \pi)$. }
   //$Substitute$ returns π applied to $TempK$.
6. Return $KDisj$.

Algorithm $ComputeBCE(K, Rev, IC_e, Con, IC_c)$
Let $R = Rev \cup IC_e$ and $C = Con \cup \overline{IC_c}$.
1. If $R \vdash \bot$ or $K \vdash \bot$, then return $\bot$.
2. If (for any $\psi \in IC_c$, $R \cup \{\psi\} \vdash \bot$), then return $\bot$.
3. If (for any $\phi \in Con$, $R \cup \{\neg\phi\} \vdash \bot$), then return $\bot$.
4. If (for any $\phi \in Con$ and any $\psi \in IC_c$
   $\{\neg\phi\} \cup \{\psi\} \vdash \bot$), then return $\bot$.
5. $CA := Atoms(K) \cap (Atoms(R) \cup Atoms(C))$.
6. $K' := Prime(K, CA)$.
7. $K\overline{RC} := Initialize(K', R, Con, IC_c)$.
8. $In := Out := \emptyset$.
9. For each $e \in \{p' \equiv p \mid p \in CA\}$ {
   If (for any $\theta \in K\overline{RC}$ we have $e \cup \{\theta\} \vdash \bot$)
   Then $Out := Out \cup \{e\}$.
   Else $In := In \cup \{e\}$. }
10. For each $e \in In$: $K' := Replace(K', p', p)$.
11. For each $e \in Out$: $K' := Replace(K', p', \neg p)$.
12. If ($Con \neq \emptyset$) Then $K' := ForgetOutEquiv(K', Out)$.
13. Return $K' \wedge (\bigwedge Rev)$.

Algorithm $ComputeBCE$ generates a belief change extension in non-deterministic polynomial (NP) time; i.e., an extension can be computed by a deterministic polynomial Turing machine using the answers given by an NP oracle. For this purpose, we currently use the SAT-solver called Berkmin in the SAT4J library (SAT4J). The solver performs the consistency checks in lines 1 through 4 and within the for loop in Line 9. Before passing any formula to the solver, we convert it first to conjunctive normal form (CNF).[1] The CNF formula, once created, is saved with its corresponding formula so that conversions are not done repetitively.

The selection function (for the "preferred" EQ set) is left implicit in Line 9 of Algorithm $ComputeBCE$; it is realized by the particular order chosen when treating the atoms in CA. In COBA 2.0, however, we create an ordered (in ascending cardinality) list L of all $2^{|CA|}$ possible subsets of $\{p' \equiv p \mid p \in CA\}$. To help streamline the search for EQ sets and minimize memory usage, we represent each equivalence by a single bit so that it is included in an EQ set e iff its corresponding bit is 1 in e's bit-string. Furthermore, the ordered list L can accommodate our subsequent search

[1] In future, this will be replaced by a structural normal form translation, avoiding the potential exponential blow-up caused by distributivity.

for maximal EQ sets, whether the search be breadth-first or depth-first. On average, the running time and memory usage of breadth-first search is comparable to that of depth-first search, although in our experience neither is consistently superior.

Examples

We illustrate how COBA 2.0 computes belief change extensions by working through two examples. The examples include belief revision and contraction.

Revision. Consider revising a knowledge base $K = \{p, q\}$ by a formula $\alpha = \neg p \vee \neg q$. We show how COBA 2.0 computes $K\dot{+}\alpha$:

1. Find the common atoms between the knowledge base and the revision formula.
   $CA = \{p, q\}$
2. Create a new formula K' from K by priming the common atoms appearing in K.
   $K' = (p' \wedge q')$
3. Find all maximal equivalence sets $EQ = \{b' \equiv b \mid b \in CA\}$ such that $\{K'\} \cup \{\alpha\} \cup EQ$ is satisfiable.
   $EQ_1 = \{p' \equiv p\}$
   $EQ_2 = \{q' \equiv q\}$
4. For each EQ_i, create a belief change extension by (a) unpriming in K' every primed atom p' if $(p' \equiv p) \in EQ_i$, (b) replacing every primed atom p' with $\neg p$ if $(p' \equiv p) \notin EQ_i$, and finally (c) conjoining K' with the revision formula.
   $K\dot{+}_{c_1}\{\alpha\} = (p \wedge \neg q) \wedge (\neg p \vee \neg q) \equiv (p \wedge \neg q)$
   $K\dot{+}_{c_2}\{\alpha\} = (\neg p \wedge q) \wedge (\neg p \vee \neg q) \equiv (\neg p \wedge q)$
5. The resulting knowledge base is the deductive closure of either the disjunction of all belief change extensions for *skeptical* change, or one belief change extension for *choice* change.
   $K\dot{+}\{\alpha\} = Cn((p \wedge \neg q) \vee (\neg p \wedge q))$

Contraction. Consider contracting a knowledge base $K = \{p \vee q\}$ by a formula $\alpha = p \vee q$. We show how COBA 2.0 computes $K\dot{-}\alpha$:

1. Find the common atoms between the knowledge base and the contraction formula.
   $CA = \{p, q\}$
2. Create a new formula K' from K by priming the common atoms appearing in K.
   $K' = (p' \vee q')$
3. Find all maximal equivalence sets $EQ = \{b' \equiv b \mid b \in CA\}$ such that $\{K'\} \cup \{\neg\alpha\} \cup EQ$ is satisfiable.
   $EQ_1 = \{\}$
4. For each EQ_i, create a belief change extension by (a) unpriming in K' every primed atom p' if $(p' \equiv p) \in EQ_i$, (b) replacing every primed atom p' with $\neg p$ if $(p' \equiv p) \notin EQ_i$, and finally (c) taking the disjunction of all possible substitutions of $\top$ or $\bot$ into those atoms in K' that are in CA but whose corresponding equivalences are not in EQ_i.
   $K\dot{-}_{c_1}\{\alpha\} = (\top)$
5. The resulting knowledge base is the deductive closure of either the disjunction of all belief change extensions for $skeptical$ change, or one belief change extension for $choice$ change.
   Here, there is only one resulting knowledge base for skeptical change and for choice change: $K\dot{-}\{\alpha\} = Cn((\neg\bot \vee \neg\bot) \vee (\neg\bot \vee \neg\top) \vee (\neg\top \vee \neg\bot) \vee (\neg\top \vee \neg\top)) = Cn(\top)$

Implementation

In this section, we describe the COBA 2.0 implementation. We discuss features, syntax, and syntactic and consistency checks on input formulas.

Features

COBA 2.0 is available as an interactive Java applet, complete with a menu, text boxes, buttons, and separate panels for belief change, integrity constraints, and snapshots.

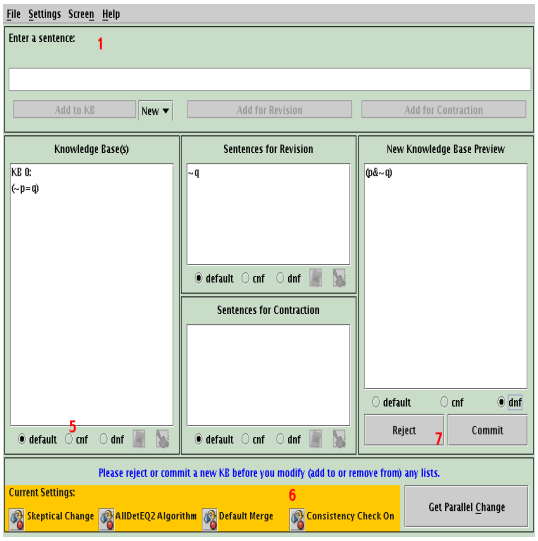


Figure 1: COBA 2.0's Main Screen

Via the menu, users can import belief change scenarios from files, specify the type (skeptical or choice) of belief change desired, and obtain a resulting knowledge base.

Users may also

1. enter belief change scenarios in text boxes,
2. view logs of the changes made to the knowledge base (KB) list, the entailment-based integrity constraints (EB IC) list, and the consistency-based integrity constraints (CB IC) list,
3. revert to an older KB, EB IC, or CB IC snapshot,
4. save any list to an output file,
5. view formulas in CNF or DNF,
6. turn off the various consistency checks,

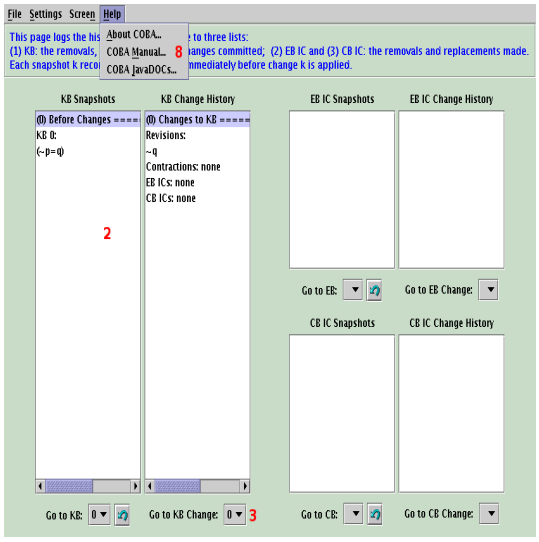


Figure 2: COBA 2.0's History Screen

7. preview, and then reject or commit, a resulting knowledge base, and
8. view the user manual and JavaDocs in external browser windows (if the applet is running in an html document).

COBA 2.0 automatically simplifies formulas where applicable, for example, eliminating occurrences of $\top$ and $\bot$ in subformulas. COBA 2.0 also automatically informs users of any syntactically ill-formed input formulas. The consistency checks in 6. above and the syntax checks are elaborated on in Subsection 4.3. The applet, user manual, Java code, and Javadocs of COBA 2.0 are accessible from (COBA 2.0).

Syntax

COBA 2.0 accepts almost all alphanumerical strings for atom names. The exceptions are the symbols in the following list: `'`, `+`, `&`, `^`, `~`, `=`, `>`, (and). Note that `T` and `F` stand for $\top$ and $\bot$, respectively.

More complex formulas can be built from formulas A and B using connectives.

- `~A` for the negation of A
- `(A&B)` for the conjunction of A and B
- `(A+B)` for the disjunction of A and B
- `(A>B)` for A implies B
- `(A=B)` for A is equivalent to B

A top-level formula with a binary connective (`&`, `+`, `>`, or `=`) must be enclosed in parentheses. Parentheses within a formula, however, are optional and are used only to enforce precedence. For example, `(a&b+c)` is a valid input sentence and is different from `(a&(b+c))`, whereas a top-level sentence like `a&b` is syntactically illformed.

Encoding Input Files. Input file formats (for belief change scenarios) vary according to the list (KB, Revision, Contraction, EB IC, or CB IC) to which formulas are to be added. Any KB file to be loaded should precede each knowledge base by a line "KB :" (without the double quotes) and list each formula on a separate line. Each formula is listed on a separate line in any Revision and EB IC input files. For any contraction and CB IC input files, each line is interpreted as an independent formula for contraction and as a CB IC, respectively.

KB	Rev	Cont	EB IC	CB IC
KB : (p&q&r) (~q+~s)	q ~p	p ~q	(a&b+c) (x&(y+z))	d ~d

Table 3: Example Input Files

Consider an example contraction file. While the formula `(p&~q)` means that $(p \& \neg q)$ is to be removed from the consequences of the resulting knowledge base, `p` / `~q` listed on two separate lines means that both p and $\neg q$ are to be dropped from the consequences of the resulting knowledge base.

As an example, Table 2 shows some valid input files.

Input Checks

COBA 2.0 performs syntax and consistency checks on all input formulas. The former checks are always enforced, while the latter checks are optional but carried out by default. See below for details.

Syntax Checks. With regard to the syntax detailed earlier in Subsection 4.2, COBA 2.0 informs users of ill-formed input formulas. Thus, for example, the following ill-formed input strings would be flagged with an appropriate message: `q)`, `q+`, `p^`, `p'`, `(p`, `(p&(q)`, `(p+q&)`, and `(+q)`.

Consistency Checks. To preempt inconsistent belief change scenarios, COBA 2.0 prohibits certain kinds of input formulas that result in inconsistent belief change scenarios. This preemptive measure accords well with the consistency checks in lines 1 through 4 of Algorithm $ComputeBCE$ in Section 3. Automatic consistency checks on input formulas, although carried out by default, can be optionally disabled by users wishing to speed up computations. One caveat is that, if these checks are disabled, `F` might be obtained as the resulting knowledge base.

Let $(\bigwedge Rev)$ denote the conjunction of all formulas in Rev for revision, $(\bigwedge EBIC)$ the conjunction of all entailment-based integrity constraints. The following inconsistent belief change scenarios should be avoided; sample error messages, where applicable, are italicised.

1. a contradiction in Rev: *The conjunction of revisions is inconsistent!*
2. a contradiction in $EBIC$: *The conjunction of EB ICs is a contradiction!*
3. a contradiction as a KB, revision, or EB IC formula: No error message; sentence not added.

4. a tautology as a contraction formula: No error message; sentence not added.
5. a contradiction as a CB IC formula: No error message; sentence not added.
6. conflict between $(\bigwedge Rev)$ and $(\bigwedge EBIC)$: *The conjunction of revisions is inconsistent with the conjunction of EB ICs!*
7. conflict between $(\bigwedge Rev)$ and contraction formulas: *The contraction indexed 0 is inconsistent with the conjunction of revisions (indexing starts at 0)!*
8. conflict between $(\bigwedge Rev)$ and CB IC formulas: *The CB IC indexed 1 is inconsistent with the conjunction of revisions (indexing starts at 0)!*
9. conflict between $(\bigwedge EBIC)$ and contraction formulas: *The contraction indexed 6 is inconsistent with the conjunction of EB ICs (indexing starts at 0)!*
10. conflict between $(\bigwedge EBIC)$ and CB IC formulas: *The CB IC indexed 3 is inconsistent with the conjunction of EB ICs (indexing starts at 0)!*
11. conflicting pairs of CB IC formulas and contraction formulas: *The contraction indexed 2 is inconsistent with the CB IC indexed 0 (indexing starts at 0)!*

The aforementioned consistency checks correspond to the consistency checks on input in Algorithm $ComputeBCE$ from Section 3. Specifically, 1., 2., 3., and 6. correspond to the checks $(R \vdash \bot)$ and $(K \vdash \bot)$ in Line 1 of $ComputeBCE$; 5., 8., and 10. to the check $(R \cup \{\psi\} \vdash \bot$, for any $\psi \in IC_c)$ in Line 2 of $ComputeBCE$; 4., 7., and 9. to the check $(R \cup \{\neg\phi\} \vdash \bot$, for any $\phi \in Con)$ in Line 3 of $ComputeBCE$; lastly, 11. to the check $(\{\neg\phi\} \cup \{\psi\} \vdash \bot$, for any $\phi \in Con$ and any $\psi \in IC_c)$ in Line 4 of $ComputeBCE$.

Experiments

It has been shown that skeptical revision and contraction are Π_2^P-hard problems (Delgrande & Schaub 2003). In (Delgrande *et al.* 2004) is was shown how the approach could be encoded using quantified Boolean formulas (QBF). This allows us to compare COBA 2.0 with an implemented version of the approach using the quantified Boolean formula solver QUIP (Egly *et al.* 2000).

For comparing the implementations, we created knowledge bases and revision sentences made up of randomly generated 3-DNF formulas, and converted each to a QBF. We also devised an experimental prototype of COBA 2.0 which performs structural transformation (by replacing subformulas with new atoms) instead of the CNF conversion of formulas (for consistency checks). Experiments were then conducted on QUIP, and on both the stable version (the applet) and the experimental prototype of COBA 2.0.

Preliminary experimental results reveal that most of COBA 2.0's run-time is attributed to its structural or CNF conversion of formulas and to its consistency checks. The run-time of all three implementations shows an exponential growth rate. QUIP, however, is relatively faster than both versions of COBA 2.0. The experimental prototype seems to be more than two orders of magnitude faster than the stable version of COBA 2.0, and this observation suggests that structural transformation be done in lieu of CNF conversion in our future implementation.

Conclusion

We have presented COBA 2.0, an implementation of a consistency-based approach for belief change incorporating integrity constraints. Operators for belief revision and contraction incorporating integrity constraints are readily defined in a general framework that satisfys the majority of the AGM postulates. As demonstrated by COBA 2.0, the framework is easily implementable, for the results of our operators are finite and vocabulary-restricted belief change can be performed instead. Examples of how COBA 2.0 computes belief change are detailed in Section 3.

Our preliminary experiments show that our stable version (the applet) still has much potential for improvement. To this end, we devised an experimental prototype (with structural transformation in lieu of CNF conversion) that seems to be more than two orders of magnitude faster than the stable version (with CNF conversion). Hence, we are optimistic that COBA 2.0 can be improved to achieve a similar run-time behaviour as the monolithic QUIP system.

To our knowledge, COBA 2.0 is the most general belief change system currently available, capable of computing arbitrary combinations of belief revision and contraction that (possibly) incorporate consistency-based and entailment-based integrity constraints. Moreover, COBA 2.0's general framework is easily extensible to consistency-based merging operators as detailed in (Delgrande & Schaub 2004), and currently we are refining our implementation so as to accommodate the merging of knowledge bases.

The applet, user manual, Java code, and Javadocs of COBA 2.0 are all accessible from (COBA 2.0).

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic* 50(2):510–530.

`http://www.cs.sfu.ca/~cl/software/-COBA/coba2.html`.

Delgrande, J., and Schaub, T. 2003. A consistency-based approach for belief change. *Artificial Intelligence* 151(1-2):1–41.

Delgrande, J., and Schaub, T. 2004. Consistency-based approaches to merging knowledge bases. In Delgrande, J., and Schaub, T., eds., *Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning (NMR 2004)*, 126–133.

Delgrande, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2004. On computing belief change operations using quantified boolean formulas. *Journal of Logic and Computation* 14(6):801–826.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000. Solving advanced reasoning tasks using quantified Boolean

formulas. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 417–422.

Fuhrmann, A. 1988. *Relevant Logics, Modal Logics, and Theory Change*. Ph.D. Dissertation, Australian National University, Australia.

Katsuno, H., and Mendelzon, A. 1992. On the difference between updating a knowledge base and revising it. In Gärdenfors, P., ed., *Belief Revision*, 183–203. Cambridge University Press.

Kowalski, R. 1978. Logic for data description. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum Press. 77–103.

Reiter, R. 1984. Towards a logical reconstruction of relational database theory. In Brodie, M.; Mylopoulos, J.; and Schmidt, J., eds., *On Conceptual Modelling*. Springer-Verlag. 191–233.

Sadri, F., and Kowalski, R. 1987. A theorem-proving approach to database integrity. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann Publishers. chapter 9, 313–362.

A satisfiability library for java. `http://www.sat4j.org`.

3.4 Modelling biological networks by action languages via answer set programming

Modelling biological networks by action languages via answer set programming

Susanne Grell and **Torsten Schaub**
Institut für Informatik
Universität Potsdam
Postfach 900327
D-14439 Potsdam, Germany

Joachim Selbig
Institut für Informatik und Institut für Biologie/Biochemie
Universität Potsdam
Postfach 900327
D-14439 Potsdam, Germany

Abstract

We describe an approach to modelling biological networks by action languages via answer set programming. To this end, we propose an action language for modelling biological networks, building on previous work by Baral et al. We introduce its syntax and semantics along with a translation into answer set programming. Finally, we describe one of its applications, namely, the sulfur starvation response-pathway of the model plant Arabidopsis thaliana and sketch the functionality of our system and its usage.

Introduction

Molecular biology has seen a technological revolution with the establishment of high-throughput methods in the last years. These methods allow for gathering multiple orders of magnitude more data than was procurable before. For turning such huge amounts of data into knowledge, one needs appropriate and powerful knowledge representation tools that allow for modelling complex biological systems and their behaviour. Of particular interest are qualitative tools that allow for dealing with biological and biochemical networks. Since these networks are very large, a biologist can manually deal with a small part of it at once. Among the currently used, more traditional formalisms for the qualitative modelling of biological networks, we find e.g. Petri Nets (Reddy, Mavrovouniotis, & Liebman 1993; Pinney, Westhead, & McConkey 2003), Flux Balance Analysis (Bonarius, Schmid, & Tramper 1997) or Boolean Networks (Shmulevich *et al.* 2002). As detailed in (Baral *et al.* 2004), these approaches lack sufficiently expressive reasoning capacities, like explanation and planning.

Groundbreaking work addressing this deficiency was recently done by Chitta Baral and colleagues who developed a first *action language* for representing and reasoning about biological networks (Tran & Baral 2004; Baral *et al.* 2004). Action languages were introduced in the 1990s by Gelfond and Lifschitz (cf. (Gelfond & Lifschitz 1993)). By now, there exists a large variety of action languages, like the most basic language $\mathcal{A}$ and its extensions (Gelfond & Lifschitz 1998) as well as more expressive action languages like $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) or $\mathcal{K}$ (Eiter *et al.* 2000). Traditionally, action languages are designed for applications in autonomous agents, planning, diagnosis, etc, in which the explicit applicability of actions plays a dominant role. This is slightly different in biological systems where *reactions* are a major concern. For instance, while an agent usually has the choice to execute an action or not, a biological reaction is often simply triggered by its application conditions. This is addressed in (Baral *et al.* 2004) by proposing *trigger* and *inhibition rules* as an addition to the basic action language $\mathcal{A}$; the resulting language is referred to as $\mathcal{A}^0_T$. A further extension, allowing knowledge about event ordering, is introduced in (Tran, Baral, & Shankland 2005).

The advantages of action languages for modelling biological systems are manifold:

- We get a simplified model. It is not necessary to have any kinetic parameters. The approach can thus already be used in a very early state to verify whether the proposed model of the biological system can or cannot hold.
- Different kinds of reasoning can be used to plan and support experiments. This helps to reduce the number of expensive experiments.
- Further reasoning modes allow for prediction of consequences and explanation of observations.
- The usage of static causal laws allows to easily include background knowledge like environmental conditions, which play an important role for the development of a biological system but are usually difficult to include in the model.
- The approach is elaboration tolerant because it allows to easily extend the model without requiring to change the rest of the model.

We start by introducing our action language $\mathcal{C}_{TAID}$ by building on language $\mathcal{A}^0_T$ (Baral *et al.* 2004) and $\mathcal{C}$ (Giunchiglia & Lifschitz 1998). $\mathcal{C}_{TAID}$ extends $\mathcal{C}$ by adding biologically relevant concepts from $\mathcal{A}^0_T$ such as triggers and it augments $\mathcal{A}^0_T$ by providing static causal laws for modelling background knowledge. Moreover, fluents are no longer inertial by definition and the concurrent execution of actions can be restricted. A feature distinguishing $\mathcal{C}_{TAID}$ from its predecessors is its concept of *allowance*, which was motivated by our biological applications. The corresponding *allowance rules* let us express that an action can occur under certain conditions but does not have to occur. In fact, biological systems are characterised by a high degree of incomplete knowledge about the dependencies among differ-

ent component and the actual reasons for their interaction. If the dependencies are well understood, they can be expressed using triggering rules. However, if the dependencies are only partly known or not part of the model, e.g. environmental conditions, they cannot be expressed appropriately using triggering rules. The concept of allowance permits actions to take place or not, as long as they are allowed (and not inhibited). This introduces a certain non-determinism that is used to model alternative paths, actions for which the preconditions are not yet fully understood, and low reaction rates. Of course, such a non-deterministic construct increases the number of solutions. However, this is a desired feature since we pursue an exploratory approach to bioinformatics that allows the biologist to browse through the possible models of its application.

We introduce the syntax and semantics of $\mathcal{C}_{TAID}$ and give a soundness and completeness result, proved in (Grell 2006). For implementing $\mathcal{C}_{TAID}$, we have developed a compilation technique that maps a specification in $\mathcal{C}_{TAID}$ into logic programs under answer set semantics (Baral 2003). This has been implemented in Java and was used meanwhile in ten different application scenarios at the Max-Planck Institute for Molecular Plant Physiology for modelling metabolic as well as signal transduction networks. Among them we present the smallest application, namely the sulfur starvation response-pathway of the model plant Arabidopsis thaliana.

Action Language $\mathcal{C}_{TAID}$

The alphabet of our action language $\mathcal{C}_{TAID}$ consists of two nonempty disjoint sets of symbols: a set of *actions* A and a set of *fluents* F. Informally, fluents describe changing properties of a world and actions can influence fluents. In what follows, we deal with propositional fluents that can either be *true* or *false*. A *fluent literal* is a fluent $f \in F$ possibly preceded by $\neg$.

We distinguish three sublanguages of $\mathcal{C}_{TAID}$: The *action description language* is used to describe the general knowledge about the system, the *action observation language* is used to express knowledge about particular points of time and the *action query language* is used to reason about the described system.

Action Description Language. To begin with, we fix the syntax of $\mathcal{C}_{TAID}$'s action description language:

Definition 1 *A domain description $D(A, F)$ in $\mathcal{C}_{TAID}$ consists of expressions of the following form:*

$$(a \textbf{ causes } f_1, \ldots, f_n \textbf{ if } g_1, \ldots, g_m) \quad (1)$$
$$(f_1, \ldots, f_n \textbf{ if } g_1, \ldots, g_m) \quad (2)$$
$$(f_1, \ldots, f_n \textbf{ triggers } a) \quad (3)$$
$$(f_1, \ldots, f_n \textbf{ allows } a) \quad (4)$$
$$(f_1, \ldots, f_n \textbf{ inhibits } a) \quad (5)$$
$$(\textbf{noconcurrency } a_1, \ldots, a_n) \quad (6)$$
$$(\textbf{default } f) \quad (7)$$

where $a, a_1, \ldots, a_n \in A$ are a actions and $f, f_1, \ldots, f_n, g_1, \ldots, g_m \in F$ are fluent literals.

Note that $\mathcal{A}_T^0$ consists of expressions of form (1), (3), and (5) only.

A *dynamic causal law* is a rule of form (1), stating that $f_1, \ldots, f_n$ hold after the occurrence of action a if $g_1, \ldots, g_m$ hold when a occurs. If there are no preconditions of the form $g_1, \ldots, g_m$, the if-part can be omitted. Rule (2) is a *static causal law*, used to express immediate dependencies between fluents. It guarantees that $f_1, \ldots, f_n$ hold whenever $g_1, \ldots, g_m$ hold. To express whether and when an action can or cannot occur rules (3) to (6) can be used. A *triggering rule* (3) is used to state that action a occurs immediately if the preconditions $f_1, \ldots, f_n$ hold, unless it is inhibited. An *allowance rule* of form (4) states that action a can but need not occur if the preconditions $f_1, \ldots, f_n$ hold. An action for which triggering or allowance rules are specified can only occur if one of its triggering or allowance rules, resp., is satisfied. An *inhibition rule* of form (5) can be used to express that action a cannot occur if $f_1, \ldots, f_n$ hold. A rule of the form (6) is a no-concurrency constraint. Actions included in such a constraint cannot occur at the same time. Rule (7) is a *default rule*, which is used to define a default value for a fluent. This makes us distinguish two kinds of fluents: inertial and non-inertial fluents. Inertial fluent change their value only if they are affected by dynamic or static causal laws. Non-inertial fluents on the other hand have the value, specified by a default rule, unless they are affected by a dynamic or static causal law. Every fluent that has no default value is regarded to be inertial. Additionally, we distinguish three groups of actions depending on the rules defined for them. An action can either be a triggered, an allowed or an exogenous action. That means, for one action there can be several triggering or several allowance rules but not both.

As usual, the semantics of a domain description $D(A, F)$ is defined in terms of transition systems. An *interpretation* I of F is a complete and consistent set of fluents, i.e. for every fluent $f \in F$ either $f \in I$ or $\neg f \in I$.

Definition 2 (State) *A state $s \in S$ of the domain description $D(A, F)$ is an interpretation of F such that for every static causal law $(f_1, \ldots, f_n \textbf{ if } g_1, \ldots, g_n) \in D(A, F)$, we have $\{f_1, \ldots, f_n\} \subseteq s$ whenever $\{g_1, \ldots, g_n\} \subseteq s$.*

Hence, we are only interested in sets of fluents satisfying all static causal laws, i.e. correctly model the dependencies between the fluents.

Depending on the state, it is possible to decide which actions can or cannot occur. Therefore we define the notion of active, passive and applicable rules.

Definition 3 *Let $D(A, F)$ be a domain description and s a state of $D(A, F)$.*

1. *An inhibition rule ($f_1, \ldots, f_n$ **inhibits** a) is active in s, if $s \models f_1 \wedge \ldots \wedge f_n$, otherwise the inhibition rule is passive. The set $A_I(s)$ is the set of actions for which there exists at least one active inhibition rule in s.*
2. *A triggering rule ($f_1, \ldots, f_n$ **triggers** a) is active in s, if $s \models f_1 \wedge \ldots \wedge f_n$ and all inhibition rules of action a are passive in s, otherwise the triggering rule is passive in s. The set $A_T(s)$ is the set of actions for which there exists at least one active triggering rule in s. The set $\overline{A}_T(s)$ is the set of actions for which there exists at least one triggering rule and all triggering rules are passive in s.*

3. *An allowance rule ($f_1, \ldots, f_n$* **allows** *a) is active in s, if $s \models f_1 \wedge \ldots \wedge f_n$ and all inhibition rules of action a are passive in s, otherwise the allowance rule is passive in s. The set $A_A(s)$ is the set of actions for which there exists at least one active allowance rule in s. The set $\overline{A}_A(s)$ is the set of actions for which there exists at least one allowance rule and all allowance rules are passive in s.*
4. *A dynamic causal law (a* **causes** *$f_1, \ldots, f_n$* **if** *$g_1, \ldots, g_n$) is applicable in s, if $s \models g_1 \wedge \ldots \wedge g_n$.*
5. *A static causal law ($f_1, \ldots, f_n$* **if** *$g_1, \ldots, g_n$) is applicable in s, if $s \models g_1 \wedge \ldots \wedge g_n$.*

Observe that point two and three of the definition express that an action has to occur or may occur as long as there is one active triggering or allowance rule respectively. An action cannot occur if either an inhibition rule for the action is active or if all triggering or allowance rules for the action are passive.

The effects of an action are determined by the applicable dynamic causal laws defined for this action. Following (Gelfond & Lifschitz 1998), the effects of an action a in a state s of domain description $D(A, F)$ are defined as follows:

$$E(a, s) = \{f_1, \ldots, f_n \mid (a \textbf{ causes } f_1, \ldots, f_n \textbf{ if } g_1, \ldots, g_m) \text{ is applicable in } s\}$$

The effects of a set of actions A is defined as the union of the effects of the single actions: $E(A, s) = \bigcup_{a \in A} E(a, s)$. Besides the direct effects of actions, a domain description also defines the consequences of static relationships between fluents. For a set of static causal laws in a domain description $D(A, F)$ and a state s, the set

$$L(s) = \{f_1, \ldots, f_n \mid (f_1, \ldots, f_n \textbf{ if } g_1, \ldots, g_m) \text{ is applicable in } s\}$$

contains the heads of all static causal laws whose preconditions hold in s.

Finally, the way the world evolves according to a domain description is captured by a *transition relation*. It defines to which state the execution of a set of actions leads.

Definition 4 *Let $D(A, F)$ be a domain description and S be the set of states of $D(A, F)$. Then, the transition relation $\Phi \subseteq S \times 2^A \times S$ determines the resulting state after executing all actions $B \subseteq A$ in state $s \in S$ as follows:*

- $(s, B, s') \in \Phi$, *if* $s' \in S$ *for* $s' = \{(s \cap s') \cup E(B, s) \cup L(s') \cup \Delta(s')\}$ *where*

$$\Delta(s') = \{ f \mid (\textbf{default } f) \in D(A, F), \neg f \notin E(B, s) \cup L(s')\} \cup \{\neg f \mid (\textbf{default } \neg f) \in D(A, F), f \notin E(B, s) \cup L(s')\}$$

Even if no actions are performed, there can nevertheless be a change of state due to the default values defined by the domain description. Intuitively, if actions occur, the next state is determined by taking all effects of the applicable dynamic and static causal laws and adding the default values of fluents not affected by these actions. The values of all fluents that are not affected by these actions or by default values remain unchanged.

The transition relation determines the resulting state when an action is executed, but it cannot be used to decide whether the action happens at all, since it does not consider triggering, allowance or inhibition rules. This is accomplished by the concept of a *trajectory*, which is a sequence of states and actions that takes all rules in the domain description into account.

Definition 5 (Trajectory) *Let $D(A, F)$ be a domain description.*

A trajectory $s_0, A_1, s_1, \ldots, A_n, s_n$ of $D(A, F)$ is a sequence of actions $A_i \subseteq A$ and states s_i satisfying the following conditions for $0 \leq i < n$:

1. $(s_i, A, s_{i+1}) \in \Phi$
2. $A_T(s_i) \subseteq A_{i+1}$
3. $\overline{A}_T(s_i) \cap A_{i+1} = \emptyset$
4. $\overline{A}_A(s_i) \cap A_{i+1} = \emptyset$
5. $A_I(s_i) \cap A_{i+1} = \emptyset$
6. $|A_i \cap \{a \mid a \in B\}| \leq 1$ *for all* $(\textbf{noconcurrency } B) \in D(A, F)$.

A trajectory assures that there is a reason why an action occurs or why it does not occur. The second and third point of the definition make sure that the actions of all active triggering rules are included in the set of actions and that no action for which all triggering rules are passive is included in the set of actions. Point four and five assure that no actions for which all allowance rules are passive and no inhibited actions are included in the set of actions. The definition does not include assertions about the active allowance rules, because they can be, but not necessarily have to be, included in the set of actions. (As detailed above, this is motivated by our biological application.) Point two to four imply that for an action there can either be only triggering rules or only allowance rules defined. The last point of the definition assures that all no-concurrency constraints are correctly applied.

Action Observation Language. The action observation language provides expressions to describe particular states and occurrences of actions:

$$(f \textbf{ at } t_i) \qquad (a \textbf{ occurs\_at } t_i) \tag{8}$$

where f is a fluent literal, a is an action and t_i is a point of time. The initial point of time is t_0. For a set of actions $A' = \{a_1, \ldots, a_k\}$ we write *(A'* **occurs_at** *t_i)* to abbreviate *(a_1* **occurs_at** *t_i), ..., (a_k* **occurs_at** *t_i)*. Intuitively, an expression of form $(f \textbf{ at } t_i)$ is used to state that a fluent f is *true* or present at time t_i. If the fluent f is preceded by $\neg$ it states that f is *false* or not present at t_i. An observation of form $(a \textbf{ occurs\_at } t_i)$ says that action a occurs at time t_i. It is possible that action a is preceded by $\neg$ to express that a does not occur at time t_i.

A domain description specifies how the system can evolve over time. By including observations the possibilities of this evolution are restricted. So only when all information, the

domain description and the observations, is taken into account, we get an appropriate picture of the world. The combination of domain description and observations is called an *action theory*.

Definition 6 (Action theory) *Let D be a domain description and O be a set of observations. The pair (D, O) is called an action theory.*

Intuitively, trajectories specify possible evolutions of the system with respect to the given domain description. However, not all trajectories satisfy the observations given by an action theory. Trajectories satisfying both, the domain description as well as given observations, are called *trajectory models*:

Definition 7 (Trajectory model) *Let (D, O) be an action theory.*

A trajectory $s_0, A_1, s_1, A_2, \ldots, A_n, s_n$ of D is a trajectory model of (D, O), if it satisfies all observations in O in the following way:

- *if $(f$ **at** $t) \in O$, then $f \in s_t$*
- *if $(a$ **occurs_at** $t) \in O$, then $a \in A_{t+1}$.*

The problem that arises here is to find biologically meaningful models. Obviously, such trajectory models often include redundant information, but since this is a common phenomena of biological systems it is not possible to simply exclude such trajectory models. Often, only the minimal trajectories are considered to be of interest, but this is not appropriate for biological systems, since we are not only interested in the shortest path through the transition system, but also in, possibly longer, alternative paths and just as well in models which include the concurrent execution of actions. To decide which actions are redundant is thus a rather difficult problem and the question whether a model is biologically meaningful can only be answered by a biologist, not by an automated reasoner. One way to include additional information which may be derived from data on measurement could be the use of preferences, which is subject to future work.

A question we can already answer is the question of the logical consequence of observations.

Definition 8 *Let (D, O) be an action theory. Then,*

- *(D, O) entails fluent observation $(f$ **at** $t_i)$, written $(D, O) \models (f$ **at** $t_i)$, if $f \in s_i$ for all trajectory models $s_0, A_1, \ldots, s_i, A_{i+1}, \ldots, A_n, s_n$ of (D, O),*
- *(D, O) entails action observation $(a$ **occurs_at** $t_i)$, written $(D, O) \models (a$ **occurs_at** $t_i)$, if $a \in A_{i+1}$ for all trajectory models $s_0, A_1, \ldots, s_i, A_{i+1}, \ldots, A_n, s_n$ of (D, O).*

Action Query Language. Queries are about the evolution of the biological system, i.e. about trajectories. In general, a query is of the form:

$$(f_1, \ldots, f_n \textbf{ after } A_1 \textbf{ occurs\_at } t_1, \ldots, A_m \textbf{ occurs\_at } t_m) \quad (9)$$

where $f_1, \ldots, f_n$ are fluent literals, $A_1, \ldots, A_m$ are sets of actions, and $t_1, \ldots, t_m$ are time points.

For queries the most prominent question is the notion of logical consequence. Under which circumstances entails an action theory or a single trajectory model a query.

Definition 9 *Let (D, O) be an action theory and Q be a query of form (9). Then,*

- *Q is cautiously entailed by (D, O), written $(D, O) \models_c Q$, if every trajectory model $s_0, A'_1, s_1, A'_2, \ldots, A'_p, s_p$ of (D, O) satisfies $A_i \subseteq A'_i$ for $0 < i \leq m \leq p$ and $s_p \models f_1 \wedge \ldots \wedge f_n$.*
- *Q is bravely entailed by (D, O), written $(D, O) \models_b Q$, if some trajectory model $s_0, A'_1, s_1, A'_2, \ldots, A'_p, s_p$ of (D, O) satisfies $A_i \subseteq A'_i$ for $0 < i \leq m \leq p$ and $s_p \models f_1 \wedge \ldots \wedge f_n$.*

While cautiously entailed queries are supported by all models, bravely entailed queries can be used for checking the possible hypotheses.

We want to use the knowledge given as an action theory to reason about the corresponding biological system. Reasoning includes explaining observed behaviour, but also predicting the future development of the system or how the system may be influenced in a particular way. The above notion of entailment is used to verify the different queries introduced in the next sections.

Planning. In planning, we try to find possibilities to influence a system in a certain way. Neither the initial state nor the goal state have to be completely specified by fluent observations. A plan is thus a sequence of actions starting from one possible initial state and ending at one possible goal state. There are usually several plans, taking into account different paths but also different initial and goal states.

Definition 10 (Plan) *Let (D, O_{init}) be an a action theory such that O_{init} contains only fluent observations about the initial state and let Q be a query of form (9).*

*If $(D, O_{init}) \models_b Q$, then $P = \{(A_1$ **occurs_at** $t_1), \ldots, (A_m$ **occurs_at** $t_m)\}$ is a plan for $f_1, \ldots, f_n$.*

Note that a plan is always derived from the corresponding trajectory model.

Explanation. Usually, there are not only observations about the initial state but also about other points of time and often we are more interested in understanding the observed behaviour of a system than in finding a plan to cause certain behaviour of the system.

Definition 11 (Explanation) *Let (D, O) be an action theory and let Q be a query of form (9) where $(f_1, \ldots, f_n) = true$.*

*If $(D, O) \models_b Q$, then $E = \{(A_1$ **occurs_at** $t_1), \ldots, (A_m$ **occurs_at** $t_m)\}$ is an explanation for the set of observations O.*

When explaining observed behaviour it is neither necessary to completely define the initial state, nor the final state. The less information is provided the more possible explanation there are, because an explanation is one path from one possible initial state to one possible final state, via some possible intermediate partially defined states given by the observations. The initial state and the explanation are defined by the corresponding trajectory model.

Prediction is mainly used to determine the influence of actions on the system; it tries to answer questions about the

possible evolution of the system. A query answers the question whether, starting at the current state and executing a given sequence of actions, fluents will hold or not hold after a certain time.

Definition 12 (Prediction) *Let (D, O) be an action theory and let Q be a query of form (9).*

- *If $(D, O) \models_c Q$, then $f_1, \ldots, f_n$ are cautiously predicted,*
- *If $(D, O) \models_b Q$, then $f_1, \ldots, f_n$ are bravely predicted.*

All of the above reasoning modes are implemented in our tool and used in our biological applications. Before describing its usage, we first detail how it is implemented.

Compilation

We implemented our action language by means of a compiler mapping $\mathcal{C}_{TAID}$ onto logic programs under *answer set semantics* (cf. (Gelfond & Lifschitz 1991; Baral 2003)). This semantics associates with a logic program a set of distinguished models, referred to as *answer sets*. This model-based approach to logic programming is different from the traditional one, like Prolog, insofar as solutions are read off issuing answer sets rather than proofs of posed queries. Our compiler uses efficient off-the-self answer set solvers as a back-end, whose purpose is to compute answer sets from the result of our compilation. Since we do not elaborate upon theoretical aspects of this, we refer the reader to the literature for a formal introduction to answer set programming (cf. (Gelfond & Lifschitz 1991; Baral 2003)).

Our translation builds upon and extends the one in (Tran & Baral 2004). We adapt the translation of the language $\mathcal{A}_T^0$ to include new language constructs and we extend the compilation of $\mathcal{A}_T^0$ in order to capture the semantics of static causal laws, allowance and default rules, and of no-concurrency constraints. In what follows, we stick to the syntax of the `smodels` system, using lowercase strings for predicate, function, and constant symbols and uppercase strings for variables.

Action description language. The expressions defined in a domain description $D(A, F)$ have to be composed of symbols from A an F. When constructing the logic program for $D(A, F)$, we first have to define the alphabet. We declare every fluent $f \in F$ and action $a \in A$, resp., by adding a fact of the form `fluent(f)`, and `action(a)`. We use continuously a variable `T`, representing a time point where $0 \leq$ `T` $\leq t_{max}$. This range is encoded by the `smodels` construct `time(0..`t_{max}`)`, standing for the facts `time(0),...,time(`t_{max}`)`. Furthermore, it is necessary to add constraints expressing that f and $\neg f$ are contradictory.

```
:- holds(f,T), holds(neg(f),T), fluent(f),
   time(T).
```

Whenever clear from the context, we only give translations for positive fluent literals $f \in F$ and omit the dual rule for the negative fluent, viz. $\neg f$ represented as `neg(f)`.

For each inertial fluent $f \in F$, we include rules expressing that f has the same value at t_{i+1} as at t_i, unless it is known otherwise:

```
holds(f,T+1) :- holds(f,T),
   not holds(neg(f,T+1)),not default(f),
   fluent(f),time(T),time(T+1).
```

For each non-inertial fluent $f \in F$, we add the fact `default(f)` and include for the default value $true$:

```
holds(f,T) :- not holds(neg(f),T),
   fluent(f), time(T).
```

For each dynamic causal law (1) in $D(A, F)$ and each fluent $f_i \in F$, we include:

```
holds(fi,T+1) :- holds(occurs(a),T),
   holds(g1,T),...,holds(gn,T),
   fluent(g1),...,fluent(gn),fluent(fi),
   action(a),time(T),time(T+1).
```

For each static causal law (2) in $D(A, F)$ and each fluent $f_i \in F$, we include:

```
holds(fi,T) :- holds(g1,T),...,holds(gn,T),
   fluent(g1), ..., fluent(gn),
   fluent(fi), time(T).
```

Every triggering rule (3) in $D(A, F)$ is translated as:

```
holds(occurs(a),T) :-
   not holds(ab(occurs(a)),T),
   holds(f1,T),...,holds(fn,T),
   fluent(f1),...,fluent(fn),
   action(a),time(T).
```

For each allowance rule (4) in $D(A, F)$, we include:

```
holds(allow(occurs(a)),T) :-
   not holds(ab(occurs(a)),T),
   holds(f1,T),...,holds(fn,T),
   fluent(f1),...,fluent(fn),
   action(a),time(T).
```

For every exogenous action $a \in A$, the translation includes a rule, stating that this action can always occur.

```
holds(allow(occurs(a)),T) :- action(a),
   time(T).
```

Every inhibition rule (5) in $D(A, F)$ is translated as:

```
holds(ab(occurs(a)),T) :-
   holds(f1,T),...,holds(fn,T),
   fluent(f1),...,fluent(fn),
   action(a), time(T).
```

For each no-concurrency constraint (6) in $D(A, F)$, we include an integrity constraint assuring that at most one of the respective actions can hold at time t:

```
:- 2 {holds(occurs(a1),T):action(a1),...,
   holds(occurs(an),T):action(an)},time(T).
```

Action observation language. There are two different kinds of fluent observations. On the one hand, those about the initial state, *(f* **at** *t_0)*, and on the other hand, the fluent observations about all other states, *(f* **at** *t_i)* for $i > 0$. Fluent observations about the initial state are simply translated as facts: `holds(f,0)`. Because they are just assumed to be true and need no further justification. All other fluent observations however need a justification. Due to this, fluent observations about all states except the initial state are

translated into integrity constraints of the form: `:- not holds(f,T),fluent(f),time(T).`

The initial state can be partially specified by fluent observations. In fact, only the translation of the (initial) fluent observations must be given. All possible completions of the initial state are then generated by adding for every fluent $f \in F$ the rules:

```
holds(f,0):- not holds(neg(f),0).
holds(neg(f),0):- not holds(f,0).
```
(10)

When translating action observations of form (8) the different kinds of actions have to be considered. Exogenous actions can always occur and need no further justification. Such an exogenous action observation is translated as a fact: `holds(occurs(a),T).` Unlike this, observations about triggered or allowed actions must have a reason, e.g. an active triggering or allowance rule, to occur. To assure this justification, the action observation is translated using constraints of the form:

```
:-
    holds(neg(occurs(a)),T),action(a),time(T).
```

Such a constraint assures that every answer set must satisfy the observation *(a **occurs_at** t_i)*.

Apart from planning (see below), we also have to generate possible combinations of occurrences of actions, for all states. To this effect, the translation includes two rules for every exogenous and allowed action.

```
holds(occurs(a),T) :-
   holds(allow(occurs(a)),T),
   not holds(ab(occurs(a)),T),
   not holds(neg(occurs(a)),T),
   action(a), time(T), T<t_max.
holds(neg(occurs(a)),T) :-
   not holds(occurs(a),T),
   action(a), time(T), T<t_max.
```
(11)

Basic correctness and completeness result. The following result provides a basic correctness and completeness result; corresponding results for the specific reasoning modes are either obtained as corollaries or adaptions of its proof.

Theorem 1 *Let (D, O_{init}) be an action theory such that O_{init} contains only fluent observations about the initial state. Let Q be a query as in (9) and let*

$$A_Q = \{(a \textbf{ occurs\_at } t_i) \mid a \in A_i, 1 \leq i \leq m\} .$$

Let $\mathcal{T}$ denote the translation of $\mathcal{C}_{TAID}$ into logic programs, described above.

Then, we have the following results.

1. *If $s_0, A_1, s_1, A_2, \ldots, A_m, s_m$ is a trajectory model of $(D, O_{init} \cup A_Q)$,*
   then there is an answer set X of logic program $\mathcal{T}(D, O_{init} \cup A_Q)$ such that we have for all $f \in F$ and $0 \leq k \leq m$
   (a) `holds(f,k)` $\in X$, *if $s_k \models f$ and*
   (b) `holds(neg(f),k)` $\in X$, *if $s_k \models \neg f$.*
2. *If X is an answer set of logic program $\mathcal{T}(D, O_{init} \cup A_Q)$ and for $0 \leq k \leq m$*

$$s_k = \{f \mid \texttt{holds(f,k)} \in X\} \cup \{\neg f \mid \texttt{holds(neg(f),k)} \in X\}$$

   then there is a trajectory model $s_0, A_1, s_1, A_2, \ldots, A_m, s_m$ of $(D, O_{init} \cup A_Q)$.

Action query language. In the following t_{max} is the upper time bound, which has to be provided when the answer sets are computed.

Planning. Recall that the initial state can be partially specified; it is then completed by the rules in (10) for taking into account all possible initial states. A plan for $f_1, \ldots, f_n$ (cf. Definition 10) is translated using the predicate "achieved". It ensures that the goal holds in the final state of every answer set for the query.

```
:- not achieved.
achieved :- achieved(0).
achieved :- achieved(T+1),not achieved(T),
   time(T),time(T+1).
achieved(T) :- holds(f_1,T),...,holds(f_n,T),
   achieved(T+1),fluent(f_1),...,fluent(f_n),
   time(T),time(T+1).
achieved(n) :- holds(f_1,T),...,holds(f_n,T),
   fluent(f_1),...,fluent(f_n),T = t_max.
```

Constant t_{max} is the maximum number of steps in which the goals $f_1, \ldots, f_n$ should be achieved. The proposition `achieved(T)` represents the earliest point of time `T` at which the plan is successfully achieved. Once the query is satisfied only triggered actions can occur, all other actions should not occur since that might invalidate the plan. That is why `achieved(T)` occurs in the translation of every allowed and exogenous action.

```
holds(occurs(a),T) :-
   holds(allow(occurs(a)),T),
   not achieved(T),
   not holds(ab(occurs(a)),T),
   not holds(neg(occurs(a)),T),
   action(a),time(T).
holds(neg(occurs(a)),T) :-
   not holds(occurs(a),T),
   action(a),time(T).
```

These rules are used to generate all possible combinations of occurrences of non-triggered actions. Such actions can only occur as long as the goal is not yet achieved and if they are not inhibited. If there exists an answer set X for the planning problem, then for a plan P as defined in Definition 10 we have $(a \textbf{ occurs\_at } t_i) \in P$ if `holds(occurs(a),i)` $\in X$.

Explanation. The translation of an explanation contains the translation of all action and fluent observations in O, as described above. Since the observations about the initial state are often incomplete the translation contains the rules in (10) to generate all initial states which do not contradict the observations. Also, we have to generate possible combinations of occurrences of actions for all states. To this

effect, the translation includes for every exogenous and allowed action, the rules in (11). If there exists an answer set X for the explanation problem, then for an explanation E as defined in Definition 11 we have $(a \textbf{ occurs\_at } t_i) \in E$ if `holds(occurs(a),i)` $\in X$.

Prediction. The translation includes all fluent and action observations in O, as described above. As in explanation, we have to fill in missing information, which is necessary to justify the observed behaviour. That means we have to include for every fluent f two rules of form (10) to generate possible initial states. Moreover the translation includes for every non-triggered action two rules similar to those of an explanation of form (11). The actual prediction for $f_1, \ldots, f_n$ (cf. Definition 12) is translated as:

```
predicted :- holds(f1,T), ..., holds(fn,T),
   fluent(f1),...,fluent(fn),time(T),T >= i.
```

where i is the time of the latest observation. If the atom `predicted` is included in all answer sets, it is a cautious prediction. If it is only included in some answer sets, it is a brave prediction.

Application

Meanwhile, we have used $\mathcal{C}_{TAID}$ in several different application scenarios at the Max-Planck Institute for Molecular Plant Physiology for modelling metabolic as well as signal transduction networks. For illustration, we describe below the smallest such application, namely the sulfur starvation response-pathway of the model plant Arabidopsis thaliana. Sulfur is essential for the plant. If the amount of sulfur it can access is not sufficient to allow a normal development of the plant, the plant follows a complex strategy. First the plant forms additional lateral roots to access additional sources of sulfur and to normalise its sulfur level. However, if this strategy is not successful the plant uses its remaining resources to form seeds. A simplified version of the basic causal relationships constituting this network is shown in Figure 1.

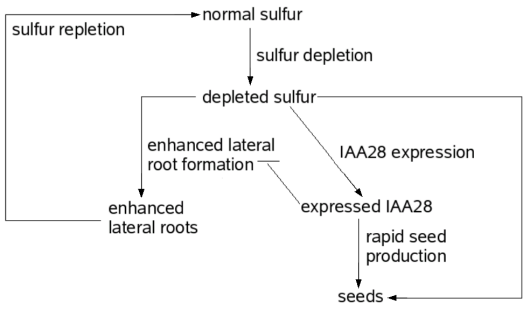


Figure 1: Sulfur starvation response-pathway of Arabidopsis thaliana

Normally, the amount of sulfur in a plant is sufficient, but due to external, e.g. environmental conditions, the amount of sulfur can be reduced. A problem, when modelling this network are these environmental conditions, which are not and cannot be part of such a model and which might or might not lead to the reduction of sulfur. Once the level of sulfur in the plant is decreased complex interactions of different compounds are triggered. Genes are activated, which induce the generation of auxin, a plant hormone, which plays a key role as a signal in coordinating the development of the plant. This eventually leads to the formation of additional lateral roots. Since this consumes the scarce resources, this development should be stopped, when it becomes apparent, that it is not successful, i.e. it takes too long and consumes too many of the plant's resources. This "emergency stop" is triggered by complex interactions that lead, via a surplus of the auxin flux, to the expression of IAA28, a gene which is subject to current research. If IAA28 is expressed and the sulfur level is still low, other processes result in a different physiological endpoint, the production of seeds.

We now show how this biological network can be represented as a domain description $D(A, F)$ in $\mathcal{C}_{TAID}$.

$$
\begin{aligned}
A &= \{\textit{sulfur\_depletion}, \textit{sulfur\_repletion}, \\
&\quad \textit{enhanced\_lateral\_root\_formation}, \\
&\quad \textit{iaa28\_expression}, \textit{rapid\_seed\_production}\} \\
F &= \{\textit{normal\_sulfur}, \textit{depleted\_sulfur}, \\
&\quad \textit{enhanced\_lateral\_roots}, \textit{expressed\_iaa28}, \textit{seeds}\}
\end{aligned}
$$

The biologist's knowledge about the biological system, gives rise to the following dynamic causal laws.

(sulfur_depletion **causes** *depleted_sulfur* **if** *normal_sulfur)*
(enhanced_lateral_root_formation **causes** *enhanced_lateral_roots)*
(sulfur_repletion **causes** *normal_sulfur)*
(iaa28_expression **causes** *expressed_iaa28)*
(rapid_seed_production **causes** *seeds)*

Additionally, two static causal laws specify the relationship between *normal sulfur* and *depleted sulfur*. They assure that at most one of the fluents is $true$ at all times.

(¬normal_sulfur **if** *depleted_sulfur)*
(¬depleted_sulfur **if** *normal_sulfur)*

For two of the actions, we know all the preconditions that have to be satisfied for the actions to occur.

(depleted_sulfur **triggers** *enhanced_lateral_root_formation)*
(expressed_iaa28 , depleted_sulfur **triggers** *rapid_seed_production)*

For the remaining three actions, it is more difficult to decide whether and when they occur. Whether the action *sulfur depletion* occurs depends on environmental conditions which are not part of the model. The same holds for the action *sulfur repletion*, which might or might not be successful, depending on the environmental conditions. For the occurrence of action *iaa28 expression* the question is not whether it occurs but when it occurs. The longer it is delayed, the more resources are used to form additional lateral roots.

(normal_sulfur **allows** *sulfur_depletion)*
(depleted_sulfur **allows** *iaa28_expression)*
(enhanced_lateral_roots **allows** *sulfur_repletion)*

There is only one inhibition relation in this example.

(expressed_iaa28 **inhibits** *enhanced_root_formation)*

But only if we add a default value for the fluent *enhanced lateral roots*, the inhibition relation has the desired effect of stopping the formation of additional lateral roots.

(**default** $\neg$*enhanced_lateral_roots)*

The knowledge that the plant either forms additional lateral roots or produces seeds can be expressed by the following no-concurrency constraint:

*(***noconcurrency** *enhanced_lateral_roots_formation ,*
rapid_seed_production)

After defining the domain description, let us define a set of observations O. The initial state where we still have a normal level of sulfur can be described by the following fluent observations:

$O =$ {*(normal_sulfur* **at** *0),*
($\neg$ *enhanced_lateral_roots* **at** *0),*
($\neg$ *expressed_iaa28* **at** *0), (*$\neg$ *seeds* **at** *0)*}

Now that we defined our action theory (D, O), we can start to reason about it. Let us first find an explanation for the observed behaviour:

$O_1 = O \cup$ {*(sulfur_depletion* **occurs_at** *0),*
(normal_sulfur **at** *3)*}

For a time bound of $t_{max} = 3$ there are already 4 possible explanations. They all have in common that *sulfur depletion* occurs at time point 0, the formation of lateral roots is triggered at time point 1 and the action *sulfur repletion* occurs at time point 2. The explanations differ in whether and when the action *iaa28 expression* and the action *rapid seed production* occurs. One explanation is:

(D,O_1*)* $\models_b$ *(true* **after** *sulfur_depletion* **occurs_at** *0,*
enhanced_lateral_root_formation **occurs_at** *1,*
enhanced_lateral_root_formation **occurs_at** *2,*
sulfur_repletion **occurs_at** *2)*

A second explanation is:

(D,O_1*)* $\models_b$ *(true* **after** *sulfur_depletion* **occurs_at** *0,*
enhanced_lateral_root_formation **occurs_at** *1,*
enhanced_lateral_root_formation **occurs_at** *2,*
sulfur_repletion **occurs_at** *2, iaa28_expression*
occurs_at *2)*

Our next question is whether the given observations are sufficient to predict a certain behaviour of the plant.

(D,O) $\models_c$ *(seeds* **after** *sulfur_depletion* **occurs_at** *0,*
iaa28_expression **occurs_at** *1)*
(D,O) $\models_b$ *(normal_sulfur* **after** *sulfur_depletion*
occurs_at *0, iaa28_expression* **occurs_at** *1)*

Using these predictions, we can say that when sulfur is depleted and IAA28 is expressed the plant grows seeds, but it is still possible that it also stabilises its sulfur level.

Finally, we want to find a plan for the action theory (D, O) that results in the production of seeds. For time bound $t_{max} = 3$, there are 4 plans. One possible plan is:

(D,O) $\models_b$ *(seeds* **after** *sulfur_depletion* **occurs_at** *0,*
iaa28_expression **occurs_at** *1,*
enhanced_lateral_root_formation **occurs_at** *1,*
rapid_seed_production **occurs_at** *2,*
rapid_seed_production **occurs_at** *3)*

The number of plans and explanations depend on the number of allowance rules, since the different possibilities for the occurrence of such an allowed action is reflected by different answer sets.

Discussion

We proposed the action language $\mathcal{C}_{TAID}$ and showed how it can be used to represent and reason about biological networks. $\mathcal{C}_{TAID}$ is based on the action language $\mathcal{A}_T^0$ introduced in (Tran & Baral 2004). The latter language provides only minimal features to define dynamic causal laws, triggering and inhibition rules, which turn to be a fruitful basis but insufficient for modelling our biological applications. Moreover, our exploratory approach made us propose the concept of allowance that enables the experimenter to investigate alternative models "in silico". As a consequence, we extended $\mathcal{A}_T^0$ by static causal laws, allowance rules, default rules and no-concurrency constraint which furnish a more appropriate representation of our biological networks. Especially static causal laws and default rules can be used to include background knowledge and other dependencies like environmental conditions which influence the biological system, but are not part of the actual model. Allowance rules are mainly used to express incomplete knowledge about the reasons why an action occurs. This missing information is a common problem for biologist which is due to the immanent complexity of biological systems.

We fixed the semantics of $\mathcal{C}_{TAID}$ in the standard way by means of transition relations, trajectories and trajectory models. In contrast to $\mathcal{A}_T^0$, for example, default values can enable state changes without the occurrence of an action. Also, Baral et al. guarantee a unique trajectory model and a unique answer set, if the initial state is completely defined by a set of observations. This is not the case in $\mathcal{C}_{TAID}$ because of the non-determinism introduced by allowance rules that may yield multiple answer sets.

We implemented our action language by means of a compiler mapping $\mathcal{C}_{TAID}$ onto logic programs under answer set semantics. Our translation builds upon and extends the one given in (Tran & Baral 2004). The resulting tool is implemented in Java and freely available at (bio). Meanwhile, it has been used in ten different application scenarios at the Max-Planck Institute for Molecular Plant Physiology for modelling metabolic as well as signal transduction networks. For illustration, we described the smallest such application, namely part of the sulfur starvation response-pathway of the model plant Arabidopsis thaliana.

Beyond the traditional approaches mentioned in the introductory section, further logic-based approaches using rule-based languages have emerged recently: Closely related work has been conducted in abductive logic programming where abduction was used in (Papatheodorou, Kakas, & Sergot 2005) as the principal mode of inference for modelling gene relations from micro-array data. A very sophisticated and much more advanced automated reasoning tool for systems biology can be found in the area of constraint programming, namely the BIOCHAM (Chabrier-Rivier, Fages, & Soliman 2004) system. BIOCHAM relies on CTL (Clarke, Grumberg, & Peled 1999) and is thus particularly strong in modelling temporal aspects of systems biology. Unlike our abstract approach, the constraint-based approach offers fine-grained capacities for modelling biochemical processes, including kinetics and reactions.

References

Baral, C.; Chancellor, K.; Tran, N.; Tran, N.; Joy, A.; and Berens, M. 2004. A knowledge based approach for representing and reasoning about signaling networks. In *ISMB/ECCB (Supplement of Bioinformatics)*, 15–22.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

http://bioinformatics.mpimp-golm.mpg.de/bionetreasoning/.

Bonarius, H. P. J.; Schmid, G.; and Tramper, J. 1997. Flux analysis of underdetermined metabolic networks: The quest for the missing constraints. *Trends Biotechnol* 15:308314.

Chabrier-Rivier, N.; Fages, F.; and Soliman, S. 2004. The biochemical abstract machine biocham. In Danos, V., and Schächter, V., eds., *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, 172–191. Springer.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model checking*. Cambridge, MA, USA: MIT Press.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2000. Planning under incomplete knowledge. In *CL '00: Proceedings of the First International Conference on Computational Logic*, 807–821. London, UK: Springer-Verlag.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–321.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electron. Trans. Artif. Intell.* 2:193–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: preliminary report. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, 623–630. AAAI Press.

Grell, S. 2006. Investigation and analysis of new approaches for representing and reasoning about biological networks using action languages. Diploma thesis, University of Potsdam and Max Planck Institute of Molecular Plant Physiology.

Papatheodorou, I.; Kakas, A. C.; and Sergot, M. J. 2005. Inference of gene relations from microarray data by abduction. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, 389–393. Springer.

Pinney, J. W.; Westhead, D. R.; and McConkey, G. A. 2003. Petri net representations in systems biology. *Biochem Soc Trans* 31(Pt 6):1513–1515.

Reddy, V.; Mavrovouniotis, M.; and Liebman, M. 1993. Petri net representations in metabolic pathways. *Proc. First ISMB* 328–336.

Shmulevich, I.; Dougherty, E.; Kim, S.; and Zhang, W. 2002. Probabilistic boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics* 18(2):261–274.

Tran, N., and Baral, C. 2004. Reasoning about triggered actions in ansprolog and its application to molecular interactions in cells. In *KR*, 554–564.

Tran, N.; Baral, C.; and Shankland, C. 2005. Issues in reasoning about interaction networks in cells: Necessity of event ordering knowledge. In Veloso, M. M., and Kambhampati, S., eds., *AAAI*, 676–681. AAAI Press AAAI Press / The MIT Press.

3.5 A Non-Monotonic Reasoning System for RDF Metadata

A Non-Monotonic Reasoning System for RDF Metadata

Efstratios Kontopoulos[1], Nick Bassiliades[1], Grigoris Antoniou[2]

[1]Department of Informatics, Aristotle University of Thessaloniki
GR-54124 Thessaloniki, Greece
{ skontopo,nbassili}@csd.auth.gr

[2]Institute of Computer Science, FO.R.T.H.
P.O. Box 1385, GR-71110, Heraklion, Greece
antoniou@ics.forth.gr

Abstract

Non-monotonic reasoning constitutes an approach to reasoning with incomplete or changing information and is significantly more powerful than standard reasoning, which simply deals with universal statements. Defeasible reasoning, a member of the non-monotonic reasoning family, offers the extra capability of dealing with conflicting information and can represent facts, rules and priorities among rules. The main advantages of defeasible reasoning, however, are not only limited to its enhanced representational capabilities, but also feature low computational complexity compared to mainstream non-monotonic reasoning. This paper presents a system for non-monotonic reasoning on the Semantic Web called VDR-Device, which is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules. It is implemented on top of the CLIPS production rule system and features a RuleML compatible syntax. The operational semantics of defeasible logic are implemented through compilation into a generic deductive rule language. Since the RuleML syntax may appear complex for many users, we have also implemented a graphical authoring tool for defeasible logic rules that acts as a shell for the defeasible reasoning system. The tool constrains the allowed vocabulary through analysis of the input RDF documents, so that the user does not have to type-in class and property names.

Introduction

The development of the Semantic Web (Berners-Lee, Hendler and Lassila 2001) proceeds in a hierarchy of layers, with each layer being on top of other layers. At present, the highest layer that has reached sufficient maturity is the ontology layer, where OWL (Dean and Schreiber 2004), a description logic based language, is currently the dominant standard.

Above the ontology layer lie the logic and proof layers, towards which the next steps in the development of the Semantic Web will be directed. Rule systems can play a twofold role in the Semantic Web initiative: (a) they can serve as extensions of, or alternatives to, description logic based ontology languages, since rules are more expressive than description logic languages like OWL and (b) they can be used to develop declarative systems on top of (using) ontologies.

Non-monotonic reasoning (Antoniou 1997) constitutes an approach that allows reasoning with incomplete or changing information. More specifically, it provides mechanisms for taking back conclusions that, in the presence of new information, turn out to be wrong and for deriving new, alternative conclusions instead. Contrary to standard reasoning, which simply deals with universal statements, non-monotonic reasoning offers a significantly higher level of expressiveness.

Defeasible reasoning (Nute 1987), a member of the non-monotonic reasoning family, represents a simple rule-based approach to reasoning not only with incomplete or changing but also with conflicting information. When compared to mainstream non-monotonic reasoning, the main advantages of defeasible reasoning are enhanced representational capabilities coupled with low computational complexity.

Defeasible reasoning can represent facts, rules and priorities and conflicts among rules. Such conflicts arise, among others, from rules with exceptions, which are a natural representation for policies and business rules (Antoniou, Billington and Maher 1999) and priority information is often available to resolve conflicts among rules. Other application domains are described later on in this work.

In this paper we report on the implementation of VDR-DEVICE which is a visual, integrated environment for the development and application of defeasible logic rule bases on top of RDF ontologies. VDR-Device consists of: (i) a visual RuleML-compliant rule editor and (ii) a defeasible reasoning system for the Semantic Web that processes RDF data and RDF Schema ontologies.

VDR-Device supports multiple rule types of defeasible logic (strict rules, defeasible rules and defeaters) as well as priorities among rules. It also supports two types of negation (strong negation and negation-as-failure) and conflicting (mutually exclusive) literals.

The system has a RuleML-compatible (Boley et al. 2001) syntax, which expresses the main standardization effort for rules in the Semantic Web. Input and output of data is performed through processing of RDF data and RDF Schema ontologies.

The reasoning system of VDR-Device is built on-top of a CLIPS-based implementation of deductive rules, called R-Device (Bassiliades and Vlahavas 2006). The core mechanism of the system performs the translation of defeasible knowledge into a set of deductive rules, including derived and aggregate attributes.

The rest of the paper is organized as follows: Firstly, the motivation for utilizing defeasible reasoning in the Semantic Web is more thoroughly examined. Then, a brief introduction to defeasible logics is made, followed by the section that presents the VDR-Device system. The presentation includes the architecture and functionality of the system, the syntax of the defeasible logic rule language, the underlying core deductive rule language, the translation from the defeasible logic rules to the deductive rules and the graphical rule editor. Related work on defeasible reasoning systems is discussed, next. Finally, this paper sums up the conclusions and gives potential directions for future work.

Conflicting Rules in the Semantic Web

This section briefly describes the main cases, where conflicting rules might be applied in the Semantic Web.

Reasoning with Incomplete Information

In (Antoniou 2002) a scenario is described where business rules have to deal with incomplete information: in the absence of certain information some assumptions have to be made that lead to conclusions not supported by classical predicate logic. In many applications on the Web such assumptions must be made because other players may not be able (e.g. due to communication problems) or willing (e.g. because of privacy or security concerns) to provide information. This is the classical case for the use of non-monotonic knowledge representation and reasoning (Marek and Truszczynski 1993).

Rules with Exceptions

As mentioned earlier, rules with exceptions are a natural way of representation for policies and business rules. And priority information is often implicitly or explicitly available to resolve conflicts among rules. Potential applications include security policies (Ashri et al. 2004), business rules (Antoniou and Arief 2002), e-contracting (Governatori 2005), brokering (Antoniou et al. 2005) and agent negotiations (Governatori et al. 2001).

Default Inheritance in Ontologies

Default inheritance is a well-known feature of certain knowledge representation formalisms. Thus it may play a role in ontology languages, which currently do not support this feature. In (Grosof and Poon 2003) some ideas are presented for possible uses of default inheritance in ontologies. A natural way of representing default inheritance is rules with exceptions plus priority information. Thus, non-monotonic rule systems can be utilized in ontology languages.

Ontology Merging

When ontologies from different authors and/or sources are merged, contradictions arise naturally. Predicate logic based formalisms, including all current Semantic Web languages, cannot cope with inconsistencies. If rule-based ontology languages are used (e.g. DLP (Grosof et al. 2003)) and if rules are interpreted as defeasible (that is, they may be prevented from being applied even if they can fire) then we arrive at non-monotonic rule systems. A skeptical approach, as adopted by defeasible reasoning, is sensible because it does not allow for contradictory conclusions to be drawn. Moreover, priorities may be used to resolve some conflicts among rules, based on knowledge about the reliability of sources or on user input). Thus, non-monotonic rule systems can support ontology integration.

Defeasible Logics

A *defeasible theory D* is a couple (*R*,>) where *R* a finite set of rules, and > a superiority relation on R. In expressing the proof theory we consider only propositional rules. Rules containing free variables are interpreted as the set of their variable-free instances.

There are three kinds of rules: *Strict rules* are denoted by $A \rightarrow p$, and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is “Professors are faculty members”. Written formally:

`professor(X)` $\rightarrow$ `faculty(X).`

Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by $A \Rightarrow p$ and can be defeated by contrary evidence. An example of such a rule is:

`professor(X)` $\Rightarrow$ `tenured(X)`

which reads as follows: “Professors are typically tenured”.

Defeaters are denoted as $A \sim> p$ and cannot actively support conclusions, but are used only to prevent some of them. A defeater example is:

`assistantProf(X) ~> ¬tenured(X)`

which is interpreted as follows: “Assistant professors may be not tenured”.

A *superiority relation* on R is an acyclic relation > on R (that is, the transitive closure of > is irreflexive). When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1.

This expresses that r_1 may override r_2. For example, given the defeasible rules

r_1: professor(X) ⇒ tenured(X)

r_2: visiting(X) ⇒ ¬tenured(X)

which contradict one another, no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation > with $r_2 > r_1$, then we can indeed conclude that a visiting professor is not tenured.

Another important element of defeasible reasoning is the notion of *conflicting literals*. In applications, literals are often considered to be conflicting and at most one of a certain set should be derived. An example of such an application is price negotiation, where an offer should be made by the potential buyer. The offer can be determined by several rules, whose conditions may or may not be mutually exclusive. All rules have `offer(X)` in their head, since an offer is usually a positive literal. However, only one offer should be made; therefore, only one of the rules should prevail, based on superiority relations among them. In this case, the conflict set is:

C(offer(x,y)) = { ¬offer(x,y) } ∪
{ offer(x,z) | z ≠ y }

For example, the following two rules make an offer for a given apartment, based on the buyer's requirements. However, the second one is more specific and its conclusion overrides the conclusion of the first one.

r_5: size(X,Y),Y≥45,garden(X,Z) ⇒
offer(X,250+2Z+5(Y−45))

r_6: size(X,Y),Y≥45,garden(X,Z),central(X) ⇒
offer(X,300+2Z+5(Y−45))

$r_6 > r_5$

The VDR-Device System

The VDR-Device system consists of two primary components:

1. DR-Device, the reasoning system that performs the RDF processing and inference and produces the results, and
2. DRREd (Defeasible Reasoning Rule Editor), the rule editor, which serves both as a rule authoring tool and as a graphical shell for the core reasoning system.

Although these two subsystems utilize different technologies and were developed independently, they intercommunicate efficiently, forming a flexible and powerful integrated environment.

The Non-Monotonic Reasoning System

The core reasoning system of VDR-Device is DR-Device (Bassiliades, Antoniou and Vlahavas 2006) and consists of two primary components (Fig. 1): The *RDF loader/translator* and the *rule loader/translator*. The user can either develop a rule base (program, written in the RuleML-like syntax of VDR-Device) with the help of the rule editor described in a following section, or he/she can load an already existing one, probably developed manually. The rule base contains: (a) a set of rules, (b) the URL(s) of the RDF input document(s), which is forwarded to the RDF loader, (c) the names of the derived classes to be exported as results and (d) the name of the RDF output document.

The rule base is then submitted to the *rule loader* which transforms it into the native CLIPS-like syntax through an XSLT stylesheet and the resulting program is then forwarded to the *rule translator*, where the defeasible logic rules are compiled into a set of CLIPS production rules (`http://www.ghg.net/clips/CLIPS.html`). This is a two-step process: First, the defeasible logic rules are translated into sets of deductive, derived attribute and aggregate attribute rules of the basic deductive rule language, using the translation scheme described in (Bassiliades, Antoniou and Vlahavas 2006). Then, all these deductive rules are translated into CLIPS production rules according to the rule translation scheme in (Bassiliades and Vlahavas 2006). All compiled rule formats are also kept in local files (structured in project workspaces), so that the next time they are needed they can be directly loaded, improving speed considerably (running a compiled project is up to 10 times faster).

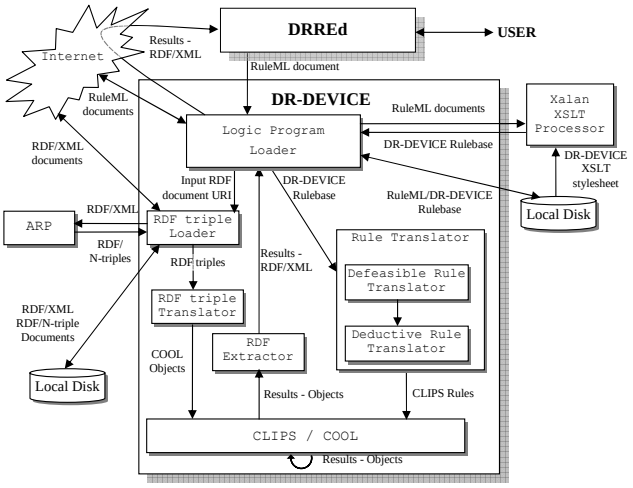


Fig. 1. Architecture of the VDR-DEVICE system.

Meanwhile, the *RDF loader* downloads the input RDF documents, including their schemas, and translates RDF descriptions into CLIPS objects, according to the RDF-to-object translation scheme in (Bassiliades and Vlahavas 2006), which is briefly described below.

The inference engine of CLIPS performs the reasoning by running the production rules and generates the objects that constitute the result of the initial rule program. The compilation phase guarantees correctness of the reasoning process according to the operational semantics of defeasible logic. Finally, the result-objects are exported to the user as an RDF/XML document through the RDF extractor. The RDF document includes the instances of the exported derived classes, which have been proved.

Syntax of the Defeasible Logic Rule Language

There are three types of rules in DR-DEVICE, closely reflecting defeasible logic: strict rules, defeasible rules, and defeaters. Each rule type is declared with a corresponding keyword (`strictrule`, `defeasiblerule` and `defeater` respectively). For example, the following rule construct represents the defeasible rule r_1: `professor(X)` ⇒ `tenured(X)`.

```
(defeasiblerule r1
   (professor (name ?X))
⇒
   (tenured (name ?X)))
```

Predicates have named arguments, called slots, since they represent CLIPS objects. DR-DEVICE has also a RuleML-like syntax. The same rule is represented in RuleML notation (version 0.85) as follows:

```
<imp>
   <_rlab ruleID="r1" ruletype="defeasiblerule">
     <ind>r1</ind></_rlab>
   <_head>
     <atom><_opr><rel>professor</rel></_opr>
          <_slot name="name"/><var>X</var></_slot>
     </atom></_head>
   <_body>
     <atom><_opr><rel href="tenured"/></_opr>
          <_slot name="name"><var>X</var></_slot>
     </atom>
   </_body>
</imp>
```

We have tried to re-use as many features of RuleML syntax as possible. However, several features of the DR-DEVICE rule language could not be captured by the existing RuleML DTDs (version 0.9); therefore, we have developed a new DTD (Fig. 2), using the modularization scheme of RuleML, extending the Datalog with strong negation. For example, rules have a unique (`ID`) `ruleID` attribute in their `_rlab` element, so that superiority of one rule over the other can be expressed through an `IDREF` attribute of the superior rule. For example, the following rule `r2` is superior to rule `r`, presented above.

```
(defeasiblerule r2
   (declare (superior r1)) (visiting (name ?X))
⇒
   (not (tenured (name ?X))))
```

In RuleML notation, there is a `superiority` attribute in the rule label.

```
<imp>
   <_rlab ruleID="r2" ruletype="defeasiblerule"
superior="r1">
     <ind>r2</ind>
   </_rlab>
...
</imp>
```

Classes and objects (facts) can also be declared in DR-DEVICE; however, the focus in this paper is the use of RDF data as facts. The input RDF file(s) are declared in the rdf_import attribute of the `rulebase` (root) element of the RuleML document. There exist two more attributes in the `rulebase` element: the `rdf_export` attribute that declares the address of the RDF file with the results of the rule program to be exported, and the `rdf_export_classes` attribute that declares the derived classes whose instances will be exported in RDF/XML format.

Further extensions to the RuleML syntax, include function calls that are used either as constraints in the rule body or as new value calculators at the rule head. Multiple constraints in the rule body can be expressed through the logical operators: `_not`, `_and`, `_or`.

```
<!ENTITY % LABELs "IDREFS">
<!ENTITY % CLASSes "NMTOKENS">
<!ATTLIST _rlab
   ruleID ID #REQUIRED
   ruletype (strictrule|defeasiblerule|defeater)
             #REQUIRED
   superior %LABELs; #IMPLIED>
<!ENTITY % _calc.cont "(function+)">
<!ELEMENT calc %_calc.cont;>
<!ENTITY % _head.content " (calc?, (atom | neg))">
<!ENTITY % _body.content "(atom | neg | and | or)">
<!ENTITY % _fname.cont "(#PCDATA)">
<!ELEMENT fname %_fname.cont;>
<!ENTITY % pos_term "(ind | var | function)">
<!ELEMENT function (fname, (%pos_term;)*)>
<!ENTITY % term "(_not | %pos_term;)">
<!ELEMENT _not (ind | var)>
<!ELEMENT _or (%term;, (%term;)+)>
<!ELEMENT _and (%term;, (%term;)+)>
<!ENTITY % constraint "(_not | _or | _and)">
<!ENTITY % _slot.content "(ind | var | %constraint;)">
<!ENTITY % negurdatalog_include SYSTEM
   "http://www.ruleml.org/0.85/dtd/neg/negurdatalog.dtd">
%negurdatalog_include;
<!ATTLIST rulebase
      rdf_import CDATA #IMPLIED
      rdf_export_classes %CLASSes; #IMPLIED
      rdf_export CDATA #IMPLIED>
```

Fig. 2. RuleML syntax DTD of the VDR-DEVICE rule language.

The Deductive Rule Language of R-DEVICE

R-DEVICE has a powerful deductive rule language which includes features such as normal (ground), unground, and generalized path expressions over the objects, stratified negation, aggregate, grouping, and sorting, functions. The rule language supports a second-order syntax, where variables can range over classes and properties. However, second-order variables are compiled away into sets of first-order rules, using instantiations of the metaclasses. Users can define views which are materialized and, optionally, incrementally maintained by translating deductive rules into CLIPS production rules. Users can choose between an OPS5/CLIPS-like or a RuleML-like syntax. Finally, users can use and define functions using the CLIPS host language. R-DEVICE belongs to a family of previous such deductive object-oriented rule languages (Bassiliades et al. 2000). Examples of rules are given below.

R-DEVICE, like DR-DEVICE, has both a native CLIPS-like syntax and a RuleML-compatible syntax. Here we will present a few examples using the former, since it is more concise. For example, assume there is an RDF class `carlo:owner` that defines the owners of the apartments and a property `carlo:has-owner` that relates an apartment to its owner.

The following rule returns the names of all apartments owned by "Smith":

```
(deductiverule test1
   (carlo:apartment (carlo:name ?x)
      ((carlo:lastName carlo:has-owner) "Smith"))
 =>
   (result (apartment ?x)))
```

The above rule has a ground path expression `(carlo:lastName carlo:has-owner)` where the right-most slot name `(carlo:has-owner)` is a slot of the "departing" class `carlo:apartment`. Moving to the left, slots be-long to classes that represent the range of the predecessor slots. In this example, the range of `carlo:has-owner` is `carlo:owner`, so the next slot `carlo:lastName` has domain `carlo:owner`. The value expression in the above pattern (e.g. constant "Smith") actually describes a value of the left-most slot of the path (`carlo:lastName`). Notice that we have adopted a right-to-left order of attributes, contrary to the left-to-right C-like dot notation that is commonly assumed, because we consider path expressions as function compositions, if we assume that each property is a function that maps its domain to its range.

Another example that demonstrates aggregate function in R-DEVICE is the following rule, which returns the number of apartments owned by each owner:

```
(deductiverule test2
   (carlo:apartment (carlo:name ?x)
          ((carlo:lastName carlo:has-owner) ?o))
 =>
   (result (owner ?o) (apartments (count ?x))))
```

Function `count` is an aggregate function that returns the number of all the different instantiations of the variable `?x` for each different instantiation of the variable `?o`. There are several other aggregate functions, such as `sum`, `avg`, `list`, etc.

Translating Defeasible Logic Rules into Deductive Rules

The translation of defeasible rules into R-DEVICE rules is based on the translation of defeasible theories into logic programs through the well-studied meta-program of (Antoniou et al. 2000). However, instead of directly using the meta-program at run-time, we have used it to guide defeasible rule compilation. Therefore, at run-time only first-order rules exist.

Before going into the details of the translation we briefly present the auxiliary system attributes (in addition to the user-defined attributes) that each defeasibly derived object in DR-DEVICE has, in order to support our translation scheme:

- `pos`, `neg`: These numerical slots hold the proof status of the defeasible object. A value of 1 at the `pos` slot denotes that the object has been defeasibly proven; whereas a value of 2 denotes definite proof. Equivalent `neg` slot values denote an equivalent proof status for the negation of the defeasible object. A 0 value for both slots denotes that there has been no proof for either the positive or the negative conclusion.
- `pos-sup`, `neg-sup`: These attributes hold the rule ids of the rules that can *potentially* prove the object positively or negatively.
- `pos-over`, `neg-over`: These attributes hold the rule ids of the rules that have overruled the positive or the negative proof of the defeasible object. For example, in the rules r_1 and r_2 presented above, rule r_2 has a negative conclusion that overrides the positive conclusion of rule r_1. Therefore, if the condition of rule r_2 is satisfied then its rule id is stored at the `pos-over` slot of the corresponding derived object.
- `pos-def`, `neg-def`: These attributes hold the rule ids of the rules that can defeat overriding rules when the former are superior to the latter. For example, rule r_2 is superior to rule r_1. Therefore, if the condition of rule r_2 is satisfied then its rule id is stored at the `neg-def` slot of the corresponding derived object along with the rule id of the defeated rule r_1. Then, even if the condition of rule r_1 is satisfied, it cannot overrule the negative conclusion derived by rule r_2 (as it is suggested by the previous paragraph) because it has been defeated by a superior rule.

Each *defeasible rule* in DR-DEVICE is translated into a set of 5 R-DEVICE rules:

- A *deductive* rule that generates the derived defeasible object when the condition of the defeasible rule is met. The proof status slots of the derived objects are initially set to 0. For example, for rule r_2 the following deductive rule is generated:

```
(deductiverule r2-deductive
   (visiting (name ?X))
⇒
   (tenured (name ?X) (pos 0) (neg 0)))
```

  Rule `r2-deductive` states that if an object of class `visiting` with slot `name` equal to `?X` exists, then create a new object of class `tenured` with a slot `name` with value `?X`. The derivation status of the new object (according to defeasible logic) is unknown since both its positive and negative truth status slots are set to 0. Notice that if a `tenured` object already exists with the same name, it is not created again. This is ensured by the value-based semantics of the R-DEVICE deductive rules.
- An aggregate attribute "*support*" rule that stores in `-sup` slots the rule ids of the rules that can potentially prove positively or negatively the object. For example, for rule r_2 the following "support" rule is generated (`list` is an aggregate function that just collects values in a list):

```
(aggregateattrule r2-sup
   (visiting (name ?X))
   ?gen23 <- (tenured (name ?X))
 ⇒
   ?gen23 <- (tenured (neg-sup (list r5))))
```

Rule `r2-sup` states that if there is a `visiting` object named `?X`, and there is a `tenured` object with the same name, then derive that rule `r2` could potentially support the defeasible negation of the `tenured` object (slot `neg-sup`).

- A derived attribute "*defeasibly*" rule that defeasibly proves either positively or negatively an object by storing the value of 1 in the `pos` or `neg` slots, if the rule condition has been at least defeasibly proven, if the opposite conclusion has not been definitely proven and if the rule has not been overruled by another rule. For example, for rule `r2` the following "defeasibly" rule is generated:

```
(derivedattrule r2-defeasibly
(visiting (name ?X) (pos ?gen29&:(>= ?gen29 1)))
  ?gen23 <- (tenured (name ?X) (pos ~2)
  (neg-over $?gen25&:(not (member$ r5 $?gen25))))
 ⇒
  ?gen23 <- (tenured (neg 1)))
```

Rule `r2-defeasibly` states that if it has been defeasibly proven that a `visiting` object named `?X` exists, and there is a `tenured` object with the same name that is not already strictly-positively proven and rule r_2 has not been overruled (check slot `neg-over`), then derive that the `tenured` object is defeasibly-negatively proven.

- A derived attribute "*overruled*" rule that stores in `-over` slots the rule id of the rule that has overruled the positive or the negative proof of the defeasible object, along with the ids of the rules that support the opposite conclusion, if the rule condition has been at least defeasibly proven, and if the rule has not been defeated by a superior rule. For example, for rule r_1 the following "overruled" rule is generated (through `calc` expressions, arbitrary user-defined calculations are performed):

```
(derivedattrule r1-over
  (professor (name ?X)
    (pos ?gen22&:(>= ?gen22 1)))
  ?gen16 <- (tenured (name ?X) (neg-sup $?gen19)
              (neg-over $?gen20)
              (pos-def $?gen18&
                :(not (member$ r4 $?gen18))))
 ⇒
(calc (bind $?gen21
  (create$ r1-over $?gen19 $?gen20)))
  ?gen16 <- (tenured (neg-over $?gen21)))
```

Rule `r1-over` actually overrules all rules that can support the negative derivation of `tenured`, including rule r_2. Specifically, it states that if it has been defeasibly proven that a `professor` object named `?X` exists, and there is a `tenured` object with the same name that its negation can be potentially supported by rules in the slot `neg-sup`, then derive that rule `r1-over` overruled those "negative supporters" (slot `neg-over`), unless it has been defeated (check slot `pos-def`).

- A derived attribute "*defeated*" rule that stores in `-def` slots the rule id of the rule that has defeated overriding rules (along with the defeated rule ids) when the former is superior to the latter, if the rule condition has been at least defeasibly proven. A "defeated" rule is generated only for rules that have a superiority relation, i.e. they are superior to others. For example, for rule r_5 the following "defeated" rule is generated:

```
(derivedattrule r2-def
  (visiting (name ?X)
    (pos ?gen29&:(>= ?gen29 1)))
  ?gen23 <- (tenured (name ?X) (pos-def $?gen26))
⇒
(calc (bind $?gen25 (create$ r2-def r1 $?gen26)))
  ?gen23 <- (tenured (pos-def $?gen25)))
```

Rule `r2-def` actually defeats rule r_1, since r_2 is superior to r_1. Specifically, it states that if it has been defeasibly proven that a `visiting` object named `?X` exists, and there is a `tenured` object with the same name then derive that rule `r2-def` defeats rule r_1 (slot `pos-def`).

Strict rules are handled in the same way as defeasible rules, with an addition of a derived attribute rule (called *definitely* rule) that definitely proves either positively or negatively an object by storing the value of 2 in the `pos` or `neg` slots, if the condition of the strict rule has been definitely proven, and if the opposite conclusion has not been definitely proven. For example, for the strict rule r_3: `visiting(X)` → `professor(X)`, the following "definitely" rule is generated:

```
(derivedattrule r3-definitely
  (visiting (name ?X) (pos 2))
  ?gen9 <- (professor (name ?X) (pos ~2))
  ⇒
  ?gen9 <- (professor (pos 2)))
```

Defeaters are much weaker rules that can only overrule a conclusion. Therefore, for a defeater only the "overruled" rule is created, along with a deductive rule to allow the creation of derived objects, even if their proof status cannot be supported by defeaters.

Execution Order

The order of execution of all the above rule types is as follows: "deductive", "support", "definitely", "defeated", "overruled", "defeasibly". Moreover, rule priority for stratified defeasible rule programs is determined by stratification. Finally, for non-stratified rule programs rule execution order is not determined. However, in order to ensure the correct result according to the defeasible logic theory for each derived attribute rule of the rule types "definitely", "defeated", "overruled" and "defeasibly" there is an opposite "truth maintenance" derived attribute rule that undoes (retracts) the conclusion when the condition is no longer met. In this way, even if rules are not executed in the correct order, the correct result will be eventually deduced because conclusions of rules that should have not been executed can be later undone. For example, the following rule undoes the "defeasibly" rule of rule r_2 when either the condition of the defeasible rule is no longer defeasibly satisfied, or the opposite conclusion has been definitely proven, or if rule r_5 has been overruled.

```
(derivedattrule r2-defeasibly-dot
```

```
?gen23 <- (tenured (name ?X) (neg 1)
            (neg-sup $? r5 $?))
(not (and (visiting (name ?X) (pos ?gen29&
              :(>= ?gen29 1)))
?gen23 <- (tenured (pos ~2) (neg-over $?g&
            :(not (member$ r2 $?g))))))
⇒
?gen23 <- (tenured (neg 0)))
```

DR-DEVICE has been tested for correctness using a tool that generates scalable test defeasible logic theories that comes with Deimos, a query answering defeasible logic system (Maher et al. 2001).

The Rule Editor

Writing rules in RuleML can often be a highly cumbersome task. Thus, the need for authoring tools that assist end-users in writing and expressing rules is apparently imperative.

VDR-Device is equipped with DRREd, a Java-built visual rule editor that aims at enhancing user-friendliness and efficiency during the development of VDR-Device RuleML documents. Its implementation is oriented towards simplicity of use and familiarity of interface. Other key features of the software include: (a) functional flexibility - program utilities can be triggered via a variety of overhead menu actions, keyboard shortcuts or popup menus, (b) improved development speed - rule bases can be developed in just a few steps and (c) powerful safety mechanisms – the correct syntax is ensured and the user is protected from syntactic or RDF Schema related semantic errors.

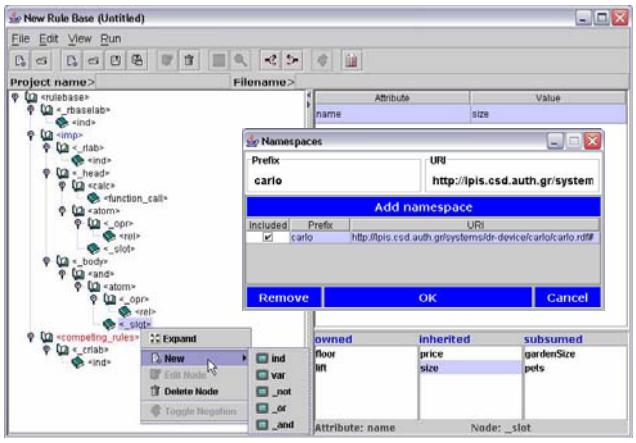


Fig. 3. The graphical rule editor and the namespace dialog window.

More specifically, and as can be observed in Fig. 3, the main window of the program is composed of two major parts: a) the upper part includes the menu bar, which contains the program menus, and the toolbar that includes icons, representing the most common utilities of the rule editor, and b) the central and more "bulky" part is the primary frame of the main window and is in turn divided in two panels.

The left panel displays the rule base in XML-tree format, which is the most intuitive means of displaying RuleML-like syntax, because of its hierarchical nature. The user has the option of navigating through the entire tree and can add to or remove elements from the tree. However, since each rule base is backed by a DTD document, potential addition or removal of tree elements has to obey to the DTD limitations. Therefore, the rule editor allows a limited number of operations performed on each element, according to the element's meaning within the rule tree.

The right panel shows a table, which contains the attributes that correspond to the selected tree node in the left-hand area. The user can also perform editing functions on the attributes, by altering the value for each attribute in the panel that appears below the attributes table on the right-hand side. The values that the user can insert are obviously limited by the chosen attribute each time.

The development of a rule base using VDR-Device is a delicate process that depends heavily on the parameters around the node that is being edited each time. First of all, there is an underlying procedure behind tree expansion, which is "launched" each time the user is trying to add a new element to the rule base. Namely, when a new element is added to the tree, all the mandatory sub-elements that accompany it are also added. In the cases where there are multiple alternative sub-elements, none of them is added to the rule base and the final choice is left to the user to determine which one of them has to be added. The user has to right-click on the parent element and choose the desired sub-element from the pop-up menu that appears (Fig. 3).

Another important component is the namespace dialog window (Fig. 3), where the user can determine which RDF/XML namespaces will be used by the rule base. Actually, we treat namespaces as addresses of input RDF Schema ontologies that contain the vocabulary for the input RDF documents, over which the rules will be run. The namespaces entered by the user, as well as those contained in the input RDF documents (indicated by the `rdf_import` attribute of the `rulebase` root element), are analyzed in order to extract all the allowed class and property names for the rule base being developed (see next section). These names are then used throughout the authoring phase of the RuleML rule base, constraining the corresponding allowed names that can be applied and narrowing the possibility for errors on behalf of the user.

Moving on to more node-specific features of the rule editor, one of the rule base elements that are treated in a specific manner is the `atom` element, which can be either negated or not. The response of the editor to an atom negation is performed through the wrapping/unwrapping of the `atom` element within a `neg` element and it is performed via a toggle button, located on the overhead toolbar.

Some components that also need "special treatment" are the rule IDs, each of which uniquely represents a rule within the rule base. Thus, the rule editor has to collect all of the RuleIDs inserted, in order to prohibit the user from entering the same RuleID twice and also equipping other

IDREF attributes (e.g. `superior` attribute) with the list of RuleIDs, constraining the variety of possible values.

The names of the functions that appear inside a `fun_call` element are also partially constrained by the rule editor, since the user can either insert a custom-named function or a CLIPS built-in function. Through radio-buttons the user determines whether he/she is using a custom or a CLIPS function. In the latter case, a list of all built-in functions is displayed, once again constraining possible entries.

Finally, users can examine all the exported results via an Internet Explorer window, launched by VDR-Device. Also, to improve reliability, the user can also observe the execution trace of compilation and running, both during run-time and also after the whole process has been terminated.

Related Work

There exist several previous implementations of defeasible logics, although to the best of our knowledge none of them is supported by a user-friendly integrated development environment or a visual rule editor. *Deimos* (Maher et al. 2001) is a flexible, query processing system based on Haskell. It implements several variants, but neither conflicting literals nor negation as failure in the object language. Also, the current implementation does not integrate with Semantic Web, since it is solely a defeasible logic engine (for example, there is no way to treat RDF data and RDFS/OWL ontologies; nor does it use an XML-based or RDF-based syntax for syntactic interoperability). Therefore, it is only an isolated solution, although external translation modules could provide such interoperability. Finally, it is propositional and does not support variables.

Delores (Maher et al. 2001) is another implementation, which computes all conclusions from a defeasible theory. It is very efficient, exhibiting linear computational complexity. Delores only supports ambiguity blocking propositional defeasible logic; so, it does not support ambiguity propagation, nor conflicting literals, variables and negation as failure in the object language. Also, it does not integrate with other Semantic Web languages and systems, and is, thus, an isolated solution as well.

SweetJess (Grosof, Gandhe and Finin 2002) is yet another implementation of a defeasible reasoning system based on Jess. It integrates well with RuleML. However, SweetJess rules can only express reasoning over ontologies expressed in DAMLRuleML (a DAML-OIL like syntax of RuleML) and not on arbitrary RDF data, like VDR-DEVICE. Furthermore, SweetJess is restricted to simple terms (variables and atoms). This applies to VDR-DEVICE to a large extent; however, the basic R-DEVICE language (Bassiliades and Vlahavas 2006) can support a limited form of functions in the following sense: (a) path expressions are allowed in the rule condition, which can be seen as complex functions, where allowed function names are object referencing slots; (b) aggregate and sorting functions are allowed in the conclusion of aggregate rules. Finally, VDR-DEVICE can also support conclusions in non-stratified rule programs due to the presence of truth-maintenance rules (Bassiliades, Antoniou and Vlahavas 2006).

Mandarax (Dietrich et al. 2003) is a Java rule platform, which provides a rule mark-up language (compatible with RuleML) for expressing rules and facts that may refer to Java objects. It is based on derivation rules with negation-as-failure, top-down rule evaluation, and generating answers by logical term unification. RDF documents can be loaded into Mandarax as triplets. Furthermore, Mandarax is supported by the Oryx graphical rule management tool. Oryx includes a repository for managing the vocabulary, a formal-natural-language-based rule editor and a graphical user interface library. Contrasted, the rule authoring tool of DR-DEVICE lies closer to the XML nature of its rule syntax and follows a more traditional object-oriented view of the RDF data model (Bassiliades and Vlahavas 2006). Furthermore, DR-DEVICE supports both negation-as-failure and strong negation, and supports both deductive and defeasible logic rules.

Conclusions and Future Work

In this paper we have argued that defeasible reasoning is useful for many applications in the Semantic Web, mainly due to conflicting rules and rule priorities. We have also presented a system for defeasible reasoning on the Web, called VDR-Device. It is a visual environment for developing defeasible logic rule bases that, after analyzing the input RDF ontologies, it constrains the allowed vocabulary. Furthermore, the system employs a user-friendly graphical shell and a powerful defeasible reasoning system that supports the following:

- Multiple rule types of defeasible logic, such as strict rules, defeasible rules, and defeaters.
- Priorities among rules.
- Two types of negation (strong, negation-as-failure) and conflicting (mutually exclusive) literals.
- Compatibility with RuleML, the main standardization effort for rules on the Semantic Web.
- Direct import from the Web and processing of RDF data and RDF Schema ontologies.
- Direct export to the Web of the results (conclusions) of the logic program as an RDF document.

The defeasible reasoning system is built on-top of a CLIPS-based implementation of deductive rules. The core of the system consists of a translation of defeasible knowledge into a set of deductive rules, including derived and aggregate attributes. However, the implementation is declarative because it interprets the not operator using Well-Founded Semantics.

In the future, we plan to delve into the proof layer of the Semantic Web architecture by enhancing further the graphical environment with rule execution tracing, expla-

nation, proof exchange in an XML or RDF format, proof visualization and validation, etc. We will try to visualize the semantics of defeasible logic in an intuitive manner, by providing graphical representations of rule attacks, superiorities, conflicting literals, etc. These facilities would be useful for increasing the trust of users for the Semantic Web agents and for automating proof exchange and trust among agents in the Semantic Web. Furthermore, we will include a graphical RDF ontology and data editor that will comply with the user-interface of the RuleML editor. Finally, concerning the implementation of the graphical editor we will adhere to newer XML Schema-based versions of RuleML.

References

Antoniou, G. 1997. *Nonmonotonic Reasoning*. MIT Press.

Antoniou G. 2002. Nonmonotonic Rule Systems on Top of Ontology Layers. In *Proceedings of the 1st Int. Semantic Web Conference*. 394-398. LNCS 2342. Springer-Verlag.

Antoniou, G., and Arief, M. 2002. Executable Declarative Business Rules and their Use in Electronic Commerce. In *Proceedings of ACM Symposium on Applied Computing*. 6-10. ACM Press.

Antoniou, G., Billington, D., Governatori, G., Maher M.J. 2000. A Flexible Framework for Defeasible Logics. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*. 405-410. AAAI/MIT Press.

Antoniou, G., Billington, D., and Maher, M.J. 1999. On the Analysis of Regulations using Defeasible Rules. In *Proceedings of the 32nd Hawaii Int. Conference on Systems Science*, 7 pages (no page numbers). IEEE Press.

Antoniou G., Billington D., Governatori G. and Maher M.J., "Representation results for defeasible logic", *ACM Trans. on Computational Logic*, 2(2), 2001, pp. 255-287

Antoniou, G., Skylogiannis, T., Bikakis, A., Bassiliades, N. 2005. DR-BROKERING – A Defeasible Logic-Based System for Semantic Brokering. In *Proceedings of IEEE Int. Conf. on E-Technology, E-Commerce and E-Service*. 414-417. IEEE Computer Society.

Ashri, R., Payne, T., Marvin, D., Surridge, M., and Taylor, S. 2004. Towards a Semantic Web Security Infrastructure. In *Proceedings of Semantic Web Services 2004 Spring Symposium Series*. Stanford University, Stanford California.

Bassiliades, N., Antoniou, G., Vlahavas I. 2006. A Defeasible Logic Reasoner for the Semantic Web. *International Journal on Semantic Web and Information Systems*, 2(1): 1-41.

Bassiliades, N., and Vlahavas, I. 2006. R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata, *International Journal on Semantic Web and Information Systems*, 2(2) (to appear).

Bassiliades, N., Vlahavas, and I., Elmagarmid, A.K. 2000. E DEVICE: An extensible active knowledge base system with multiple rule type support. *IEEE TKDE*. 12(5): 824-844.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American* 284(5):34-43.

Boley, H., Tabet, S., and Wagner, G. 2001. Design Rationale for RuleML: A Markup Language for Semantic Web Rules. *SWWS 2001*: 381-401.

Dean, M., and Schreiber, G. eds. 2004. *OWL Web Ontology Language Reference*. `www.w3.org/TR/2004/REC-owl-ref-20040210/`

Dietrich, J.; Kozlenkov A.; Schroeder, M.; Wagner, G. 2003. Rule-based agents for the semantic web. *Electronic Commerce Research and Applications*. 2(4):323–338.

Governatori, G. 2005. Representing business contracts in RuleML, *International Journal of Cooperative Information Systems*, 14(2-3):181-216.

Governatori, G., Dumas, M., Hofstede, A. ter and Oaks P. 2001. A formal approach to protocols and strategies for (legal) negotiation, In *Proceedings of the 8th International Conference of Artificial Intelligence and Law*. 168-177. ACM Press.

Grosof, B.N., Gandhe, M.D., Finin, T.W. 2002. SweetJess: Translating DAMLRuleML to JESS. In *Proceedings of Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*. Held at 1st Int. Semantic Web Conference.

Grosof, B. N. and Poon T. C. 2003. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the 12th Int. Conference on World Wide Web*. 340-349. ACM Press.

Hayes, P., "RDF Semantics", *W3C Recommendation*, Feb. 2004, `http://www.w3c.org/TR/rdf-mt/`

Maher, M.J.; Rock, A.; Antoniou, G.; Billington, D.; Miller T. 2001. Efficient Defeasible Reasoning Systems. *Int. Journal of Tools with Artificial Intelligence*. 10(4):483-501.

Marek, V.W., Truszczynski, M. 1993. *Nonmonotonic Logics; Context Dependent Reasoning*. Springer-Verlag.

McBride, B. 2001. Jena: Implementing the RDF Model and Syntax Specification. In *Proceedings of the 2nd Int. Workshop on the Semantic Web*, CEUR Workshop Proceedings, Vol. 40.

Nute, D. 1987. Defeasible Reasoning. In *Proceedings of the 20th Int. Conference on Systems Science*, 470-477. IEEE Press.

3.6 Relating Defeasible Logic to the Well-Founded Semantics for Normal Logic Programs

Relating Defeasible Logic to the Well-Founded Semantics for Normal Logic Programs

Frederick Maier and **Donald Nute**
Department of Computer Science and Artificial Intelligence Center
The University of Georgia
Athens, GA 30602
fmaier@uga.edu and dnute@uga.edu

Abstract

The most recent version of defeasible logic (Nute 1997) is related to the well-founded semantics by translating defeasible theories into normal logic programs. A correspondence is shown between the assertions of a defeasible theory and the literals contained in the well-founded model of the translation. The translation scheme is based upon (Antoniou & Maher 2002) but is modified to account for unique features of this defeasible logic.

Introduction

This paper relates the most recent version of defeasible logic first described in (Nute 1997) to the well-founded semantics for normal logic programs via a translation scheme. After the scheme is presented, a correspondence is proven to exist between, on the one hand, positive and negative assertions of a defeasible theory, and (selected) positive and negative literals of the logic program's well-founded model.

The translation scheme is based upon work of Antoniou and Maher in (2002) which is based on a notational variant of an earlier version of defeasible logic described in (Nute 1992; 1994). For the purposes of this paper, we will call the earlier version of defeasible logic DL and the newer version NDL. Antoniou and Maher show a relationship between DL and stable models, but the result is limited to defeasible theories without cycles in their rules. They establish a more general relationship between all defeasible theories and Kunen's 3 valued semantics described in (Kunen 1987). Antoniou and Maher cite extensive recent work on the theory and applications of DL, and further discussion of applications of NDL can be found in (Nute 2001). A deontic version of NDL is also presented in (Nute 1997).

NDL goes beyond DL by including a more extensive treatment of the ways that defeasible rules may conflict and by explicitly considering failures of proofs due to cycles in defeasible rules. Both of these are important improvements in defeasible logic, and the impact of these changes are discussed in detail in (Nute 2001). Studying NDL in relation to the well-founded semantics is a natural choice. Both are formalisms for nonmonotonic reasoning which explicitly consider dependency cycles between literals, and both are purportedly computationally tractable.

The majority of this paper is devoted to proving the correspondence between the assertions of a defeasible theory in NDL and the well-founded model of the translation. We interpret this result as showing that well-founded models indirectly provide a semantics for NDL. (For an alternative semantics for NDL, see (Donnelly 1999).) Furthermore, our result allows us to use our translation method together with a proof method that is sound with respect the the well-founded semantics as an implementation of NDL.

Defeasible Logic: Language and Proof System

This section presents necessary terminology for NDL and its proof system. For a fuller discussion, see (Nute 2001). The only well-formed formulas in the language for NDL are literals (atomic sentences and their negations). The language also contains *strict* rules, written $A \rightarrow p$ (where p is a literal and A a finite set of literals), *defeasible* rules (written $A \Rightarrow p$), and *undercutting defeaters* or simply *defeaters* (written $A \rightsquigarrow p$).

Definition 1: A defeasible theory D is a quadruple $\langle F, R, C, \prec \rangle$, where F is a possibly empty set of literals in the language of D, R a set of rules, C a set of finite sets of literals in the language of D such that for any literal p in the language of the theory, $\{p, \neg p\} \in C$, and $\prec$ an acyclic superiority relation over R.

The basic ideas behind the proof theory for NDL is that we can derive a literal p from a defeasible theory just in case p is one of the initial facts of the theory ($p \in F$,) or p is the head of some strict or *undefeated* defeasible rule in the theory and all of the literals in the body of the rule are also derivable. The role of defeaters is solely to defeat other arguments that might otherwise establish a literal.

Let $D = \langle F, R, C, \prec \rangle$ be a defeasible theory. If C only contains sets of the form $\{p, \neg p\}$, we say D has a minimal conflict set. We say that the conflict set C of D is closed under strict rules if, for all $cs \in C$, if $A \rightarrow p$ is a rule and $p \in cs$, then $\{A \cup (cs - \{p\})\} \in C$. It is not a necessary condition that a defeasible theory be closed under strict rules, but it is certainly an attractive condition. We will call a defeasible theory *closed* if its conflict set is closed under strict rules.

The proof theory for NDL is based upon argument trees:

Definition 2: Let D be a defeasible theory and p a literal in the language of D. The expression $D \mid\sim p$ is called a *positive defeasible assertion*, while $D \sim\!\!\mid p$ is called a *negative defeasible assertion*.

Informally, $D \mid\sim p$ and $D \sim\!\!\mid p$ are interpreted to mean that a demonstration (respectively, a refutation) exists for p from D. Note that $D \sim\!\!\mid p$ is equvalent to neither $D \mid\sim \neg p$ nor $D \not\mid\sim p$. $D \sim\!\!\mid p$ means that there is a demonstration that there is no defeasible proof of p from D.

Definition 3: τ is a *defeasible argument tree* for D iff τ is a finite tree such that every node of τ is labelled either $D \mid\sim p$ or $D \sim\!\!\mid p$ (for some literal p appearing in D).

Definition 4: The *depth* of a node n is k iff n has $k-1$ ancestors in τ. The depth of a tree is taken to be the greatest depth of any of its nodes.

Definition 5: Let A be a set of literals, and n a node of τ:

1. A *succeeds* at n iff for all $q \in A$, there is a child m of n such that m is labelled $D \mid\sim q$.
2. A *fails* at n iff there is a $q \in A$ and a child m of n such that m is labelled $D \sim\!\!\mid q$.

Definition 6: τ is a defeasible proof iff τ is an argument tree for D, and for each node n of τ, one of the following obtains:

1. n is labelled $D \mid\sim p$ and either:
   a. $p \in F$,
   b. there is a strict rule $r : A \rightarrow p \in R$ such that A succeeds at n, or
   c. there is a defeasible rule $r : A \Rightarrow p \in R$ such that A succeeds at n and for all $cs_i \in C$, if $p \in cs_i$, then there is a $q \in (cs_i - (F \cup \{p\}))$ such that
      i. for all strict rules $s : B \rightarrow q \in R$, B fails at n, and
      ii. for all defeasible rules $s : B \Rightarrow q \in R$, either B fails at n or else $s \prec r$, and
      iii. for all defeaters $s : B \rightsquigarrow q \in R$, either B fails at n or else $s \prec r$.
2. n is labelled $D \sim\!\!\mid p$ and:
   a. $p \notin F$,
   b. for all strict rules $r : A \rightarrow p \in R$, A fails at n, and
   c. for all defeasible rules $r : A \Rightarrow p \in R$, either
      i. A fails at n, or
      ii. there is a $cs_i \in C$ such that $p \in cs_i$, and for all $q \in cs_i - (F \cup \{p\})$, either
         A. there is a strict rule $s : B \rightarrow q \in R$ such that B succeeds at n,
         B. there is a defeasible rule $s : B \Rightarrow q \in R$ such that B succeeds at n and $s \not\prec r$, or
         C. there is a defeater rule $s : B \rightsquigarrow q \in R$ such that B succeeds at n and $s \not\prec r$.
3. n is labelled $D \sim\!\!\mid p$ and n has an ancestor m in τ such that m is labelled $D \sim\!\!\mid p$ and all nodes between n and m are negative defeasible assertions.

Condition 6.3 is called *failure-by-looping*. Since conclusions cannot be established by circular arguments, failure-by-looping can be used to help show that a literal cannot be derived from a defeasible theory.

Definition 7: Where D is a defeasible theory and S is a set of literals in the language of D, $D \mid\sim S$ if and only if for all $p \in S$, $D \mid\sim p$.

Some important formal properties of this logic are established in the following theorems.

Theorem 1 (Coherence): *If D is a defeasible theory and $D \mid\sim p$, then $D \not\sim\!\!\mid p$.*

Theorem 2 (Consistency): *If $D = \langle F, R, C, \prec\rangle$ is a defeasible theory, $S \in C$, and $D \mid\sim S$, then $\langle F, \{A \rightarrow p : A \rightarrow p \in R\}, C, \prec\rangle \mid\sim S$.*

Theorem 3 (Cautious Monotony): *If $D = \langle F, R, C, \prec\rangle$ is a defeasible theory, $D \mid\sim p$, and $D \mid\sim q$, then $\langle F \cup \{p\}, R, C, \prec\rangle \mid\sim q$.*

Theorem 1 assures us that we can not both prove and demonstrate the absence of any proof for the same literal. Theorem 2 says that any incompatible set of literals derivable from a defeasible theory must be derivable from the facts and the strict rules alone. In other words, the defeasible rules of a theory can never introduce any new incompatibilites. Of course, this interpretation of Theorem 2 assumes that all possible incompatibilities are captured in the conflict set of the theory. Cautious Monotony is a principle which many authors working on nonmonotonic reasoning propose as a necessary feature for any adequate nonmonotonic formalism.

The advantages of adding failure-by-looping to our proof theory should be obvious. Consider a simple defeasible theory like the following:

$D = \langle\ \{mammal\}\ ,\ \{\{furry, has\_wings\} \Rightarrow bat,\ bat \Rightarrow furry,\ bat \Rightarrow has\_wings,\ bat \Rightarrow flies,\ mammal \Rightarrow \neg flies\},\ \{\{bat, \neg bat\},\ \{furry, \neg furry\},\ \{has\_wings, \neg has\_wings\},\ \{mammal, \neg mammal\},\ \{flies, \neg flies\}\},\ \emptyset\rangle$.

In earlier versions of defeasible logic that lacked failure-by-looping, although we could easily see that there was no way to show $D \mid\sim bat$, we could not demonstrate this in the proof theory, that is, we could not show $D \sim\!\!\mid bat$. Consequently, neither could we show $D \mid\sim \neg flies$. Failure-by-looping provides a mechanism for showing $D \sim\!\!\mid bat$, which then allow us to show $D \mid\sim \neg flies$. When we later define a translation of defeasible theories into a standard logic programs in such a way that the consequences of a theory correspond to the well-founded semantics for the logic program, this example will also serve to show that failure-by-looping is necessary to get this correspondence. Where the theory D above is translated into the standard logic program P_D, `¬defeasibly(bat)`, `¬defeasibly(furry)`, `¬defeasibly(has_wings)`, and `defeasibly(neg_flies)` are all in the well-founded model of P_D, but the corresponding results are undetermined in versions of defeasible logic without failure-by-looping. Where a defeasible theory has cyclic rules, failure-by-looping is needed to capture within the proof theory the concept of a literal being unfounded.

Adding explicit conflict sets and closing them under strict rules provides an alternative solution to a class of examples that have always seemed odd to the authors. Consider the theory

$D = \langle\ \{quaker, republican\}\ ,\ \{quaker \Rightarrow dove,$

$republican \Rightarrow hawk$, $dove \Rightarrow activist$, $hawk \Rightarrow activist$, $dove \rightarrow \neg hawk$, $hawk \rightarrow \neg dove\}$, $\{\{quaker, \neg quaker\}, \{republican, \neg republican\}, \{dove, \neg dove\}, \{hawk, \neg hawk\}, \{dove, hawk\}\}, \emptyset\rangle$.

Theories like Reiter's default logic (Reiter 1980) would generate two default extensions for a theory like this: $\{quaker, republican, dove, activist\}$ and $\{quaker, republican, hawk, activist\}$. The skeptical approach would accept the intersection of these extensions, $\{quaker, republican, activist\}$, including $activist$ as a *floating conclusion* (Makinson & Schechta 1991). This seems unintuitive to us. A Republican quaker might be a dove or a hawk, but might just as well be neither. This is reflected in NDL. In this theory, the rules $quaker \Rightarrow dove$ and $republican \Rightarrow hawk$ conflict with each other since $\{dove, hawk\}$ is a conflict set in the theory. Neither rule takes precedence over the other; so neither consequent is defeasbily derivable. Thus neither of the rules to establish $activist$ is satisfied, and $activist$ is also not defeasibly derivable. Our proof theory avoids these floating conclusions in an intuitively reasonable way.

Logic Programs

Recall that a logic program consists of a set of rules having the form

$$P \texttt{:-} Q_1, Q_2 \ldots Q_n.$$

where each P and Q_i are atomic formulae of first order logic, or else such formulae preceded by a negation symbol. A rule in which the set of Q's is empty is called a *fact*. The neck ':-' is interpreted as 'if', though it is incorrect to view it as material implication (Gelder, Ross, & Schlipf 1991). Commas separating formulae in the body indicate logical conjunction.

Constants, variables, and function symbols are allowed to appear in each formula. If variables are used, then the rule is assumed to be universally quantified, though the quantifiers themselves are usually omitted in the written program.

Definite logic programs do not contain negation (in any form) in either the head or body of a rule. A *normal* (sometimes *general* (Gelder, Ross, & Schlipf 1991)) logic program may contain atoms in the body preceded by `not`, where the negation is taken to be negation-as-failure: An atom preceded by *not* is true only if it cannot be proven from the program. This can be contrasted with *strong* (sometimes *explicit*) negation (e.g., $\neg p$). Programs in which both strong negation and negation-as-failure appear are called *extended logic programs*. In a normal program, only `not` is allowable.

Definite programs have a unique minimal Herbrand model, and this is often taken to be the intended meaning of the program (Gelder, Ross, & Schlipf 1991). This result was shown in a paper by Van Emden and Kowalski in 1976 (Emden & Kowalski 1976). It was also shown there that this least Herbrand model corresponds to least fixed-point obtainable via the T operator (the *immediate consequence operator*), defined below.

Let P be a logic program and $\mathcal{I}$ a set of ground atoms. Then:

$$T_P(\mathcal{I}) = \{p \mid r \text{ is a rule of } P \text{ with head } p, \text{ and each } q_i \text{ in the body of } r \text{ is in } \mathcal{I}\}.$$

Importantly, when negation occurs in a program, a single least Herbrand model need not exist. For instance, the program `p :- not q` has two minimal models: $\{p\}$ and $\{q\}$ (Gelder, Ross, & Schlipf 1991). In such cases, the meaning of the program is unclear.

The Well-Founded Model for Normal Logic Programs

Several attempts have been made to provide a reasonable interpretation of logic programs containing negation. The *well-founded semantics* (Gelder, Ross, & Schlipf 1988; 1991) was developed for normal programs but has since been applied to extended logic programs.

Let P be a normal logic program containing only ground terms. An interpretation $\mathcal{I}$ of P is simply a consistent set of positive and negative literals whose atoms are taken from the Herbrand base of P. The consistency of $\mathcal{I}$ is required—if a literal p is in $\mathcal{I}$, then its complement $\neg p$ does not appear in $\mathcal{I}$.

If p appears in $\mathcal{I}$, then p is said to be $true$ in $\mathcal{I}$. If $\neg p$ appears in $\mathcal{I}$, then p is *false* in $\mathcal{I}$. If neither p nor $\neg p$ appears in $\mathcal{I}$, then p is said to be *undefined* in $\mathcal{I}$.

A set of literals S from the Herbrand base of P is said to be *unfounded* with respect to an interpretation $\mathcal{I}$ iff for each each $p \in S$ and for each rule r of P with head p, there exists a (positive or negative) subgoal q of r such that

1. q is false in $\mathcal{I}$ (that is, $\neg q$ appears in $\mathcal{I}$), or
2. q is positive and appears in S.

Unfounded sets are closed under union, and so for any P and $\mathcal{I}$, there exists a *greatest unfounded set* of P wrt $\mathcal{I}$, denoted $U_P(\mathcal{I})$:

$$U_P(\mathcal{I}) = \{\bigcup A \mid A \text{ is an unfounded set of } P \text{ with respect to } \mathcal{I}\}.$$

$U_P(\mathcal{I})$ can be viewed as a monotone operator and together with T_P is used to define another operator, W_P:

$$W_P(\mathcal{I}) = T_P(\mathcal{I}) \cup \neg \cdot U_P(\mathcal{I})$$

where $\neg \cdot U_P(\mathcal{I})$ is the element-wise negation of $U_P(\mathcal{I})$. $U_P(\mathcal{I})$, $T_P(\mathcal{I})$, and $W_P(\mathcal{I})$ are all monotonic, and can be used to define the sequence $(\mathcal{I}_0, \mathcal{I}_1, \ldots)$, as follows:

1. $\mathcal{I}_0 = \emptyset$
2. $\mathcal{I}_{k+1} = W_P(\mathcal{I}_k)$

The well-founded model of P, $wfm(P)$, is the limit of this sequence. If the Herbrand base is finite, then this limit can be reached in a finite number of iterations.

In the following, we will only be dealing with the propositional case—only finite grounded defeasible theories and programs are considered. Also, we allow $\mathcal{I}_0$ to contain facts of programs.

Translating a defeasible theory into a logic program

Let D be a finite propositional defeasible theory $\langle F, R, C, \prec \rangle$, where F is a set of facts, R is a set of strict, defeasible, and defeater rules, C is a conflict set, and $\prec$ is an acyclic superiority relation over R.
The translation of D into a normal logic program $LP(D)$ is shown below. We shall use $head(r)$ to denote the head of a rule and $body(r)$ to denote the set of its subgoals. $\neg p$ denotes the complement of a literal p. The expression not indicates negation as failure.

1. For each literal $p \in F$ add the following fact to $LP(D)$:

   `defeasibly(p).`

   Facts cannot be defeated, and so they can always safely be inferred.

2. For every pair of rules r and s, if $r \prec s$, add the following fact to $LP(D)$:

   `sup(s,r).`

3. For each strict rule $r \in R$ ($r : q_1, q_2 \ldots q_n \rightarrow p$), add to $LP(D)$ the following rule:

   `defeasibly(p) :- defeasibly(`q_1`), ... defeasibly(`q_n`).`

   The consequent of a strict rule can be derived if it's antecedent holds. Counterarguments need not be considered.

4. For each defeasible rule $r \in R$ ($r : q_1, q_2 \ldots q_n \Rightarrow p$), add to $LP(D)$ rules of the following form:

   a. `defeasibly(p) :- defeasibly(`q_1`), ... defeasibly(`q_n`), ok(r).`
   A literal p is defeasibly provable using r if each subgoal $q_i \in body(r)$ is defeasibly provable and it's *ok* to detach the head (i.e., the rule is not blocked). For each defeasible rule in D, exactly one rule of this form is added to $LP(D)$.

   b. `ok(r) :- ok(r, cs`$_1$`), ok(r, cs`$_2$`)... ok(r, cs`$_m$`).`
   where $head(r) \in cs_i$. It's ok to apply rule r if it's ok with respect to each conflict set containing the head of r (note that there will always be at least one conflict set).

   c. `ok(r, cs`$_i$`) :- blocked_literal(r, q`$_j$`).`
   where $q_j \in cs_i - (F \cup head(r))$. It's ok to apply rule r with respect to a conflict set if there's a blocked literal in the set $cs_i - (F \cup head(r))$. Note that a rule of this form will exist for each $q_j \in cs_i - (F \cup head(r))$. Facts need not be considered, since these can never be blocked.

   d. `blocked_literal(r, q`$_i$`) :-`
   `blocked(r,`r_1`),`
   `blocked(r,`r_2`),`
   `...`
   `blocked(r,`r_k`).`
   where $head(r_1) = head(r_2) \ldots = head(r_k) = q_i$, and each r_i is any sort of rule. A literal q_i is blocked with respect to a rule r if every rule r_i having it as a head is blocked by r.
   One clause of the above form will exist for each unique (r, q_i) pair, where $q_i \in cs_i - (F \cup head(r))$ and cs_i is a conflict set containing $head(r)$. The literal q_i can never be a fact of the defeasible theory, since (again) facts cannot be blocked. Also, if no rule has head q_i, then `blocked_literal(r, q`$_i$`)` is asserted as a fact of the logic program.

   e. `blocked(r,`r_i`):- not defeasibly(`s_j`).`
   where $s_j \in body(r_i)$. A rule r blocks another r_i if r_i has a subgoal s_j that is not defeasibly provable.
   If a rule r_i has no body (in the case of defeasible rules), then no clauses of form (e) will occur in $LP(D)$.

   f. `blocked(r,`r_i`):- sup(r,`r_i`).`
   A rule r also blocks another r_i if r is takes precedence over r_i.

Note that the only place negation-as-failure occurs in the translation is in 3.e. Also, for any strict or defeasible rule r of D with head p, there is a single corresponding logic program rule with head `defeasible(p)`. We refer to this corresponding rule as $trans(r)$.
In the following section, we show that the above translation process is correct, in that $D \mid\sim p$ iff `defeasible(p)` $\in wfm(LP)$, and $D \not\mid\sim p$ iff $\neg$`defeasible(p)` $\in wfm(LP)$.

A Proof of the Translation's Correctness

We begin with the 'if' direction and induct on the depth of a defeasible argument tree τ.
Let D be a defeasible theory, LP its logic program translation, and τ be a defeasible argument tree for D.

Theorem 4: *If the root of τ is labeled with $D \mid\sim p$, then* `defeasibly(p)` $\in wfm(LP)$. *If the root of τ is labeled with $D \sim\!\!\mid p$, then* $\neg$`defeasibly(p)` $\in wfm(LP)$.

(Base Case) Suppose τ is just a single node n labeled $D \mid\sim p$ or $D \sim\!\!\mid p$. We consider each case separately.

Case 1: Suppose that n is labeled $D \mid\sim p$. Then definition 6.1 holds.
Since strict rules with empty bodies are forbidden, then either 6.1.a or 6.1.c must obtain. If 6.1.a obtains, p is a fact of D and `defeasibly(p)` is a fact of LP, and so obviously `defeasibly(p)` $\in wfm(LP)$. If 6.1.c obtains, then there is some rule $r : A \Rightarrow p$ that succeeds at n, and for all conflict sets cs such that $p \in cs$, there is a literal $q \in (cs - (F \cup \{p\}))$ such that 6.1.c.i, 6.1.c.ii, and 6.1.c.iii hold. Since n has no children, A must be empty.
Let cs_i be a conflict set such that $p \in cs_i$, and $q_j \in (cs_i - (F \cup \{p\}))$ a literal with the above properties. Since n has no children, there can be no strict rules with q_j as a head. For the same reason, the body of any defeasible or defeater rule with head q_j must be empty.
Let s_k be a defeasible or defeater rule with head q_j. Since 6.1.c obtains, $s_k \prec r$. Generalizing on s_k, for all rules s with head q_j, $s \prec r$, and so `sup(r, s)` appears as a fact in LP. Given this, `blocked_literal(r, q_j)` and hence `ok(r, cs_i)` $\in wfm(LP)$.

Generalizing on cs_i, for each conflict set cs containing p, `ok(r, cs)` $\in wfm(LP)$. But if this is the case, `ok(r)` $\in wfm(LP)$, and so `defeasibly(p)` $\in wfm(LP)$.

Case 2: Suppose that n is labelled $D \sim\!\!| \; p$.
τ consists of only a single node, and so failure-by-looping cannot apply. Definition 6.2 must obtain: p is not a fact, there are no strict rules with head p (since n is a leaf node), and for all defeasible rules $r : A \Rightarrow p$, either

i A fails at n, or

ii There is a conflict set cs_i such that $p \in cs_i$, and for all $q \in (cs_i - (F \cup \{p\}))$, either,

- a There is a strict rule s: $B \rightarrow q$ such that B succeeds at n, or
- b There is a defeasible rule s: $B \Rightarrow q$ such that B succeeds at n and $s \not\prec r$, or
- c There is a defeater s: $B \rightsquigarrow q$ such that B succeeds at n and $s \not\prec r$.

Let r_i be a rule $A \Rightarrow p$. Since n is a leaf node, neither i nor $ii.a$ can obtain.
Let cs_k be a conflict set satisfying (ii) above, and let q be a member of $(cs_k - (F \cup \{p\}))$. Suppose (ii.b) or (ii.c) obtains. Then there is a rule s: $B \Rightarrow q$ ($B \rightsquigarrow q$) such that B succeeds at n and $s \not\prec r_i$. Since B must be empty, clauses of the form

```
blocked(r_i, s):- not defeasibly(q)
```

do not appear in LP. Since $s \not\prec r_i$, the clause `sup(r_i,s)` does not appear as a fact in LP, and because no rules can derive `sup(r_i,s)`, $\neg$`sup(r_i,s)` $\in wfm(LP)$. The only rule with head `blocked(r_i, s)` now has a subgoal whose complement appears in $wfm(LP)$, and so $\neg$`blocked(r_i, s)` $\in wfm(LP)$.
Similarly, since `blocked(r_i, s)` appears in the rule

```
blocked_literal(r_i, q):-
    blocked(r_i, s_1),
    blocked(r_i, s_2),
    ...
    blocked(r_i, s_k).
```

and there are no other rules with that head, $\neg$`blocked_literal(r_i, q)` $\in wfm(LP)$.
Generalizing on q, for any q_j in the set $cs_k - (F \cup \{p\}))$, $\neg$`blocked_literal(r_i, q_j)` $\in wfm(LP)$. From this it follows that $\neg$`ok(r_i, cs_k)` $\in wfm(LP)$, and hence $\neg$`ok(r_i)` $\in wfm(LP)$.
Generalizing on rule r_i, $\neg$`defeasibly(p)` $\in wfm(LP)$.

(Induction) Suppose the claim holds for trees of depth k or less and let τ be a tree of depth $k+1$.

Case 1: Suppose the root n of τ is labeled $D \mid\!\sim p$. Then 6.1 again holds. We will show that regardless of whether 6.1.a, 6.1.b, or 6.1.c is true, `defeasibly(p)` $\in wfm(LP)$:

Case 1.a: p is a fact of D, in which case `defeasibly(p)` $\in wfm(LP)$.

Case 1.b: There is a strict rule $A \rightarrow p$ such that A succeeds at n. Then for all $q \in A$, there is a child of m labeled $D \mid\!\sim q$. Each such q is the root of a valid argument tree of maximum depth less than $k+1$, and so by inductive hypothesis, `defeasibly(q)` $\in wfm(LP)$. Applying the immediate consequence operator, `defeasibly(p)` $\in wfm(LP)$.

Case 1.c: There is a rule r: $A \Rightarrow p$ such that A succeeds at n and for all conflict sets cs with $p \in cs$, there is a q in $cs - (F \cup \{p\}))$ such that:

i. For all strict rules s: $B \rightarrow q$, B fails at n,

ii. For all defeasible rules s: $B \Rightarrow q$ fails at n or else $s \prec r$, and

iii. For all defeater rules s: $B \rightsquigarrow q$ fail at n or else $s \prec r$.

Let cs_i and q_k be such a conflict set and literal, and let s be any rule with head q_k. If s fails at n, then some some child of n is labeled $D \sim\!\!| \; d$, where d is in the body of s. This is the root of a valid proof tree, and so by inductive hypothesis, $\neg$`defeasibly(d)` $\in wfm(LP)$. Based on this, `blocked(r, s)` $\in wfm(LP)$.
If s is a defeasible or defeater rule and $s \prec r$, then `sup(r,s)` is a fact of LP, and so `blocked(r, s)` is again in $wfm(LP)$.
Generalizing on s, `blocked_literal(r, q_k)` $\in wfm(LP)$, and so `ok(r, cs_i)` $\in wfm(LP)$. Generalizing on cs_i, `ok(r, cs)` $\in wfm(LP)$ for all cs in which p appears, and so `ok(r)` appears in $wfm(LP)$.
Since A succeeds at n, for all $u \in A$, there is a child of n labeled $D \mid\!\sim u$. Each such u is the root of a valid argument tree of maximum depth less than $k+1$, and so by inductive hypothesis, `defeasibly(u)` $\in wfm(LP)$.
It is now that case that every subgoal of the LP rule

```
defeasibly(p) :-
        defeasibly(u_1),
        ...
        defeasibly(u_n),
        ok(r).
```

corresponding to $r \in R_D$ is true in the $wfm(LP)$. It follows that `defeasibly(p)` $\in wfm(LP)$.

Case 2: Suppose the root n of τ is labeled $D \sim\!\!| \; p$.
For this part of the proof, we show that the collection of literals appearing in negative assertions of τ correspond to an unfounded set with respect to $wfm(LP)$.
Any branch of a proof tree involving failure-by-looping need not extend beyond the topmost node where definition 6.3 applies. As this is so, the tree can be trimmed to that point, and so 6.3 only applies to the leaves of the tree. We may assume without loss of generality that τ is of this form.
Define S and U as follows:

$S = \{n : n$ is a node of τ labeled with $D \sim\!\!| \; u$ for some $u\}$.

$U = \{$`defeasibly(u)`: $D \sim\!\!| \; u$ appears as a label for some member of $S\}$.

Let n be any node in S. Then n is labeled $D \sim\!\!| \; \phi$ for some ϕ. The following shows that all rules with head ϕ are unfounded.
Node n is either a leaf or an internal node.

Case 2.a: Suppose that n is an internal node. Then 6.2 obtains, and ϕ is not a fact.

If r is a strict or defeasible rule with head ϕ that fails at n, n has a child labeled $D \sim\!\!| \; v$, where $v \in body(r)$. By definition `defeasibly(v)` appears in U.
If r is a defeasible rule with head ϕ that does not fail at n, then by 6.2.c.ii there is a $cs \in C$ such that $p \in cs$, and for all $q \in cs - (F \cup \{p\})$, either

i. there is a strict rule $s : B \rightarrow q \in R$ such that B succeeds at n,

ii. there is a defeasible rule $s : B \Rightarrow q \in R$ such that B succeeds at n and $s \not\prec r$, or

ii. there is a defeater rule $s : B \rightsquigarrow q \in R$ such that B succeeds at n and $s \not\prec r$

Let cs_i be such a conflict set as above, q_j a member of $cs_i - (F \cup \{p\})$, and let s be a strict (defeasible, defeater) rule with head q_j such that $body(s)$ succeeds at n and $s \not\prec r$ (even if s is strict, $s \not\prec r$ holds).
Since $body(s)$ succeeds at n, n has a child labeled $D \vdash\!\!\sim u$ for each $u \in body(s)$. By inductive hypothesis, `defeasibly(u)` $\in wfm(LP)$ for each $u \in body(s)$. Since $s \not\prec r$, `¬sup(r,s)` $\in wfm(LP)$. As this is so, every rule with head `blocked_(r,s)` has a subgoal whose complement appears in $wfm(LP)$, and so `¬blocked_(r,s)` itself appears in $wfm(LP)$.
Since `¬blocked_(r,s)` $\in wfm(LP)$, `¬blocked_literal(r,q)` $\in wfm(LP)$. Generalizing on q, `¬ok(r,cs_i)` $\in wfm(LP)$, and hence `¬ok(r)` $\in wfm(LP)$.
Generalizing on rule r, every logic program rule with head `defeasibly(`ϕ`)` has a subgoal in U or else a subgoal whose complement appears in $wfm(LP)$.

Case 2.a: Suppose that n is a leaf node. Either 6.2 or 6.3 obtains. If 6.2 obtains, then as was shown in the base case, for any rule r with head `defeasibly(`ϕ`)` , `¬ok(r)` $\in wfm(LP)$. If 6.3 obtains, then there is a non-leaf node labeled $D \sim\!\!| \; \phi$, and we have shown there that all rules with head `defeasibly(`ϕ`)` have (1) a subgoal whose complement appears in $wfm(LP)$, or (2) a subgoal in U.

Given the above 2 cases, for every member `defeasibly(u)` $\in U$, each rule with head `defeasibly(u)` has a subgoal in U or else a subgoal whose complement is in $wfm(LP)$. By definition, U is unfounded with respect to the $wfm(LP)$.
As this is so, for each $u \in U$, `¬defeasibly(u)` $\in wfm(LP)$. In particular, `¬defeasibly(p)` $\in wfm(LP)$. □

For the 'only if' portion of the proof, it is convenient to transform the logic program by combining rules of the form

```
ok(r) :-
    ok(r, cs_1),
    ok(r, cs_2),
    ...
    ok(r, cs_k).
ok(r, cs_i) :-
    blocked_literal(r, q_1).
```

to create new rules such as below:

```
ok(r) :-
    blocked_literal(r, q_1),
    ...
    blocked_literal(r, q_k).
```

The old rules are deleted from the program. In each new rule, the i^{th} `blocked_literal(r, q)` corresponds to one element of the set $cs_i - (F \cup \{head(r)\})$. Intuitively, this means that rule r is 'ok' to fire if each conflict set has a blocked literal. Each rule contains a tuple from $\{cs_1 - (F \cup \{head(r)\})\} \times \{cs_2 - (F \cup \{head(r)\})\} \times \ldots \{cs_k - (F \cup \{head(r)\})\}$. Every possible combination is used in some rule.

The following lemmas are used in the proof:

Lemma 1: *If $ok(r) \in U_n$, then there exists a conflict set cs in D with $head(r) \in cs$, and for each $q \in cs - (F \cup \{head(r)\})$ there exists a rule s of D with $head(q)$ such that $s \not\prec r$ and for each $v \in body(s)$,* `defeasibly(v)` $\in \mathcal{I}_{n-1}$.

Suppose $ok(r) \in U_n$. Then every rule of the form

```
ok(r) :-
    blocked_literal(r, q_1),
    ...
    blocked_literal(r, q_k).
```

has a subgoal `¬blocked_literal(r, q`$_i$`)` in $\mathcal{I}_{n-1}$ or else in U_n. If `¬blocked_literal(r, q`$_i$`)` $\in I_{n-1}$, then `blocked_literal(r, q`$_i$`)` $\in U_{n-1}$. U is monotonic, and so we may assume wlog that `blocked_literal(r, q`$_i$`)` $\in U_n$.
Since a tuple from $\{cs_1 - (F \cup \{head(r)\})\} \times \{cs_2 - (F \cup \{head(r)\})\} \times \ldots \{cs_m - (F \cup \{head(r)\})\}$ comprises the body of each rule with head $ok(r)$, and every rule has a subgoal in U_n, it readily follows that for at least one set $cs_i - (F \cup \{head(r)\})$, every literal of $cs_i - (F \cup \{head(r)\})$ is in U_n.
Let q_j be some literal in $cs_i - (F \cup \{head(r)\})$. Since `blocked_literal(r, q`$_j$`)` $\in U_n$, it follows that the rule

```
blocked_literal(r, q_j):-
    blocked(r, s_1),
    blocked(r, s_2),
    . . .
    blocked(r, s_m).
```

has some subgoal `blocked(r, s_u)` $\in U_n$. Rules with head `blocked(r, s_u)` have `not defeasibly(v)` or `sup(r,s_u)` as their bodies, where v is some subgoal of s_u.
Applying the definition of unfoundedness, for each rule with body `not defeasibly(v), defeasibly(v)` must appear in $\mathcal{I}_{n-1}$, and so `defeasibly(v)` $\in \mathcal{I}_{n-1}$ for every $v \in body(s_u)$. If `sup(r,s_u)` appears as the body of some rule, then $sup(r, s_u) \in U_n$ (the same holds even if no rule has such a body). If that is the case, then $s_u \prec r$ does not appear in D.
Generalizing on q_j, every literal of $cs_i - (F \cup \{head(r)\})$ has a rule s with $s \not\prec r$ such that for all $v \in body(s)$, `defeasibly(v)` $\in \mathcal{I}_{n-1}$. □

Lemma 2: *If $ok(r) \in \mathcal{I}_n$, then for each conflict set cs containing $head(r)$, there exists a $q \in cs - (F \cup \{head(r)\})$ such that for all rules s with head q, either: (1) s contains a subgoal* `defeasible(v)` $\in U_{n-1}$, *or (2) $s \prec r$ is in D.*

If `ok(r)` $\in \mathcal{I}_n$, there is some rule

```
ok(r) :-
    blocked_literal(r, u_1),
    ...
    blocked_literal(r, u_k).
```

such that each subgoal is in $\mathcal{I}_m$, for some $m < n$. Each `blocked_literal(r,u_j)` appears as the head of exactly one rule of $LP(D)$:

```
blocked_literal(r, u_j):-
    blocked(r, s_1),
    blocked(r, s_2),
    ...
    blocked(r, s_n).
```

Since `blocked_literal(r,`u_j`)` $\in \mathcal{I}_m$, each subgoal `blocked(r, s_i)` must be in $\mathcal{I}_l$, for some $l < m$.
Rules with head `blocked(r, s_i)` can have `not defeasibly(v)` or `sup(r,s_i)` as their bodies. Since `blocked(r, s_i)` $\in \mathcal{I}_l$, either ¬`defeasibly(v)` $\in \mathcal{I}_k$, $k < l$ for some $v \in s_i$ or else `sup(r, s_i)` is a fact of $LP(D)$. If the former, `defeasibly(v)` $\in \mathcal{U}_k$ (and hence in U_{n-1}). If the latter, then $s_i \prec r$ must appear in D.
Generalizing on s_i, for each rule s of D with head u_j, $s \prec r$ or else for some $v \in s$, `defeasibly(v)` $\in U_{n-1}$. Generalizing on cs_i, for each conflict cs set containing $head(r)$, there exists a $u \in cs - (F \cup \{head(r)\})$ such that if rule r_i has head u, then $r_i \prec r$ or else r_i contains a subgoal in U_{n-1}. □

Theorem 5: Let $\mathcal{I}_0 = \{$`defeasible(p)` $| p \in F_D\}$ and $\mathcal{I}_{k+1} = W_{LP}(\mathcal{I}_k)$. For all $\mathcal{I}_n$, if `defeasibly(p)` $\in \mathcal{I}_n$, then $D \mid\sim p$, and if ¬`defeasibly(p)` $\in \mathcal{I}_n$, then $D \sim\!\!\mid p$.

(Base Case) Suppose `defeasibly(p)` $\in \mathcal{I}_0$. Then `defeasibly(p)` is a fact of LP and so p is a fact of D. Trivially, $D \mid\sim p$.
For any p, ¬`defeasibly(p)` $\notin \mathcal{I}_0$, and so the claim is trivially satisfied.

(Induction) Suppose that `defeasibly(p)` $\in \mathcal{I}_k$ implies $D \mid\sim p$ and ¬`defeasibly(p)` $\in \mathcal{I}_k$ implies $D \sim\!\!\mid p$, for all $k \leq n$. We will treat positive and negative literals in turn.

Case 1: Let `defeasibly(p)` $\in \mathcal{I}_{n+1}$. Then either `defeasibly(p)` is a fact of $LP(D)$ (and so obviously $D \mid\sim p$) or else there is some rule $t = trans(r)$ with head `defeasibly(p)` such that for all $q \in body(t)$, $q \in \mathcal{I}_n$.
If each subgoal of t is of form `defeasibly(q_i)`, then r is strict. By inductive hypothesis, $D \mid\sim q_i$ for each such q_i, and so for each there exists a defeasible proof tree with root labeled $D \mid\sim q_i$ (denoted $\tau_{D \mid\sim q_i}$). We may append these proofs to a node labeled $D \mid\sim p$ to form a proof of $D \mid\sim p$.
If t contains an additional subgoal $ok(r)$, then r is defeasible. Since `ok(r)` $\in \mathcal{I}_n$, by Lemma 2 it follows that for each conflict cs set containing p, there exists a $u \in cs - (F \cup \{p\})$ such that if rule r_i has head u, then $r_i \prec r$ or else r_i contains a subgoal `defeasible(v)` in U_{n-1}. By inductive hypothesis, a refutation proof exists for each v. Where applicable, we may append these to a root node to show $D \mid\sim p$.
Since rule t must correspond to either a defeasible or strict rule of r, we may conclude $D \mid\sim p$.

Case 2: Suppose that ¬`defeasibly(p)` $\in \mathcal{I}_{n+1}$. Then by definition `defeasibly(p)` $\in U_{n+1}$. It immediately follows that p is not a fact of D.
We construct a series of trees as follows. Let τ_0 be the tree consisting of a single unmarked node labeled $D \sim\!\!\mid p$. For any $n > 0$, pick an unmarked leaf node x in τ_n. From the definition of τ_0 and the cases below, we will see that x is labeled $D \sim\!\!\mid q$ and `defeasibly(q)` $\in U_{n+1}$. Then for each rule r with head q, $trans(r)$ has a subgoal s such that $\neg s \in \mathcal{I}_n$, or else $s \in U_{n+1}$. Either $s =$ `defeasibly(t)` for some $t \in body(r)$ or $s = ok(r)$. Consider each posibility in turn.
Case 2.a: $s = ok(r)$ and $ok(r) \in U_{n+1}$. By Lemma 1, there exists a conflict set cs in D with $q \in cs$, and for each $u \in cs - (F \cup \{q\})$, there exists a rule r' with head u such that $r' \not\prec r$ and `defeasibly(v)` $\in I_n$ for each v in the body of r'. By inductive hypothesis, for each such v, $D \mid\sim v$. For each $u \in cs - (F \cup \{q\})$, if its associated rule s has a nonempty body, append proof trees for its body to x and mark every node of each such subtree.
Case 2.b: $s = ok(r)$ and $\neg ok(r) \in \mathcal{I}_n$. By definition, $ok(r) \in \mathcal{U}_n$. Since U is monotonic, $ok(r) \in U_{n+1}$, and we make the same additions to τ_n.
Case 2.c: $s =$ `defeasible(t)` and ¬`defeasible(x)` $\in \mathcal{I}_n$. Then `defeasible(t)` $\in U_n$, and by inductive hypothesis $D \sim\!\!\mid t$. Append to x a proof tree for $D \sim\!\!\mid t$ and mark every node of this subtree.
Case 2.d: $s =$ `defeasible(t)` and `defeasible(t)` $\in U_{n+1}$. Append to x a node y labled $D \sim\!\!\mid t$. If y satisfies condition i or 3 in Definition 6, then mark y. Otherwise, leave y unmarked.
After applying one of the cases 2.a-2.d for each rule r with head q, examine the resulting tree to see if there is an unmarked non-leaf node z in the tree such that all the children of z are marked. If such a node z is found, mark it. Repeat this procedure until there are no more unmarked nodes in the tree all of whose children are marked. The resulting tree is τ_{n+1}.
Let $\tau = \bigcup_{i=0}^{\infty} \tau_i$.
Suppose x is a marked node in τ. If x was added to τ using cases 2.a, 2.b, or 2.c, then x occurs within a subtree of τ that is a proof tree. So x must satisfy one of the conditions in Definition 6. If x was added to τ and marked according to case 2.d, then x is a leaf node in τ and x satisfies condition 2 or 3 of Definition 6. Otherwise, x is a non-leaf node in τ, x was added to τ using condition 2.d, and x was marked because all of its children were marked. Looking at the four cases used to add the children of x to τ, we see that x must satisfy condition 2 in Definition 6. So if τ is finite and if every node in τ is marked, then τ is a proof tree.
Since D contains only finitely many rules, the branching fac-

tor for constructing τ must be finite. So if τ is infinite, then τ must have an infinitely long branch. Consider such a branch. Every node in this branch (other than the top node) must have been added using case 2.d since all the other branches add proof trees which are finite. So every node in the branch must be labeled $D \sim\!\!| \; q$ for some literal q. Furthermore, no node in the branch satisfies condition 3 in Definition 6 since if it did, it would have been marked when it was added to τ and it would therefore have no children. But since D is finite, only finitely many literals occur in D. So there must be some literal q such that two different nodes in our infinite branch are labeled $D \sim\!\!| \; q$. But then one of these two nodes does satisfy condition 3 of Definition 6, which is a contradiction. Therefore, τ is not infinite.
Since τ is not infinite, we can let n be a non-negative integer such that $\tau = \tau_n$. Suppose τ_n has an unmarked node. Since a node must be marked if all its children are marked, τ_n must have an unmarked leaf node x. This node must have been added by case 2.d of our construction, and so we can let q be a literal such that x is labeled $D \sim\!\!| \; q$, and `defeasible(q)` $\in U_{n+1}$. Since x is not marked, it satisfies neither condition 2 or 3 of Definition 6. Since `defeasible(q)` $\in U_{n+1}$, $q \notin F$. If there is no rule $r \in R$ such that $head(r) = q$, then x satisfies condition 2 of Definition 6. So there is a rule $r \in R$ such that $head(r) = q$, and one of the cases 2.a-2.d applies to x. So there must be some $m > n$ such x has a child node in τ_m. Then x is not a leaf node in τ_m and x is not a leaf node in τ, a contradiction. Therefore, every node in τ satisfies some condition in Definition 6 and τ is a proof tree.□

References

Antoniou, G., and Maher, M. J. 2002. Embedding defeasible logic into logic programs. In *Proceedings of ICLP*, 393–404.

Donnelly, S. 1999. *Semantics, Soundness, and Incompleteness for a Defeasible Logic*. Masters thesis, The University of Georgia.

Emden, M. H. V., and Kowalski, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM* 23:733–742.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. 1988. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings 7th ACM Symposium on Principles of Database Systems*, 221–230.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 221–23.

Kunen, K. 1987. Negation in logic programming. *Journal of Logic Programming* 4:289–308.

Makinson, D., and Schechta, K. 1991. Floating conclusions and zombie paths: two deep difficulties in the 'directly skeptical' approach to inheritance nets. *Artificial Intelligence* 48:199–209.

Nute, D. 1992. Basic defeasible logic. In del Cerro, L. F., and Penttonen, M., eds., *Intensional Logics for Programming*. Oxford University Press. 125–154.

Nute, D. 1994. Defeasible logic. In Gabbay, D., and Hogger, C., eds., *Handbook of Logic for Artificial Intelligence and Logic Programming, Vol. III*. Oxford University Press. 353–395.

Nute, D. 1997. Apparent obligation. In Nute, D., ed., *Defeasible Deontic Logic*, Synthese Library. Dordrecht, Netherlands: Kluwer Academic Publishers. 287–315.

Nute, D. 2001. Defeasible logic: Theory, implementation, and applications. In *Proceedings of INAP 2001, 14th International Conference on Applications of Prolog*, 87–114. Tokyo: IF Computer Japan.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

3.7 ProLogICA: a practical system for Abductive Logic Programming

ProLogICA: a practical system for Abductive Logic Programming

Oliver Ray[*]
Imperial College London, UK
email: or@doc.ic.ac.uk

Antonis Kakas[†]
University of Cyprus, Cyprus
email: antonis@cs.ucy.ac.cy

Abstract

This paper presents a new system called *ProLogICA* for Abductive Logic Programming (ALP) with Negation as Failure (NAF) and Integrity Constraints (ICs). The system builds upon existing ALP techniques but includes several optimisations and extensions necessitated by recent applications in computational biology, temporal reasoning and machine learning. Unlike some other ALP systems that support non-ground abduction through the integrated use of constraint solving, we adopt a more lightweight approach which avoids this complexity at the expense of only computing ground hypotheses. We argue our approach is suited to a wide class of real-world problems and demonstrate the effectiveness of *ProLogICA* on three non-trivial applications.

Introduction

The utility of Abductive Logic Programming (ALP) for knowledge representation and problem solving with Negation as Failure (NAF) and Integrity Constraints (ICs) is widely accepted (Kakas, Kowalski, & Toni 1992; Kakas & Denecker 2002). Moreover, state-of-the-art ALP systems, such as the *A-system* (Kakas, Van Nuffelen, & Denecker 2001) and *CIFF* (Endriss *et al.* 2004) also incorporate powerful constraint solvers that enable the inference of non-ground abducibles and the solution of complex optimisation problems characteristic of some application domains. These systems are particularly effective in constraint-based tasks such as planning (Van Nuffelen & Denecker 2000) and scheduling (Kakas & Michael 2001).

But many applications do not require complex constraint handling capabilities and simpler ALP procedures with lower computational overheads are often appropriate. This is especially true of recent machine learning techniques, such as *Hybrid Abductive Inductive Learning* (HAIL) (Ray 2005), that integrate abduction and induction in a common reasoning framework. In this task, (i) the availability of an inductive reasoning module for generalising abductive explanations means that ground based abduction is sufficient, (ii) the use of standard Prolog as a representation language and execution model means that support for Prolog libraries and the efficient processing of logical integrity constraints is more useful than constraint solving, and (iii) since ALP is just one of part in a much larger system, low overheads and good performance are paramount.

This paper presents a practical lightweight ALP system called *ProLogICA*[1] that was first developed as the abductive component of HAIL, but has also been applied in three real-world applications involving temporal event calculus reasoning (Alberti *et al.* 2005), the inference of genetic regulatory networks (Papatheodorou, Kakas, & Sergot 2005), and a novel form of "in-silico genotyping" for predicting HIV drug resistance (Ray *et al.* 2006). In fact, because these applications were carried out in parallel with the development of *ProLogICA*, each of them prompted a number of developments that were incorporated into the computational model of the resulting procedure.

ProLogICA is closely based on the ALP procedure of Kakas and Mancarella (KM) (Kakas & Mancarella 1990a), but includes several optimisations and extensions that were necessitated by the three applications mentioned above. The optimisations are concerned with pruning redundant branches from abductive and consistency computations, whereas the extensions are concerned with overcoming the representational restrictions of the KM procedure and avoiding the computation of non-minimal solutions. The system also supports dynamic integrity constraints, which are generated on-the-fly as the computation unfolds, in order to avoid floundering in consistency computations.

Improved performance is realised, through a syntactic analysis of the ALP theory to identify sources of determinism that can be exploited and, also, through run time heuristics for literal selection and pruning. Enhanced functionality is provided by preprocessing the ALP theory to overcome the representational restrictions of KM and, additionally, by the provision of metalogical operators that enable a finer degree of control over the search space. *ProLogICA* supports most Prolog built-in predicates and offers a facility for depth bounded computation. These features may be customised by the user via system parameters described in this paper.

[*]Part of this work was completed while visiting the Department of Computer Science at the University of Cyprus.

[†]Part of this work was completed while visiting the Department of Computing at Imperial College London.

[1]"*ProLogICA*" stands for "Prolog with Integrity Constraints and Abduction", but is also obtained by rearranging the capitalised letters in "Abductive LOGIC PROgramming". The system can be downloaded from http://www.doc.ic.ac.uk/˜or/proLogICA.

ALP

ALP is an extension of normal logic programs for reasoning with incomplete information. As in conventional logic programming, ALP seeks to establish the conditions under which a goal follows from a theory. But, in addition to computing a set of bindings for the variables in the goal, ALP returns a set of ground atoms that can be added to theory to ensure the goal succeeds. These atoms are usually drawn from a predefined set of ground atoms, called *abducibles*, which represent those facts for which there is only partial information in the form of integrity constraints.

Semantics

Formally, an abductive theory is a triple (P, IC, A) comprising a normal logic program P (domain knowledge), a set of formulae IC (integrity constraints), and a set of ground atoms A (abducibles). A goal G is a set of literals and an abductive explanation of G with respect to (P, IC, A) is a set of atoms $\Delta \subseteq A$ such that $P \cup \Delta \models^* \exists G \wedge \forall IC$, where $\models^*$ denotes the satisfaction of the formula on the right in a *stable model* of the program on left. In the terminology of (Kakas & Mancarella 1990b), G is said to be satisfied in a *Generalised Stable Model* (GSM) of (P, IC, A).

To discriminate between alternative abductive explanations, additional preference criteria are often utilised. Two popular desiderata are *minimality* and *basicality*. Formally, an explanation Δ of G with respect to (P, IC, A) is *minimal* iff there is no $\Delta' \subset \Delta$ such that Δ' is also an explanation of G, and is *basic* iff there is no $\Delta' \not\supseteq \Delta$ such that Δ' is also an explanation of Δ. Intuitively, an explanation Δ is minimal if none of its atoms are redundant; and it is basic if none of its atoms can be further explained.

Procedure

The KM procedure is a standard ALP technique that computes abductive explanations by interleaving abductive derivations (in which abducibles are assumed) and consistency derivations (in which consistency is enforced). Given a theory (P, IC, A) and a goal G, the KM procedure starts an abductive derivation by unfolding the goal G against the program P in Prolog fashion until an abducible a is selected. At this point, a consistency derivation is invoked to see if the atom a can be added to the initially empty hypothesis Δ without violating the integrity constraints IC.

A consistency derivation comprises one separate branch for each resolvent of the integrity constraints and the abducible a under investigation. Each such resolvent is regarded as a query that must be shown to fail in order for the integrity check, as a whole, to succeed. This is established by repeatedly resolving on selected literals in all possible ways and, if necessary, assuming further abducibles (that are carried over to any remaining branches of the computation).

When all branches of the consistency derivation have been closed, the outer abductive computation continues with a and any other abducibles assumed in the consistency computation added to Δ – indicating that all subsequent calls to these abducibles should immediately succeed. If any branch of the consistency derivation cannot be closed, then the outer abductive computation is failed and the backtracking mechanism is invoked.

Negative literals, which are preceded by the connective not, are treated as abducibles subject to the (implicit) integrity constraint $\forall(not\, b \leftrightarrow \neg b)$. Thus, whenever a negative literal $not\, b$ is selected in an abductive (resp. consistency) derivation, it is assumed (resp. failed) by the KM procedure subject to there being a successful consistency (resp. abductive) derivation for the positive literal b – possibly resulting in deeply nested consistency and abductive computations.

Example

The KM procedure is best illustrated by an example. Fig. 1 shows an abductive theory describing the lactose metabolism of the bacterium E. Coli. The program P models the fact that E. coli can feed on the sugar lactose (`lact`) if it makes two enzymes permease (`perm`) and galactosidase (`gala`). Like all enzymes (`E`), these are made if they are coded by a gene (`G`) that is expressed. These enzymes are coded by two genes (`lac(y)` and `lac(z)`) in cluster of genes (`lac(X)`) – called an *operon* – that is only expressed when the amounts of glucose (`gluc`) are low (`lo`) and lactose are high (`hi`).[2] The abducibles A declare all ground instances of the predicates `amt` and `sugar` as assumable. The integrity constraints state that the amount of a substance (`S`) may not be both high and low; and can only be known if the substance is a sugar. (The atom `ic` denotes logical falsity, so the first constraint is equivalent to $\forall_S : \neg(amt(S, lo) \wedge amt(S, hi))$), while the second constraint is equivalent to $\forall_{S,V} : amt(S, V) \rightarrow sugar(S)$.)

```
%------ Domain Knowledge (P) ------%
feed(lact):-make(perm),make(gala).
make(E):-code(G,E),express(G).
express(lac(X)):-amt(gluc,lo),amt(lact,hi).
code(lac(y),perm).
code(lac(z),gala).

%--- Integrity Constraints (IC) ---%
ic :- amt(S,lo), amt(S,hi).
ic :- amt(S,V), not sugar(S).

%--------- Abducibles (A) ---------%
abducible_predicate(amt).
abducible_predicate(sugar).
```

Figure 1: ALP model of lactose metabolism regulation in E. coli. — Simplified from the case study in (Ray 2005).

The KM computation resulting from the goal `feed(lact)` is shown in Fig. 2. In the notation of (Kakas & Mancarella 1990a), abductive derivations are enclosed in a single-line boxes, while consistency computations are enclosed in double boxes. The initial goal is

[2] Biologically, this reflects the fact that glucose is a preferred food source that E. coli metabolises more efficiently than lactose.

shown at the very top of the computation. The next four goals are obtained by unfolding the top goal Prolog-style by resolving with the clauses in the program P.

Upon selecting `amt(gluc,lo)`, KM adds this abducible to the initially empty hypothesis Δ and begins the first consistency derivation. Since the abducible resolves with both integrity constraints, there are two branches, which, for convenience, are separated by a double horizontal line. The first must show the failure of `amt(gluc,hi)`, which it does by abducing its negation `not amt(gluc,hi)`.

Adding this to Δ, KM proceeds with the second branch, which must show the failure of `not sugar(gluc)`. This means showing `sugar(gluc)` by a subsidiary abductive computation. Since it is abducible, this is assumed subject to its own integrity check. But, as it does not trigger any integrity constraints, the innermost consistency derivation trivially succeeds; and, hence, so do the enclosing abductive and consistency derivations.

Note that a double box succeeds when all of its branches are disproved (denoted ■), whereas a single box succeeds when all of its goals are proved (denoted □). Note also how the hypothesis Δ grows monotonically throughout the computation. For compactness, we only show the hypothesis at the beginning and end of each (branch of a) consistency derivation. It is assumed the final hypothesis is exported to the enclosing abductive derivation.

Having abduced `amt(gluc,lo)`, `sugar(gluc)` and `not amt(gluc,hi)`, KM resumes the outer abductive computation, where the goals `amt(lact,hi)` and `make(gala)` are pending. The former results in a consistency derivation analogous to that described above, but the latter succeeds the already abduced atoms `amt(gluc,lo)` and `amt(lact,hi)` without needing to perform any additional consistency checks.

This example shows the top-down nature of KM and its handling of negation through subsidiary computations like NAF in conventional Prolog. It also illustrates the main strengths of this procedure, which are (i) the interleaving of abductive and consistency computations so that integrity violations are detected as soon as they arise, and (ii) the avoidance of excessive integrity checking by recording the *absence* of abducibles selected in consistency computations and by checking only those constraints that resolve with the abducible being assumed.

To avoid the complex integrity checking procedures needed to handle existentially quantified variables in abducibles, KM is committed to selecting ground abducibles only. This means KM must explore all ways of failing a constraint which may lead to variable bindings that result in the grounding of abducibles. However, experience shows that existing implementations of KM generate excessive numbers of choice points, which makes them too inefficient for the applications described in this paper. In addition, as discussed in the next section, KM imposes two restrictions upon ALP theories that are not always appropriate, and it provides no mechanisms for handling the large number of explanations that can arise in practical applications.

We have developed *ProLogICA* to address these issues.

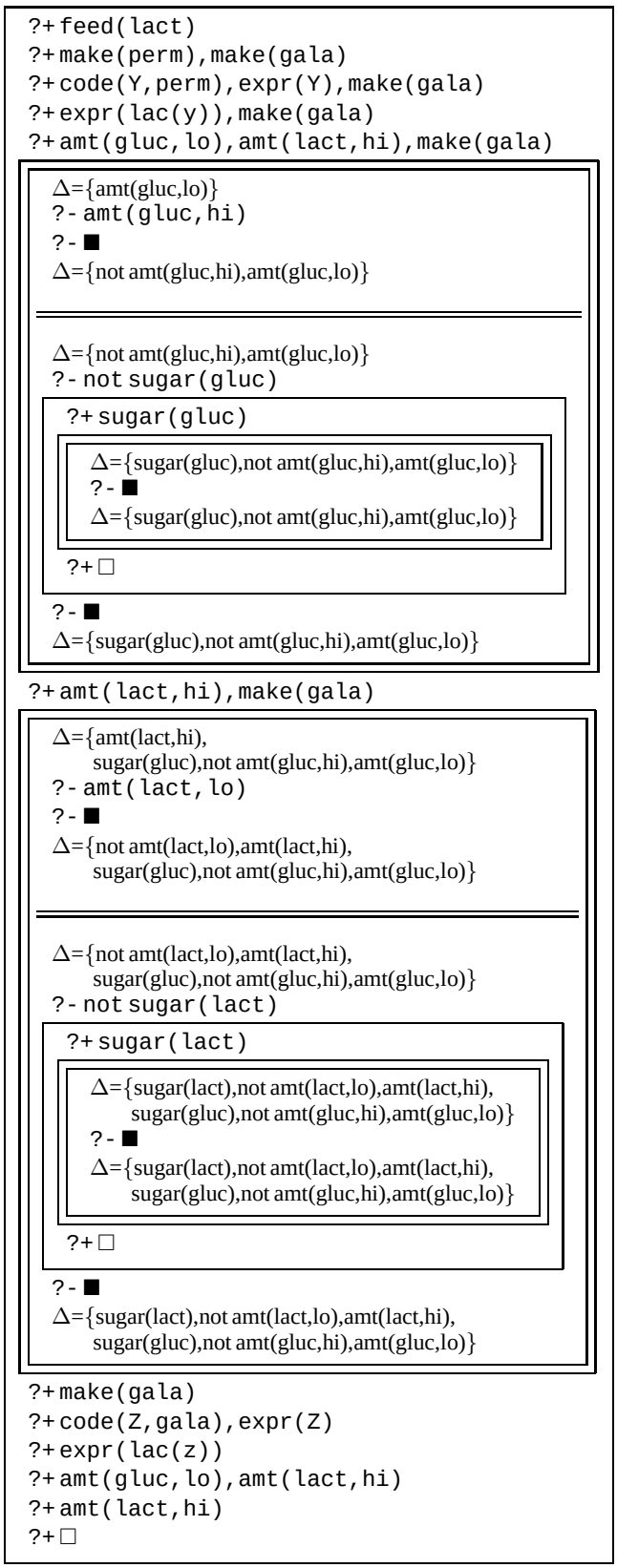


Figure 2: Successful KM computation for the query `feed(lact)` using the ALP theory in Fig. 1.

ProLogICA

Initialisation

ProLogICA is a meta-interpreter for SICStus Prolog 3.11.2. The code is contained in a single Prolog file `alp.pl` and is invoked with the command `sicstus -l alp`. A list of available commands is obtained by typing `help`. An ALP theory to be interpreted is loaded with the command `file('fname')`. If necessary, the working directory is set with the command `path('dirpth')`. The object file syntax is shown in Fig. 1. Negative literals may appear in the bodies of clauses using the operator `not/1` (which may not be nested). Integrity constraints are clauses with the head atom `ic`. Abducible predicates are declared by `abducible_predicate/1`. Built-in Prolog predicates can be used, but impure ones should be treated with care. Control primitives like cut and disjunction are not supported. All non-built-in predicates should be declared dynamic: e.g. with the following declaration for the code in Fig. 1:

```
:-dynamic feed/1,make/1,code/2,express/1,
amt/2,sugar/1,abducible_predicate/1,ic/0.
```

Invocation

Abductive queries are invoked through the predicate `demo/2`. There first argument should be instantiated to a list of goals. The second argument will become bound to a list of abducibles. Thus, running the query `?-demo([feed(lact)],D)` on the code in Fig. 1 results in the answer `D=[sugar(lact), not amt(lact,lo), amt(lact,hi), sugar(gluc), not amt(gluc,hi), amt(gluc,lo)]`. The Prolog backtracking mechanism will search for alternative explanations by entering a ";". If `eval/2` is used in place of `demo/2`, it will first compute all explanations and then return only the minimal ones.

Preprocessing

The KM procedure relies heavily on two representational restrictions: (i) that all integrity constraints contain an abducible predicate and (ii) that no abducible predicates are defined in the theory. These assumptions allow KM to avoid repeatedly re-checking all of the integrity constraints every time an atom is abduced. From a knowledge representation perspective, these restrictions can be motivated by arguing that abducibles are predicates whose extents are completely unknown except for the integrity constraints.

However, these assumptions are not always practicable as the first is often violated when the application provides partial knowledge of abducible predicates, and the second is often violated when the application suggests general domain integrity constraints that do not explicitly mention any abducible predicates. Moreover, in situations that involve refining incomplete theories (Ray 2005), these assumptions are particularly inconvenient.

Thus, *ProLogICA* provides a preprocessor that transforms any abductive theory into a theory satisfying (i) and (ii). In fact, it is well known that (ii) can always be ensured by replacing each abducible predicate `p/n` by a new abducible predicate `p'/n` and adding a clause to the program of the form `p(X1,...,Xn):- p'(X1,...,Xn)`. Since, this usually leads to the violation of (i), *ProLogICA* repeatedly unfolds each constraint with no abducibles by resolving on a selected atom with the program clauses.

Since, in general, it may be impossible to ensure that all constraints derived in this way contain abducibles, after a preset number of steps, *ProLogICA* collects those with abducibles, as these can be efficiently processed during the computation, and leaves the others to be checked at the end. The transformed clauses can be seen by typing `list` and can be recomputed with the command `reload`.

Features

ProLogICA offers many features not found in existing KM implementations. To ensure termination, derivations are depth bounded. To avoid floundering in consistency computations, dynamic integrity constraints are issued (Kakas, Michael, & Mourlas 2000). A selection policy is used that preferentially selects ground atoms, evaluates built-ins as soon as they are executable, and processes assumed or inconsistent abducibles when they become ground. A meta-predicate `atleast/2` is offered for ensuring the success of N goals from a given list whilst assuming the fewest number of abducibles. Efficient pruning mechanisms are also provided, as described below.

Settings

Customisation of *ProLogICA* is achieved via user-definable system parameters whose values are set with the command `set(param,value)`. Boolean settings take the values `true` or `false`, while numeric settings take integers. The parameters are summarised below (with default values).

max_ab_depth (70): bounds the combined depth of all abductive derivations in a computation.

max_con_depth (50): bounds the individual depth of each consistency derivations in a computation.

max_ic_unfold (5): bounds the number of times integrity constraints are unfolded during preprocessing. Setting this to 0 disables integrity unfolding, but still performs a consistency check at the end of each computation. Setting this to a negative value disables all pre and postprocessing of integrity constraints and abducibles.

enable_pruning (true): activates pruning mechanisms that effect a substantial reduction in the number of redundant branches in the search space at the expense of a negligible loss of performance on non-redundant branches. This is done mainly by eliminating unnecessary backtracking points in consistency derivations.

exploit_determinism (true): uses so-called *closed predicates*, which do not depend on any abducibles, to enable further pruning of consistency computations. Although it reduces execution times, this option can lead to a loss of completeness if the program contains positive cycles.

attempt_minimal (false): To avoid computing non-minimal solutions, *ProLogICA* offers a heuristic search strategy that attempts to minimise the number of literals

in each explanation. This almost always reduces the number of non-minimal hypotheses, but can reduce or increase the execution time according to context.

show_negatives (false): When true, this option prevents negative literals being removed from computed explanations during postprocessing.

show_constraints (false): When true, this option results in dynamic integrity constraints being appended on to computed explanations during postprocessing.

show_time (false): When set, this option results in execution times being appended on to computed explanations during postprocessing.

debug_info (false): When true, this option results in minimal debugging information being written onto the user output during execution.

Limitations

ProLogICA avoids the need for more complex constraint solving techniques by precluding the selection of non-ground abducibles in abductive computations. This brings with it the possibility of floundering - where a branch of the search space cannot be explored as none of the goal literals can be selected. In consistency computations floundering is avoided through the use of dynamic integrity constraints. In abductive computations floundering is partially avoided by postponing the selection on non-ground abducibles. But, the programmer must ensure that variables are sufficiently grounded so floundering does not prune useful solutions. (This is analogous to the use of standard Prolog without constructive negation.) *ProLogICA*'s preprocessing facility does not use clever heuristics in the unfolding of integrity constraints. No subsumption checking is performed when issuing dynamic integrity constraints. More user-friendly debugging and visualisation tools are needed.

Applications[3]

Temporal Reasoning in Multi-Agent Systems

ProLogICA has been used within a multi-agent systems project called *Societies of Computees* (SOCS) (Alberti *et al.* 2005; Stathis *et al.* 2004) to implement the Temporal Reasoning (TR) component of its computees (autonomous agents). This aspect of the computee architecture allows an agent to maintain a coherent model of its evolving environment and infer temporal information missing from its necessarily incomplete view of the external world.

The TR representation is based on a variant of the Abductive Event Calculus (Eshghi 1988; Denecker, Missiaen, & Bruynooghe 1992; Shanahan 2000), based on the Language $\mathcal{E}$ (Kakas & Miller 1997), which models how the truth or falsity of certain fluents (properties) vary over time as a result of certain events (actions) happening, or not. Atoms of the form $holds\_at(F,T)$ denote that the fluent F is true at time T, while atoms of the form $happens(A,T)$ denote that the action A occurs at time T.

The abductive theory is formed of two parts. The "domain independent" axioms, shown below, formalize how fluents are caused by events and how they persist forwards in time. For an agent to believe a fluent F holds at time T, either (i) an event A previously occurred that initiated F, or (ii) F was observed to hold in the past, or (iii) F is assumed (abduced) to hold initially – providing that no intervening event terminates F. Negative fluents are represented $neg(F)$ and are treated symmetrically.

$$holds\_at(F,T) \leftarrow happens(A,T_1), \\ T_1 < T, initiates(A,T_1,F), not\ clipped(T_1,F,T).$$

$$holds\_at(F,T) \leftarrow observed(F,T_1), \\ T_1 \leq T, not\ clipped(T_1,F,T).$$

$$holds\_at(F,T) \leftarrow assume\_holds(F,0), \\ not\ clipped(0,F,T).$$

$$clipped(T_1,F,T_2) \leftarrow happens(A,\ T), \\ T_1 \leq T < T_2, terminates(A,T,F).$$

The second part of the theory contains the domain specific axioms stating which actions initiate and terminate which fluent under which conditions. For example, if our domain involves the parking of cars in a car park, we may have the following axioms – stating that the action of parking a car will result in the car being in the car park if there is a free place or, even if their is no free space, will still result in the car being in the car park providing there is a parking attendant (with access to specially reserved places):

$$initiates(park(Car),T,in\_car\_park(Car)) \leftarrow \\ holds\_at(free\_place,T).$$

$$initiates(park(Car),T,in\_car\_park(Car)) \leftarrow \\ holds\_at(neg(free\_place),T), \\ holds\_at(park\_attendant,T).$$

There is one domain independent integrity constraint

$$\bot \leftarrow holds\_at(F,T), holds\_at(neg(F),T)$$

stating that a fluent and its negation cannot hold at the same time; and there may be other domain specific integrity constraints constraining evolution of one or more fluents over time. There is just one abducible predicate $assume\_holds$ expressing the fact that we have incomplete information only for fluents. As the computee operates, it also receives information from its environment in the form of a narrative containing observations about particular fluents holding and certain events happening at specific times. For example, an agent may get the following narrative.

$$observed(park\_attendant,1).$$

$$happens(park(car1),3).$$

After adding these observations to its knowledge base, the agent uses an abductive engine to infer missing knowledge about the evolution of fluents. In the example above the agent should infer that at time 3 the attendant and car are in the car park, but there may or may not

[3] In contrast to the previous two sections, this section uses the notation of classical logic to express clauses and theories: with the connective $\leftarrow$ in place of `:-` and with $\bot$ in place of `ic`.

Query	0	1	2	3	4	5	6	7	8	9	Avg.
CIFF (v2)	54.2	52.1	95.5	44.3	94.3	45.0	44.7	49.8	101	101	68.3
ProLogICA (v0)	1.17	1.38	3.31	2.12	2.51	2.19	2.22	2.37	2.27	2.29	2.18
ProLogICA (v1)	0.16	0.16	0.28	0.16	0.17	0.16	0.16	0.17	0.17	0.17	0.18

Table 1: Execution time (in seconds) of CIFF vs. *ProLogICA* for 10 queries in the SOCS temporal reasoning domain. — Experiments using the ALP model developed in (Alberti *et al.* 2005).

be a free place. Thus SOCS agents distinguish sceptical conclusions like $holds\_at(in\_car\_park(car1), 3)$ and $holds\_at(park\_attendant, 3)$ from credulous conclusions like $holds\_at(free\_place, 3)$. Formally a ground atom $holds\_at(f, t)$ is *credulously* entailed iff it has an abductive explanation; and it is *sceptically* entailed iff it is credulously entailed but its complement $holds\_at(neg(f), t)$ is not.

The SOCS consortium has implemented a computational agent platform, called PROSOCS (Stathis *et al.* 2004), that is based on the SOCS architecture. Since this platform is parametric on the underlying abductive procedure, it has enabled the SOCS consortium to carry out experiments with different abductive systems in order to achieve the most effective setup. Such experiments have compared the performance of *ProLogICA* with that of an alternative ALP system, CIFF (Endriss *et al.* 2004), which is used elsewhere in the SOCS project for agent planning tasks.

Table 1 compares the execution times of *ProLogICA* and CIFF for 10 different queries in a test-bed domain. The table shows execution times, in seconds, on a Pentium III desktop PC. For completeness, Table 1 includes data for two versions of *ProLogICA*: the latest version (v1) and an earlier version (v0) that was originally used in (Alberti *et al.* 2005) under the name ASLD(N,IC). On average, *ProLogICA* (v0) was 30 times faster than CIFF (v2), and the *ProLogICA* (v1) was 300 times faster than CIFF (v2).

According to (Alberti *et al.* 2005), *ProLogICA* is marginally slower than an unsound optimisation of CIFF, called *triggering*, that involves suppressing the unfolding of any $holds\_at$ atoms that appear within an integrity constraint. But, while their performance is comparable, *ProLogICA* without triggering produced only correct answers, while CIFF with triggering did not. Overall, *ProLogICA* was found to exhibit a high level of robustness, so that, in spite of its lightweight nature, it was found to be "significantly more effective" in this application.

Inference of Genetic Regulatory Networks

ProLogICA has been applied to the task of inferring gene interactions using data from microarray experiments on M. tuberculosis using the method developed in (Papatheodorou, Kakas, & Sergot 2005). Published experimental data measures changes in levels of gene expression in response to genetic modifications, such as knocking-out (turning off) or over-expressing (turning on) genes, and environmental stresses, such as changes in temperature or nutrient concentration. The aim of these experiments is to obtain insights into the complex feedback mechanisms by which the product of one gene affects the expression of another. As these interactions cannot be observed directly, they must be inferred indirectly from gene expression levels; and the complexity of interactions and volume of experimental data means that automatic methods are needed in this task.

As explained in (Papatheodorou, Kakas, & Sergot 2005), statistical analysis of the raw microarray data from each experiment reveals significant changes in the expression of particular genes. These differences are represented by atoms of the form $increases\_expression(E, G)$ and $reduces\_expression(E, G)$, where E is an experiment and G is a gene. Similarly, the experimental conditions are recorded by atoms of the form $knocks\_out(E, G)$ and $over\_expresses(E, G)$. Given a set of experimental observations and the general model of gene interactions described below, *ProLogICA* was used to abduce atoms of the form $induces(F, G)$ or $inhibits(F, G)$ stating that one gene F induces (increases) or inhibits (reduces) the expression of another G. These elementary hypotheses can then be built into paths and networks of gene interactions.

The ALP theory developed by (Papatheodorou, Kakas, & Sergot 2005) models the general principles by which changes in gene expression can be explained by possible gene interactions. Because these principles are (assumed to be) independent of any particular network topology, they are succinctly and modularly expressed commonsense laws such as one below, which states that an increase in the expression of a gene X in an experiment E can be explained by hypothesising that another gene G, which is knocked-out in E, is an inhibitor of X, providing that there are no other effects that could better account for the increase of X. Here, $incr\_affected\_by\_other\_gene(E, G, X)$ means that the increase of X in E is due to some gene other than G and $incr\_affected\_by\_EnvFact(E, X)$ means it is due to an environmental change.

$$
\begin{aligned}
&increases\_expression(E, X) \leftarrow \\
&\quad knocks\_out(E, G), inhibits(G, X), \\
&\quad not\ incr\_affected\_by\_other\_gene(E, G, X), \\
&\quad not\ incr\_affected\_by\_EnvFact(E, X).
\end{aligned}
$$

$$
\begin{aligned}
&incr\_affected\_by\_other\_gene(E, G, X) \\
&\quad reduces\_expression(E, Gx), \\
&\quad Gx \neq X, Gx \neq G, \\
&\quad related\_genes(Gx, G), inhibits(Gx, X).
\end{aligned}
$$

As formalised in the second rule above, an alternative explanation for the increase in the expression of X could be some other gene Gx whose expression is reduced in experiment E and which inhibits X. The $related\_genes$ predicate is one of several mechanisms in the model for declaratively constraining the search space by exploiting background

Query	0	1	2	3	4	Avg.
KM	>7000	>7000	>7000	>7000	>7000	>7000
ProLogICA (v0)	0.47	12.4	12.4	281	282	118
ProLogICA (v1)	0.03	0.08	0.08	0.16	0.13	0.09

Table 2: Execution time (in seconds) of KM vs. *ProLogICA* for 5 queries in the genetic regulatory network domain. — Experiments using the ALP model developed in (Papatheodorou, Kakas, & Sergot 2005).

biological information about the relationships between different genes. There are about a dozen background rules covering the different combinations of conditions that could arise. The model also contains several integrity constraints like the one below which states that a gene cannot not both induce and inhibit another. Other constraints, not shown, restrict the interactions of genes located on the same operon.

$$\bot \leftarrow induces(F, G), inhibits(F, G)$$

Given this simple model of gene interactions *ProLogICA* was used to to analyse data from 5 microarray experiments involving genes thought to be implicated in the response of M. tuberculosis to heat shock. (Papatheodorou, Kakas, & Sergot 2005) report how the system was able to rediscover several gene interactions already known in the literature and to suggest further experiments for investigating other possible integrations. In addition, the highly recursive nature of the model provided a challenging application on which to evaluate the efficiency of *ProLogICA*. Table 2 shows the time taken for *ProLogICA* compared to a standard implementation of KM (Kakas & Mancarella 1998) – similarly depth-bounded for a fair comparison – to find all solutions to 5 different queries each containing up to 5 atoms. Whereas KM did not terminate on any of the queries (or return any answers) after two hours, *ProLogICA* computed all minaimal solutions in less than two tenths of a second.

Clinical Management of HIV/AIDS

ProLogICA has been used in a ALP approach for assisting medical practitioners in the selection of anti-retroviral drugs for patients infected with Human Immunodeficiency Virus (HIV) (Ray *et al.* 2006). The difficulty is the ability of HIV to accrue genetic mutations that confer resistance to known medications. For this reason, clinical guidelines advocate the use of laboratory *resistance tests* to help identify which mutations a patient is carrying and predict which drugs they are most likely resistant to. But, these tests have several drawbacks: first, they cost between one and two dollars; second, they require medical access that most infected individuals do not have; and third, they cannot reliably detect *minority strains* of HIV (often comprising up to 20% of a patient's viral population) which may be harbouring drug-resistant mutations. Thus clinicians must carefully study a patient's medical history for any additional clues that may suggest the presence or absence of mutations.

To address this task, we developed an novel "in-Silico" Sequencing System built on top of *ProLogICA* to assist in the interpretation of resistance tests and, more importantly, to infer likely mutations and drug resistances in the absence of such tests. These inferences are made using a patient's clinical history, which details previous treatment failures and successes, and a set of rules expressing which mutations confer resistance to which drugs. We do not invent these associations; instead we download the same rules used by the resistance testing laboratories to predict drug resistances from mutations determined by genetic sampling of clinical isolates from infected patients. But, instead of using them deductively (i.e. *forwards*) to predict likely drug resistances implied by observed mutations, we use the rules abductively (i.e. *backwards*) so as to explain observed drug resistances in terms of likely mutations.

The domain knowledge contains 64 rules (obtained from the French national AIDS research agency) for 16 FDA-approved drugs. These rules are represented using head atoms of the form $resistant(P, T, D)$ to denote that patient a P is resistant to a drug D at time T, and body atoms of the form $mutation(P, T, M)$ to denote that patient P is carrying the mutation M at time T. A typical rule is shown below for the drug zidovudine and states that the patient is resistant to zidovudine if he is carrying at least one of the three mutations $151M$, $69i$ or $215YF$. The notation $151M$ refers to a mutation at codon 151 in the viral genome whereby the wild type amino acid (according to a standard reference strain of HIV) is replaced by the mutant methionine M. Similarly, $69i$ denotes the *insertion* of an amino acid at position 69, and $215YF$ is a compact way of writing $215Y$ or $215F$.

These rules exploit the meta-predicate $atleast/2$ to ensure the satisfaction of a given number of goals from a given list using the fewest number of abducibles. The advantage of using this predicate is that, when given the goal $resistant(P, T, zidovudine)$ the system will not even consider showing the first two mutations if it has previously established $mutation(P, T, "215YF")$.

$$\begin{array}{l} resistant(P, T, zidovudine) \leftarrow \\ \quad atleast(1, [mutation(P, T, '151M'), \\ \quad\quad mutation(P, T, '69i'), mutation(P, T, '215YF')]). \end{array}$$

Because the treatment of HIV usually involves the prescription of powerful cocktails of at least three anti-retrovirals, we provide two clauses to link the possible ineffectiveness (resp. effectiveness) of a combination treatment with the resistance (resp. non-resistance) to one of the drugs D in the set of drugs S.

$$\begin{array}{l} ineffective(P, T, S) \leftarrow \\ \quad in(D, S),\ resistant(P, T, D). \end{array}$$

$$\begin{array}{l} effective(P, T, S) \leftarrow \\ \quad in(D, S), not\ resistant(P, T, D). \end{array}$$

Query	0	1	2	3	4	5	6	7	8	9	Avg.
KM	197,760	5,439	>85,211	56,916	435,600	9,828	28,231	263	263	15,022	>83,453
ProLogICA	5,492	1,208	56,165	4,062	3,570	1,979	2,395	74	74	1,436	7,646
+atleast	4,770	1,164	25,625	3,075	3,238	1,939	2,375	74	74	73	4,241
+minimal	1,283	51	333	138	1,628	191	1,938	73	73	72	578

Table 3: Number of hypotheses returned by KM vs. *ProLogICA* for 10 patients in the HIV resistance domain. — Experiments using the ALP model developed in (Ray *et al.* 2006).

To correctly model the dynamics of drug resistance, we need an integrity constraint that captures a key biological principle underlying this work: namely, the persistence of mutations. As formalised below, this constraint states that if a patient P is carrying a mutation M at time T, then he or she must also be carrying that mutation at all later times T'.

$$\bot \leftarrow mutation(P,T,M),\ T' \geq T,\ not\, mutation(P,T',M)$$

Declaring the predicate $mutation/3$ as abducible, this theory can explain a history of treatment successes and failures in terms of mutations the patient is (or is not) carrying. For example, one explanation of the observations below is that $p56$ was *not* carrying mutations $151M$, $69i$, $215YF$, $41L$, $67N$, $70R$ and $210W$ at time 1, but *was* carrying mutation $215YF$ at time 2.

$$effective(p56, 1, [zidovudine, lamivudine, indinavir]).$$

$$ineffective(p56, 2, [zidovudine, lamivudine, indinavir]).$$

An interesting feature of this approach is that, since we can never really be sure which mutations a patient is carrying, there is no point in looking for an optimal hypothesis. Thus, in contrast to previous applications of ALP, we accept that multiple explanations are inevitable and seek to develop ways of extracting useful information from them. For this purpose, we developed a domain specific system that uses *ProLogICA* to compute a set of explanations for a patient's clinical history and then analyses those explanations so as to predict which drugs the patient may be resistant to. This is done by ranking each drug by to the number of explanations that imply its resistance. As expected, a major computational obstacle is the sheer number of explanations.

A typical patient taking 3 different drugs at each of 10 time points results in more solutions (tens of thousands) than can be conveniently analysed. To efficiently reduce the number of redundant explanations, we applied *ProLogICA*'s minimisation routine incrementally after each time point. The reduction in the number of solutions for real clinical data from 10 HIV-infected patients is shown in Table 3. On average, *ProLogICA* provided an order of magnitude improvement of a standard implementation of KM (Kakas & Mancarella 1998). Moreover, the use of *ProLogICA*'s $atleast$ predicate resulted in another factor of 2 improvement, and the incremental application of minimality produced further order of magnitude improvement (and a significantly faster computation).

Conclusions, Related and Future Work

ProLogICA is a practical system for ALP whose utility has been shown by recent applications in the areas of multi-agent systems (Alberti *et al.* 2005) and bioinformatics (Ray *et al.* 2006; Papatheodorou, Kakas, & Sergot 2005). It is a lightweight system that is appropriate for many tasks which do not require the ability to solve complex optimisation problems, and is versatile enough to be used as a component of a larger reasoning system (Ray 2005).

In many respects, *ProLogICA* takes a step back from some recent ALP systems such as ACLP (Kakas, Michael, & Mourlas 2000), SLDNFAC (Denecker & Schreye 1998), A-system (Kakas, Van Nuffelen, & Denecker 2001), and CIFF (Endriss *et al.* 2004). In these systems, the emphasis is on integrating abduction with constraint solving to enable the solution of complex optimisation problems. These systems work by producing non-ground (existentially quantified) explanations with an associated set of constraints and then rely on the efficient satisfiability check of an external constraint solver. By contrast, *ProLogICA* exploits the simplicity of early procedures for ground abduction (Eshghi & Kowalski 1989; Kakas & Mancarella 1990a) by overcoming some of their well-known inefficiencies.

ProLogICA was motivated by recent attempts to apply ALP in areas of systems biology where it seems that highly specific explanations and models are required. Here, the task is to identify one or more hypotheses from a (large) set of ground hypotheses in order to explain some experimental observations. *ProLogICA* was originally developed as the abductive component of the HAIL learning system, which aims to support scientific theory development through the integration of abductive and inductive logic programming within a common reasoning framework. In this setting, ALP is only required to construct a specific account of the input observations to seed further generalisations. *ProLogICA* is well suited to both of these settings.

An alternative framework for performing ground abduction in Logic Programming is the framework of Answer Set Programming (ASP) (Lifschitz 1999). ASP systems such as SMODELS (Simons, Niemelä, & Soininen 2002) and DLV (Dell'Armi *et al.* 2001) have been continuously improving and offer nowadays a very efficient computational paradigm for many problem domains. In contrast to ALP approaches, which are top-down query-driven, ASP approaches are bottom-up data-driven. They operate by translating extended logic programs into ground propositional theories and employing satisfiability solvers to compute the models of the original program.

Although *ProLogICA* has proven to be an effective ALP system for the applications considered in this paper, a systematic comparison of ALP and ASP remains to be carried out. Using the translation introduced in (Satoh & Iwayama 1991) – whereby ALP problems can be encoded into the ASP formalism – would enable a comparison of the two approaches on the problems described above. Such a comparison may also lead to new approaches for combining different aspects of ALP and ASP technologies. For example, ALP could utilise ASP methods in its integrity constraint checking phase after it has generated a possible explanation.

Acknowledgments

We acknowledge the help of Andrea Bracialli who kindly provided the experimental data in Table 1. We are grateful to Irene Papatheodorou for her valuable comments regarding the gene interaction model.

References

Alberti, M.; Bracciali, A.; Chesani, F.; Ciampolini, A.; Endriss, U.; Gavanelli, M.; Guerri, A.; Kakas, A.; Lamma, E.; Lu, W.; Mancarella, P.; Mello, P.; Milano, M.; Riguzzi, F.; Sadri, F.; Stathis, K.; Terreni, G.; Toni, F.; Torroni, P.; and A.Yip. 2005. Experiments with animated societies of computees. Technical Report D14, SOCS Consortium. http://lia.deis.unibo.it/Research/Projects/SOCS/.

Dell'Armi, T.; Faber, W.; Ielpa, G.; Koch, C.; Leone, N.; Perri, S.; and Pfeifer, G. 2001. System description: Dlv. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, 424–428.

Denecker, M., and Schreye, D. D. 1998. SLDNFA: An Abductive Procedure for Abductive Logic Programs. *Journal of Logic Programming* 34(2):111–167.

Denecker, M.; Missiaen, L.; and Bruynooghe, M. 1992. Temporal reasoning with abductive event calculus. In *Proceedings of the 10th European conference on Artificial intelligence*, 384–388.

Endriss, U.; Mancarella, P.; Sadri, F.; Terreni, G.; and Toni, F. 2004. The CIFF Proof Procedure for Abductive Logic Programming with Constraints. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, 31–44. Springer Verlag.

Eshghi, K., and Kowalski, R. 1989. Abduction compared with negation by failure. In Levi, G., and Martelli, M., eds., *Proceedings of the 6th International Conference on Logic Programming*, 234–254. MIT Press.

Eshghi, K. 1988. Abductive planning with event calculus. In *Proceedings of the 5th International Conference and Symposium on Logic Programming*, 562–579.

Kakas, A., and Denecker, M. 2002. Abduction in Logic Programming. In Kakas, A., and Sadri, F., eds., *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *Lecture Notes in Computer Science*. Springer. 402–436.

Kakas, A., and Mancarella, P. 1990a. Database Updates through Abduction. In *Proceedings of the 16th International Conference on Very Large Databases*, 650–661.

Kakas, A., and Mancarella, P. 1990b. Generalized Stable Models: a Semantics for Abduction. In *Proceedings of the 9th European Conference on Artificial Intelligence*, 385–391. Pitman.

Kakas, A., and Mancarella, P. 1998. KM ALP procedure. At http://www.cs.ucy.ac.cy/aclp/alp_int.pl.

Kakas, A., and Michael, A. 2001. An abductive-based scheduler for air-crew assignment. *Applied Artificial Intelligence* 15(3):333–360.

Kakas, A., and Miller, R. 1997. A simple declarative language for describing narratives with actions. *Journal of Logic Programming* 31:157–200.

Kakas, A.; Kowalski, R.; and Toni, F. 1992. Abductive Logic Programming. *Journal of Logic and Computation* 2(6):719–770.

Kakas, A.; Michael, A.; and Mourlas, C. 2000. ACLP: Abductive constraint logic programming. *Journal of Logic Programming* 44(1-3):129–177.

Kakas, A.; Van Nuffelen, B.; and Denecker, M. 2001. A-system: Problem solving through abduction. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 591–596.

Lifschitz, V. 1999. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25 year perspective*, 357–373. Springer.

Papatheodorou, I.; Kakas, A.; and Sergot, M. 2005. Inference of gene relations from microarray data by abduction. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 3662 of *Lecture Notes in Computer Science*, 389–393.

Ray, O.; Antoniades, A.; Kakas, A.; and Demetriades, I. 2006. Abductive Logic Programming in the Clinical Management of HIV/AIDS. In *Proceedings of the 17th European Conference on Artificial Intelligence*. to appear.

Ray, O. 2005. *Hybrid Abductive-Inductive Learning*. Ph.D. Dissertation, Department of Computing, Imperial College London, UK.

Satoh, K., and Iwayama, N. 1991. Computing Abduction by Using the TMS. In *Proceedings of the 8th International Conference on Logic Programming*, 505–518. MIT Press.

Shanahan, M. 2000. An abductive event calculus planner. *Journal of Logic Programming* 44(1-3):207–240.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

Stathis, K.; Kakas, A.; Lu, W.; Demetriou, N.; Endriss, U.; and Bracciali, A. 2004. Prosocs: A platform for programming software agents in computational logic. In Müller, J., and Petta, P., eds., *Proceedings of the 4th International Symposium 'From Agent Theory to Agent Implementation'*.

Van Nuffelen, B., and Denecker, M. 2000. Problem solving in ID-logic with aggregates: some experiments. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Session on Abduction, 1–5.

4 Action and Change

The special track on reasoning about action and change (RAC) aims to bring together researchers to consider the fundamental issues in the field of RAC with their links to NMR, together with the new challenges that the deployment of RAC into applications can bring.

We had six submissions which we were all able to accept for presentation at the workshop. Five of the papers address foundational issues, from model checking in the situation calculus to revising action theories. One paper reports on applying the fluent calculus to solving a variant of the *Wumpus world.*

Session chairs

Antonis Kakas, University of Cyprus, Cyprus

Gerhard Lakemeyer, RWTH Aachen, Germany

Program committee

Thomas Eiter, TU Vienna, Austria

Alfredo Gabaldon, NICTA, Australia

Sebastian Sardina, RMIT, Australia

Marek Sergot, Imperial College, UK

Tran Cao Son, New Mexico State U, USA

Michael Thielscher, TU Dresden, Germany

Schedule Tuesday 30 May 2006

- 10.30 Yilan Gu and Iluju Kiringa, Model Checking Meets Theorem Proving: a Situation Calculus Based Approach
- 11.00 Jens ClaSSen and Gerhard Lakemeyer, A Semantics for ADL as Progression in the Situation Calculus
- 11.30 Thomas Eiter, Ersa Erdem, Michael Fink and Jan Senko, Resolving Conflicts in Action Descriptions
- 12.00 James P. Delgrande, Torsten Schaub and Hans Tompits An extended query language for action languages (and its application to aggregates and preferences)
- 12.00 Lunch
- 14.00 Debora Field and Allan Ramsay, Planning ramifications: When ramifications are the norm, not the ‘problem
- 14.30 Michael Thielscher, Designing a FLUX Agent for the Dynamic Wumpus World
- 15.00 S Grell, T Schaub and J Selbig, ModeLling Biological Networks by Actions Language Via Answer Set Programming 1 (Joint talk with “Systems and Applications”)
- 15.30 Coffee
- 16.00 Panel on Action Languages and Application Modeling

4.1 Model Checking Meets Theorem Proving

Model Checking Meets Theorem Proving: a Situation Calculus Based Approach

Yilan Gu
Dept. of Computer Science
University of Toronto
yilan@cs.toronto.edu

Iluju Kiringa
SITE
University of Ottawa
kiringa@site.uottawa.ca

Abstract

To reason about properties of reactive programs, one may usually follow either an operational or a deductive approach. In this paper, we propose representing the classical model checking approach of Clarke and Emerson in the situation calculus. Doing so, we propose an approach that merges the operational and the deductive approaches into one single framework by translating Kripke models that represent system specifications into theories formulated in the situation calculus and by recasting CTL as a sublanguage of the calculus.

Introduction

The importance of long running, nondeterministic concurrent programs has been emphasized over the past two and half decades since Pnueli proposed using temporal logic for reasoning about them (Pnueli 1977). These, also called reactive systems, as opposed to sequential transformational programs, show ideally nonterminating behaviors (Clarke, Grumberg, & Peled 1999). Their mathematical properties are usually defined using either the operational or the deductive approach. In the operational approach, programs are viewed as generator of computations. Given a program, all the computations associated with it can be generated once by an interpreter, or incrementally by specifying a transition relation that holds between consecutive states of the computation of the program. In summary, the operational semantics is based on the structure of the given program (Plotkin 1981). In the deductive approach, programs are viewed as specifying a set of computations about which some statements can be proven. Dynamic logic (Harel, Tiuryn, & Kozen 2000), Hoare's systems (Hoare 1969), and the situation calculus (McCarthy 1963; Reiter 2001) are examples of formalisms used in this approach.

The semantics of concurrent programs is described in either approaches in terms of infinite behaviors, also called computations. A behavior is a sequence of states that a program moves through while executing. The behaviors are all the possible interleavings of the "atomic" steps of the subprograms running in parallel; that is, given a concurrent program P composed of subprograms $P_1, P_2, \ldots, P_n$, where the P_is, $1 \leq i \leq n$, are sequential programs running in parallel, its execution is usually modeled by nondeterministically executing the atomic steps for each P_i, $1 \leq i \leq n$, in an arbitrary order. So if P is in a state s_k, it nondeterministically goes to the next state s_{k+1} by executing an arbitrary atomic step of any of its subprograms P_i. This procedure is repeated infinitely, or at least indefinitely.

Linear and branching temporal logics are the most commonly used languages for describing computations. The model checking problem (**MC**) can be defined as follows: Given (1) a reactive system $\mathfrak{S}$ represented as a finite-state structure which generates computations, and (2) a temporal logic formula $\mathfrak{P}$ specifying a property of $\mathfrak{S}$, find whether $\mathfrak{S}$ satisfies $\mathfrak{P}$. There are successful algorithmic solutions of **MC**. As an example, in (Clarke & Sistla 1986), Kripke Structures are used to represent the reactive system and Computational Tree Logic (CTL), a branching time temporal logic, is used to represent properties of the system.

In this paper, we propose representing the classical **MC** approach of (Clarke & Sistla 1986) in the situation calculus. Our approach merges both the operational and the deductive approaches into a single framework by translating Kripke models into theories of the situation calculus and by recasting CTL as a sublanguage of the same calculus. This approach can be labeled as deductive-operational in the sense of (Pnueli 1981); that is, it deals with computations arising during program execution and, at the same time, allows us to deductively reason about those computations in the logic of the situation calculus.

In (De Giacomo, Ternovskaia, & Reiter 1997), it is argued that, for non-terminating programs, one needs to rely on a transition semantics, in which one allows for interpreting and quantifying over parts of programs and their executions. In this paper, we show that an evaluation semantics in which one allows for interpreting whole programs is possible for non-terminating programs.

The paper is organized as follows. In the next section, we introduce the situation calculus, and the classical model of concurrent systems in terms of Kripke structures. Then we give an effective method for translating a Kripke structure to a basic action theory. Next, we

show how CTL properties are specified in the situation calculus and treat the model checking of action theories within our framework. This is followed by the presentation of an an example which illustrates our approach of model checking in the situation calculus. Finally, we conclude the paper and indicate avenues for future work.

Preliminaries

The Situation Calculus

The situation calculus (McCarthy 1963; Reiter 2001) is a many-sorted second order language with equality specifically designed for representing dynamically changing world. We consider a version of the situation calculus with three sorts for actions ($\mathcal{A}$), situations ($\mathcal{S}$), and objects ($\mathcal{O}$) other than the first two. **Actions** are first order terms consisting of a $0-ary$ action function symbols corresponding to the transitions of the finite state structures. **Situations** are first order terms denoting a sequence of actions. They are represented using a binary function symbol do: $do(\alpha, s)$ denotes the sequence resulting from adding the action α to the sequence s. The constant S_0 (*initial situation*) denotes the empty sequence [ ]. In modeling systems, situations will correspond to computations. **Objects** constitute a catch-all sort representing everything else depending on the domain of application.

We shall have a finite number of unary predicates called **fluents** which represent properties with truth values varying from state to state. Fluents are denoted by predicate symbols with argument a situation term. In a Reader-Writer example given below (Example 1), $state_1(s)$ is a relational fluent, meaning that the system is in state w_1 after performing the sequence of operations in the computation s. In addition to the fluents, we shall have a finite number of ground situation independent predicates. For example, we will use ground binary predicate $trans(I, J)$ to represent a transition form state w_I to w_J of the system being modeled.

The language also includes special predicates $Poss$, and $\sqsubset$; $Poss(a, s)$ means that the action a is possible in the situation s, and $s \sqsubset s'$ states that the situation s' is reachable from s by performing some sequence of actions. In system modeling terms, $s \sqsubset s'$ means that s is a proper subcomputation of the computation s'. The predicate $\sqsubset$ will be useful in formulating properties of systems. We call this fragment of the situation calculus ${\mathcal{L}_0}^0$. In general, we can define fragments ${\mathcal{L}_j}^i$, where i (j) is the maximum number of arguments of sort $\mathcal{O}$ that an action function (fluent predicate) may have.

Axiomatizing a Domain Theory

A domain theory is axiomatized in the situation calculus with four classes of axioms which constitute a *basic action theory* (BAT – More details in (Pirri & Reiter 1999)):[1]

[1]There are also **unique names axioms** which guarantee that primitive actions of the domain are pairwise unequal.

Foundational axioms for situations ($\mathcal{D}_f$). These guarantee an infinite tree structure for the situations, and are the same for all BATs.

Action precondition axioms ($\mathcal{D}_{ap}$). There is one for each action function A, with syntactic form $Poss(A, s) \equiv \Pi_A(s)$. Here, $\Pi_A(s)$ is a formula with free variable s. These characterize the preconditions for doing action A in the situation s.

In the Reader-Writer example, the following states that it is possible for the system to move from state 1 to state 2 relative to the system computation s iff there is a transition from state 1 to state 2, and as a result of performing the actions in that computation, the system is in state 1.

$$Poss(tr_{1,2}, s) \equiv trans(1, 2) \wedge state_1(s).$$

Successor state axioms ($\mathcal{D}_{ss}$). There is one for each relational fluent $F(s)$, with syntactic form $F(do(a, s)) \equiv \Phi_F(a, s)$, where $\Phi_F(a, s)$ is a formula with free variables among a and s. These characterize the truth values of the fluent F in the next situation $do(a, s)$ in terms of the current situation s, and they embody a solution to the frame problem for deterministic actions (Reiter 2001).

In the Reader-Writer example, the following states that the system will be in the state 1 relative to the computation $do(a, s)$ iff the last system operation a in the computation was $tr_{4,1}$ or $tr_{6,1}$, or it was already in state 1 relative to the computation s, and a does not lead the system to another state.

$$state_1(do(a, s)) \equiv a = tr_{4,1} \vee a = tr_{6,1} \vee$$
$$state_1(s) \wedge a \neq tr_{1,2} \wedge a \neq tr_{1,3}.$$

Initial database ($\mathcal{D}_{S_0}$). This is a set of first order sentences whose only situation term is S_0; it specifies the initial state of the domain, in our case, the initial system state. Notice that while these initial system axioms specify a complete initial system state (as is normal for reactive systems), this is not a requirement of the theory we are presenting. Therefore our account could, for example, accommodate incomplete initial system states.

Notations

We now introduce further notations used later in the paper. Suppose $\mathcal{D}$ is a basic action theory. Furthermore, suppose that $\mathcal{U}_\mathcal{A} = \{A_1, \ldots, A_k\}$ is the (finite) set of actions of $\mathcal{D}$, ${\mathcal{U}_\mathcal{A}}^*$ is the set of action sequences, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ is the (finite) set of fluents of $\mathcal{D}$, and $V = \{0, 1\}$ is a set of labels denoting the truth values. Then $\vec{v}_0 = \langle v_{0_1}, \ldots, v_{0_n} \rangle$ denotes the vector of initial truth values of fluents of $\mathcal{D}$, where v_{0_j} with $1 \leq j \leq n$ is the initial value of fluent F_j, and $c_i(\alpha)$ specifies whether the fluent F_i holds in the situation represented by the action sequence α; c_i is called the *characteristic function* (Ternovskaia 1999) of fluent F_i.

Checking a Situation Calculus System

Let $\mathcal{D}$ be a background situation calculus axiomatization for some reactive system, as described above, and

let $Q(s)$ be a situation calculus formula – a property – with one free situation variable s.

Let $S = do(\alpha_n, do(\alpha_{n-1}, \cdots, do(\alpha_1, S_0) \cdots))$ be a situation term that mentions no free variables. We treat this as a system computation, and define the *answer* to Q *relative to this computation* to be "yes" iff $\mathcal{D} \models Q(S)$. The answer is "no" iff $\mathcal{D} \models \neg Q(S)$. So on this definition, model checking is performed relative to a system computation, and, in the most general setting, it is a theorem-proving task. In particular, the executability problem is to check whether $\mathcal{D} \models executable(S)$, where

$$executable(s) =_{df} (\forall a, s').do(a, s') \sqsubseteq s' \supset Poss(a, s').$$

It is important to notice the following property of basic action theories formulated in the language of Subsection "The Situation Calculus".

Theorem 1 *The basic action theories formulated in the fragment ${\mathcal{L}_0}^0$ of the situation calculus are decidable. That is, suppose $\mathcal{D}$ is a BAT, and ϕ is a formula, all of which are formulated in ${\mathcal{L}_0}^0$; then there is an algorithm for establishing whether $\mathcal{D} \models \phi$.*

Proof (outline):
Similar to the idea in This is a corollary of Theorem 3 in (Ternovskaia 1999) that shows decidability for a similar fragment of the situation calculus, but where there are no situation independent predicates. The proof there is now augmented by showing that adding finitely many ground situation independent predicates does not change the nature of the automata constructed for $\mathcal{D}$ and ϕ. □

GOLOG

GOLOG (Levesque *et al.* 1997) is a situation calculus-based programming language for defining complex actions in terms of a set of primitive actions axiomatized in the situation calculus according to Subsection "Axiomatizing a Domain Theory". It has control structures found in most Algol-like languages, augmented by some nonstandard structures: *Sequence* (α ; β: Do action α, followed by action β); *Test actions* (p?: Test the truth value of expression p in the current situation); *Nondeterministic action choice* ($\alpha \mid \beta$: Do α or β); *Nondeterministic choice of arguments* ($(\pi\ x)\alpha$: Nondeterministically pick a value for x, and for that value of x, do action α); Conditionals (*if-then-else*) and *while* loops; and *Procedures, including recursion.*

The following is a GOLOG procedure that executes an sequence of n randomly picked actions which are possible:

proc $execActions(n)$
$n = 0? \mid$
$n > 0?\ ;\ (\pi\ a)[Poss(a)?\ ;\ a]; execActions(n-1)$
endProc .

The semantics of GOLOG programs is defined by macro-expansion, using a ternary relation Do (Levesque *et al.* 1997); $Do(\mathtt{P}, s, s')$ is an *abbreviation* for a situation calculus formula which intuitively means that s' is one of the situations reached by evaluating the GOLOG program P, beginning in situation s. In the reactive system setting, any binding for s' represents the system computation that results from executing P, beginning in the system state defined by the computation s.

Concurrent Systems

Concurrent reactive systems are semantically characterized by transition systems (Wolper 1998; Clarke, Grumberg, & Peled 1999). Usually, the latter are modeled by *Kripke structures* (Clarke, Grumberg, & Peled 1999) which we now introduce.

Definition 1 (Kripke structure) *A finite Kripke structure is a quintuple $\mathbf{K} = (P, W, R, w_0, L)$ where*

- *P is a finite set of atomic propositions;*
- *W is a finite set of states;*
- *$R \subseteq W \times W$ is a total (transition) relation;*
- *w_0 is an initial state;*
- *$L : W \rightarrow 2^P$ maps each $w \in W$ to the set $\{p \in P \mid \models_w p\}$.*

Definition 2 (Behavior) *Suppose $\mathbf{K} = (P, W, R, w_0, L)$ is a Kripke structure. Then a behavior σ of $\mathbf{K}$ is a function from N, a subset of the natural numbers, to W such that:*

- *$N = \{0, 1, .., n\}$ for some natural number $n \in \mathbb{N}$, or N is the set of natural numbers;*
- *$\sigma(0) = w_0$;*
- *$\forall i \geq 0\ \ (\sigma(i), \sigma(i+1)) \in R$.*

If N equals the set $\mathbb{N}$ of natural numbers, then σ is called an infinite behavior.

Example 1 *Consider a concurrent system – denoted by RW – consisting of a Reader process, numbered 1, and a Writer process, numbered 2 (Emerson & Trefler 1999). We define RW as follows. Each of these processes can be in three states: Non-Trying, Trying, and Critical Section. These are thus subscripted accordingly: N_i, T_i, and C_i refer the Non-Trying, Trying, and Critical Section states of Process i. Process 1 may enter its critical section only when Process 2 is in its Non-trying section, and Process 2 may enter its critical section only when Process 1 is in its Non-Trying or Trying states. Figure 1 is a Kripke structure representing all the reachable states of the RW system. Formally, we have the following:*
$P = \{N_1, N_2, T_1, T_2, C_1, C_2\}$; $W = \{w_0, \cdots, w_7\}$;
$R = \{(w_0, w_1), (w_0, w_2), (w_1, w_3), (w_1, w_4), (w_2, w_4), (w_2, w_5)$
$(w_3,w_0),(w_3,w_6),(w_4,w_7),(w_5,w_0),(w_5,w_7),(w_6,w_2),(w_7,w_1)\}$
the initial state w_0; and $L(w_0) = \{N_1, N_2\}, L(w_1) = \{T_1, N_2\}, L(w_2) = \{N_1, T_2\}, L(w_3) = \{C_1, N_2\}, L(w_4) = \{T_1, T_2\}, L(w_5) = \{N_1, C_2\}, L(w_6) = \{C_1, T_2\}, L(w_7) = \{T_1, C_2\}$.

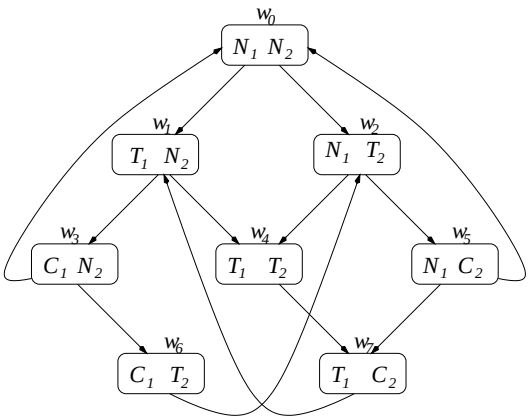


Figure 1: Reader-Writer transition system

One may unwind a Kripke structure into an infinite tree that is rooted in w_0. Such trees are called *computational trees*.

Definition 3 (Computational Tree) *Suppose* $\mathbf{K} = (P, W, R, w_0, L)$ *is a Kripke structure. Then the (infinite) computational tree* $CT_{\mathbf{K}}$ *of* $\mathbf{K}$ *is the set* $\{\sigma_1, \sigma_2, \ldots\}$ *of all (infinite) behaviors of* $\mathbf{K}$*; that is,*

- *for each* $i = 1, 2, \ldots$, σ_i *is a function from* $\mathbb{N}$ *to* W*;*
- $\sigma_i(0) = w_0$ *for all* $i = 1, 2, \ldots$*;*
- $\forall j > 0, i \geq 0 \;\; (\sigma_j(i), \sigma_j(i+1)) \in R$.

The top tree in Figure 2 shows the infinite computational tree of the RW system depicted in Figure 1. Thus any path starting in the root of the computational tree represents a behavior of the Kripke structure.

Translating Concurrent Systems into BATs

Recall that the foundational axioms guarantee the infinite tree structure of the situation calculus. We shall translate Kripke structures to the situation calculus by relating the idea of computational tree to the situation calculus tree of situations. More precisely, the Kripke structure will be translated into a BAT such that the computational tree of the Kripke structure is represented as a subtree obtained from the tree of situations by pruning away paths that are not executable.

In order to relate Kripke structures to BATs, we need to define a situation tree like model for a BAT $\mathcal{D}$; that model is precisely a tree of situations constrained appropriately using the successor state axioms and action precondition axioms of $\mathcal{D}$.

Definition 4 (Canonical Structure)[2] *Suppose* $\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0}$ *is a BAT. Then a structure* $\mathfrak{M}$ *is a canonical structure for* $\mathcal{D}$ *iff it is a pair* (D, L_D)*, where*

- $D \subseteq {\mathcal{U}_{\mathcal{A}}}^*$ *is the domain of* $\mathfrak{M}$*, satisfying the following property: if an action sequence* α *is in* D *then any prefix* α' *of* α *(i.e.,* $\alpha = \alpha'\alpha''$ *for some* $\alpha'' \in {\mathcal{U}_{\mathcal{A}}}^*$*) is in* D*;*
- L_D *is a labeling function* $D \to V^n$ *such that* $L_D([\,]) = \vec{v}_0$ *and* $L_D(\alpha) = \langle c_1(\alpha), \ldots, c_n(\alpha)\rangle \in V^n$.

  Here, $\vec{v}_0$ *is the initial vector of fluent values, and* c_i *is the characteristic function of fluent* F_i.

Notice that in the definition above, the pair (D, L_D) is in fact a situation tree constrained using the BAT $\mathcal{D}$. The domain D is the set of nodes of the tree; and for any sequence α and $1 \leq j \leq k$, αA_j is the j-th son of the node α labeled by value vector $L_D(\alpha)$. Figure 2 shows a canonical structure for the RW system depicted in Figure 1. The labeling $tr_{i,j}$ of the edges denotes an action corresponding to the transition $(S_i, S_j) \in R$. Details of the BAT underlying this tree will be clearer in Section "An Example". It suffices here to mention that $\langle N_1, N_2, T_1, T_2, C_1, C_2\rangle$ is the vector of fluents describing the properties of the system. The vector $\vec{v}_0$ of initial values for the root of the situation tree is $\langle 1, 1, 0, 0, 0, 0\rangle$, and the vectors labeling the other nodes of the tree depend on the characteristic functions of each fluent.

Now, we show how to effectively construct a basic action theory from a given Kripke structure.

Theorem 2 *Suppose* $\mathbf{K} = (P, W, R, w_0, L)$ *is a Kripke structure. Then one can effectively construct a BAT* $\mathcal{D}_{\mathbf{K}}$ *whose canonical structure* $\mathfrak{M}$ *is obtained from the computational tree* $CT_{\mathbf{K}}$ *of* $\mathbf{K}$ *such that*

$$\mathbf{K} \;\; \textit{has} \;\; CT_{\mathbf{K}} \;\; \textit{iff} \;\; \models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}.$$

Proof:
Let $\mathcal{D}_{\mathbf{K}} = \mathcal{D}_f \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{S_0}$ be the following BAT.

Fluents: for each $p \in P$, introduce a fluent $p(s)$; for each state $w_i \in W$, introduce a fluent $state_i(s)$.

Actions: for each transition $(w_i, w_j) \in R$ where $w_i, w_j \in W$, introduce an action $tr_{i,j}$.

Initial database ($\mathcal{D}_{S_0}$): Whenever $p \in L(w_0)$, introduce the axiom $p(S_0)$, otherwise introduce $\neg p(S_0)$; introduce axiom $state_0(S_0)$ and, for all $w_i \neq w_0$, introduce axioms $\neg state_i(S_0)$. We also need to introduce finitely many non-fluent predicates $tr(i, j)$ where $0 \leq i, j < |W|$, such that $trans(i, j)$ is true if and only if $(w_i, w_j) \in R$.

Action precondition axioms ($\mathcal{D}_{ap}$): for each transition $tr_{i,j}$, we have the axiom

$$Poss(tr_{i,j}, s) \equiv trans(i, j) \wedge state_i(s).$$

Successor state axioms ($\mathcal{D}_{ss}$): For every i, $0 \leq i < |W|$,

[2] This definition is similar to the notion of k-ary n-labeled situation tree associated with a BAT defined in (Ternovskaia 1999).

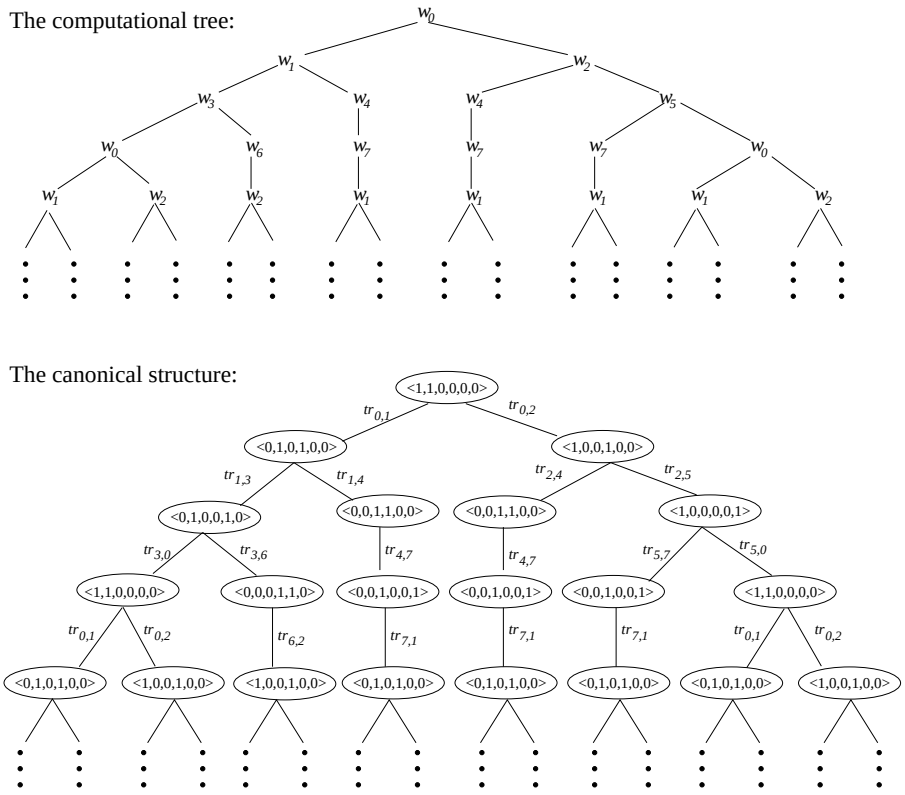


Figure 2: Computation tree and canonical structure of the RW system

$$state_i(do(a,s)) \equiv \bigvee_{j=1}^{|W|} a = tr_{j,i} \vee state_i(s) \wedge \bigwedge_{j=1}^{|W|} a \neq tr_{i,j}.$$

For every $p \in P$, suppose $W_p = \{w_i | \models_{w_i} p\}$. Then,

$$p(s) \equiv \bigvee_{w_i \in W_p} state_i(s).$$

Hence we could easily get the successor state axiom for $p(s)$.

Now we show that $\mathbf{K}$ has $CT_{\mathbf{K}}$ iff $\models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}$. Suppose $\mathbf{K}$ has $CT_{\mathbf{K}}$. Then the proof proceeds by constructing the canonical structure corresponding to $CT_{\mathbf{K}}$. It is easy to show that the BAT constructed above is satisfiable in this canonical structure.

Suppose $\models_{\mathfrak{M}} \mathcal{D}_{\mathbf{K}}$ and $D \subseteq \mathcal{U}_{\mathcal{A}}{}^*$. Then $L_D([\,]) = \vec{v}_0$ specifies the root of $CT_{\mathbf{K}}$ by telling exactly which fluent of the form $state_0$ is true and which other fluents are true in the state w_0 of the Kripke structure $\mathbf{K}$. Furthermore, $L_D(\omega A_j) = \langle c_1(w), \ldots, c_n(w) \rangle \in V^n$ determines the fluent values in situation ωA_j by telling which fluent of the form $state_i$ is true and which other fluents are true in the state w of $\mathbf{K}$ corresponding to the execution of the transitions encoded in ωA_j. Thus each path in $\mathfrak{M}$ starting in $[\,]$ yields a corresponding path in $CT_{\mathbf{K}}$. □

Model Checking

CTL

The temporal logics that are used for specifying properties in **MC** are subsets of the logic CTL* (Clarke & Sistla 1986) which expresses a branching time logic by extending linear time temporal logic with behavior quantifiers. The logic CTL is the smallest set of formulas inductively defined as follows.

Situation formulas

– `true` and `false` are atomic situation formulas, as well as are p and $\neg p$ for all $p \in P$.

– If ϕ and ψ are situation formulas, then $\phi \wedge \psi$ and $\phi \vee \psi$ are situation formulas.

– If ϕ is a behavior formula, then $A\phi$ ("ϕ holds for all behaviors") and $E\phi$ ("ϕ holds for some behavior") are situation formulas.

Behavior formulas

– If ϕ and ψ are situation formulas, then $X\phi$ ("next time ϕ"), and $\phi U \psi$ ("ϕ until ψ") are behavior formulas.

Moreover, $F\phi$ ("ϕ holds at some future state on a behavior") and $G\phi$ ("ϕ holds at all future states on a behavior") for behavior formula ϕ abbreviate `true`$U\phi$ and $\neg F \neg \phi$ respectively.

Semantics

Here, we semantically characterize CTL formulas by translating them to formulas of the decidable fragment of the situation calculus described in Subsection "The Situation Calculus". CTL formulas are interpreted over Kripke structures. We shall denote the suffix of the behavior $\sigma = S_0, s_1, s_2, \ldots$ that starts at situation s_j by σ^j. Given a Kripke structure $\mathbf{K} = (P, W, R, w_0, L)$, by Theorem 2 we first get a BAT $\mathcal{D}_\mathbf{K}$ corresponding to $\mathbf{K}$. We then introduce the notation $\phi[s]$ to denote the situation calculus formula obtained from a given expression ϕ by restoring the situation argument s in all the fluents occurring in ϕ. Finally, we view a CTL formula $(Op\ \phi)[s]$ as a macro defined in the situation calculus as follows:

$$p[s] =_{df} \bigvee_{w_i \in W_p} state_i(s), \text{ where } p \text{ is an atomic proposition and } W_p = \{w | \models_w p\},$$
$$(\neg\phi)[s] =_{df} \neg\, \phi[s]$$
$$(\phi_1 \wedge \phi_2)[s] =_{df} \phi_1[s] \wedge \phi_2[s],$$
$$EX\phi[s] =_{df} (\exists a).Poss(a, s) \wedge \phi[do(a, s)],$$
$$A(\psi_1 U \psi_2)[s] =_{df} (\forall s').succ^*(s, s') \wedge \psi_2[s'] \supset (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset \psi_1[s''],$$
$$E(\psi_1 U \psi_2)[s] =_{df} (\exists s').succ^*(s, s') \wedge \psi_2[s'] \wedge (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset \psi_1[s''].$$

Here, $succ^*(s, s')$ is defined as follows:

$$succ^*(s, s') =_{df} s \sqsubseteq s' \wedge executable(s'),$$

meaning that s' is a subsequent situation of s and s' is executable. Further operators are defined in terms of those above:

$$(\phi_1 \vee \phi_2)[s] =_{df} \neg(\neg\phi_1 \wedge \neg\phi_2)[s],$$
$$(\phi_1 \supset \phi_2)[s] =_{df} (\neg\phi_1 \vee \phi_2)[s],$$
$$AX\phi[s] =_{df} (\neg EX\neg\phi)[s],$$
$$EF\phi[s] =_{df} E(\mathtt{true} U \phi)[s],$$
$$AF\phi[s] =_{df} A(\mathtt{true} U \phi)[s],$$
$$EG\phi[s] =_{df} (\neg AF\neg\phi)[s],$$
$$AG\phi[s] =_{df} (\neg EF\neg\phi)[s].$$

Usually, the semantics of CTL is given in terms of situations (states) and behavior (paths) formulas (Clarke & Sistla 1986). We can introduce this distinction here by viewing the situations in the situation formulas as "snapshots" of the world and those in behavior formulas as "histories". For later convenience, given CTL formula ϕ we will always denote the corresponding semantic formula of ϕ at any situation s as $Q_\phi(s)$.

Checking Properties

Above, we have defined the semantics of CTL formulas in terms of a translation of these formulas into formulas of a decidable fragment of the situation calculus. Now, we define the model checking task in terms of a logical entailment. Given a Kripke structure $\mathbf{K} = (P, W, R, w_0, L)$ and a CTL formula ϕ, we first construct a BAT $\mathcal{D}_\mathbf{K}$ using the algorithm in the proof of Theorem 2. We then construct a situation calculus formula $Q_\phi(s)$ corresponding to ϕ using the method described in Subsection "Semantics". All these constructions are done in polynomial time and yield axioms and formulas that are polynomial in the size of both $\mathbf{K}$ and ϕ. Now model checking the system $\mathbf{K}$ against the property ϕ for initial state w_0 amounts to establishing the entailment

$$\mathcal{D}_\mathbf{K} \models Q_\phi(S_0).$$

In (De Giacomo, Ternovskaia, & Reiter 1997), dynamic properties of reactive systems are expressed by using the transition semantics and second order formulas expressing least and greatest fix-point properties. Here, following (Clarke, Grumberg, & Peled 1999), we specify properties by using transition relation R of the Kripke structure representing a reactive system and second order formulas expressing least and greatest fix-point properties. We use the following theorem from (Clarke, Grumberg, & Peled 1999) reformulated in the situation calculus to that end:

Theorem 3 *Suppose that K is a Kripke structure and that we identify each CTL formula ϕ with the set $\{s \mid K, s \models \phi\} \subseteq 2^{S_K}$. Then each of the basic CTL operators may be characterized as a least or greatest fix-point of an appropriate predicate transformer in the following way:*

$$EF\phi[s] \equiv \mu_Z.[\phi[s] \vee EX(Z)[s]],$$
$$AF\phi[s] \equiv \mu_Z.[\phi[s] \vee AX(Z)[s]],$$
$$EG\phi[s] \equiv \nu_Z.[\phi[s] \wedge EX(Z)[s]],$$
$$AG\phi[s] \equiv \nu_Z.[\phi[s] \wedge AX(Z)[s]],$$
$$A(\phi_1 U \phi_2)[s] \equiv \mu_Z.[\phi_2[s] \vee \phi_1[s] \wedge AX(Z)[s]],$$
$$E(\phi_1 U \phi_2)[s] \equiv \mu_Z.[\phi_2[s] \vee \phi_1[s] \wedge EX(Z)[s]].$$

An Example

Now, we effectively construct a BAT from the Kripke structure of Figure 1 representing the RW system.

Actions: $tr_{0,1}, tr_{0,2}, tr_{1,3}, tr_{1,4}, tr_{2,4}, tr_{2,5}, tr_{3,0}, tr_{3,6}, tr_{4,7}, tr_{5,0}, tr_{6,2}, tr_{7,1}$.

Fluents: $T_1(s), T_2(s), N_1(s), N_2(s), C_1(s), C_2(s), state_0(s), state_1(s), state_2(s), state_3(s), state_4(s), state_5(s), state_6(s), state_7(s)$.

Initial database:

$$N_1(S_0) \wedge N_2(S_0) \wedge \neg T_1(S_0) \wedge \neg T_2(S_0) \wedge \neg C_1(S_0) \wedge \neg C_2(S_0) \wedge trans(0,1) \wedge trans(0,2) \wedge trans(1,3) \wedge trans(1,4) \wedge trans(2,4) \wedge trans(2,5) \wedge trans(3,0) \wedge trans(3,6) \wedge trans(4,7) \wedge trans(5,0) \wedge trans(5,7) \wedge trans(6,2) \wedge trans(7,1) \wedge state_0(S_0) \wedge \neg state_1(S_0) \wedge \neg state_2(S_0) \wedge \neg state_3(S_0) \wedge \neg state_4(S_0) \wedge \neg state_5(S_0) \wedge \neg state_6(S_0) \wedge \neg state_7(S_0).$$

Action precondition axioms:

$$Poss(tr_{0,1}, s) \equiv trans(0,1) \wedge state_0(s),$$
$$Poss(tr_{0,2}, s) \equiv trans(0,2) \wedge state_0(s),$$
$$Poss(tr_{1,3}, s) \equiv trans(1,3) \wedge state_1(s),$$
$$Poss(tr_{1,4}, s) \equiv trans(1,4) \wedge state_1(s),$$
$$Poss(tr_{2,4}, s) \equiv trans(2,4) \wedge state_2(s),$$
$$Poss(tr_{2,5}, s) \equiv trans(2,5) \wedge state_2(s),$$
$$Poss(tr_{3,0}, s) \equiv trans(3,0) \wedge state_3(s),$$
$$Poss(tr_{3,6}, s) \equiv trans(3,6) \wedge state_3(s),$$
$$Poss(tr_{4,7}, s) \equiv trans(4,7) \wedge state_4(s),$$
$$Poss(tr_{5,0}, s) \equiv trans(5,0) \wedge state_5(s),$$
$$Poss(tr_{6,2}, s) \equiv trans(6,2) \wedge state_6(s),$$
$$Poss(tr_{7,1}, s) \equiv trans(7,1) \wedge state_7(s).$$

Successor state axioms:

$$state_0(do(a,s)) \equiv a = tr_{3,0} \vee a = tr_{5,0} \vee state_0(s) \wedge a \neq tr_{0,1} \wedge a \neq tr_{0,2},$$
$$state_1(do(a,s)) \equiv a = tr_{0,1} \vee a = tr_{7,1} \vee state_1(s) \wedge a \neq tr_{1,3} \wedge a \neq tr_{1,4},$$
$$state_2(do(a,s)) \equiv a = tr_{0,2} \vee a = tr_{6,2} \vee state_2(s) \wedge a \neq tr_{2,4} \wedge a \neq tr_{2,5},$$
$$state_3(do(a,s)) \equiv a = tr_{1,3} \vee state_3(s) \wedge a \neq tr_{3,0} \wedge a \neq tr_{3,6},$$
$$state_4(do(a,s)) \equiv a = tr_{1,4} \vee a = tr_{2,4} \vee state_4(s) \wedge a \neq tr_{4,7},$$
$$state_5(do(a,s)) \equiv a = tr_{2,5} \vee state_5(s) \wedge a \neq tr_{5,0} \wedge a \neq tr_{5,7},$$
$$state_6(do(a,s)) \equiv a = tr_{3,6} \vee state_6(s) \wedge a \neq tr_{6,2},$$
$$state_7(do(a,s)) \equiv a = tr_{4,7} \wedge a = tr_{5,7} \vee state_7(s) \wedge a \neq tr_{7,1}.$$

Abbreviations:

$$T_1(s) \equiv state_1(s) \vee state_4(s) \vee state_7(s),$$
$$T_2(s) \equiv state_2(s) \vee state_4(s) \vee state_6(s),$$
$$N_1(s) \equiv state_0(s) \vee state_2(s) \vee state_5(s),$$
$$N_2(s) \equiv state_0(s) \vee state_1(s) \vee state_3(s),$$
$$C_1(s) \equiv state_3(s) \vee state_6(s),$$
$$C_2(s) \equiv state_5(s) \vee state_7(s).$$

Simulation

To generate finite sequences of actions of the given concurrent system $\mathcal{D}_{\mathbf{K}}$ and check if property $Q_\phi(S_0)$ is satisfied , we solve the following deduction task:

$$\mathcal{D}_{\mathbf{K}} \models (\exists s).Do(execActions(N), S_0, s) \wedge Q_\phi(S_0),$$

where N is a constant natural number and $execActions(N)$ is the GOLOG procedure defined in the section on GOLOG.

To generate non-terminating sequences of actions and check if $\mathcal{D}_{\mathbf{K}} \models Q_\phi(S_0)$, we solve the following deduction task:

$$\mathcal{D}_{\mathbf{K}} \models checkCTL(\phi),$$

where abbreviation $checkCTL(\phi)$ represents a sentence obtained by replacing all the predicate $succ^*(s, s')$ in $Q_\phi(S_0)$ by

$$\exists \delta.Trans^*(execActions, s, \delta, s'),$$

and $execActions$ is the following GOLOG procedure that infinitely generates transitions of the system at random.

proc *execActions*
  while $true$ $(\pi\, a)[Poss(a)?;\ a]$; **endWhile**
endProc .

$Trans^*(execActions, s, \delta, s')$ represents the execution of (non-terminating) GOLOG program $execActions$ starting from situation s and getting to situation s' with program δ remained. The detailed semantics of $Trans^*$ is given in (De Giacomo, Ternovskaia, & Reiter 1997). For example, to check CTL formula whether $A(\psi_1 U \psi_2)$ holds for concurrent system $\mathbf{K}$, we are to solve whether

$$\mathcal{D}_{\mathbf{K}} \models (\forall s).(\exists \delta).Trans^*(execActions, S_0, \delta, s) \wedge \psi_2[s] \supset (\forall s').(s' \sqsubset s \supset \psi_1[s']).$$

Sample Properties

Some simple properties of the RW system expressed in CTL are: $EG(N_2 \supset EX\ N_2)$, $AG(N_2 \supset EF\ C_2)$, $EG(\neg C_1 \wedge \neg C_2)$, $EF(C_1 \wedge C_2)$, etc. By restoring the situation argument s, these properties can be viewed as macros defined in the situation calculus, which still match the intuitive semantics of the CTL formulas. For instance,

$$\begin{aligned}
&(EG(N_2 \supset EX\ N_2))[s] \\
&\quad \equiv (\neg AF(N_2 \wedge \neg EX\ N_2))[s] \\
&\quad \equiv \neg A(\mathtt{true} U (N_2 \wedge \neg EX\ N_2))[s] \\
&\quad \equiv \neg(\forall s').succ^*(s, s') \wedge (N_2 \wedge \neg EX\ N_2)[s'] \\
&\qquad\qquad \supset (\forall s'').s \sqsubseteq s'' \sqsubset s' \supset true[s''] \\
&\quad \equiv (\exists s').succ^*(s, s') \wedge N_2(s') \supset (EX\ N_2)[s] \\
&\quad \equiv (\exists s').succ^*(s, s') \wedge N_2(s') \supset \\
&\qquad\qquad (\exists s'').succ(s', s'') \wedge N_2(s'').
\end{aligned}$$

$$\begin{aligned}
&(AG(N_2 \supset EF\ C_2))[s] \\
&\quad \equiv (\neg EF(N_2 \wedge \neg EF\ C_2))[s] \\
&\quad \equiv (\neg E(trueU(N_2 \wedge \neg EF\ C_2))[s] \\
&\quad \equiv \neg(\exists s').succ^*(s, s') \wedge N_2(s') \wedge (\neg EF\ C_2)[s'] \\
&\quad \equiv (\forall s').succ^*(s, s') \wedge N_2(s') \supset (E(\mathtt{true} U C_2))[s'] \\
&\quad \equiv (\forall s').succ^*(s, s') \wedge N_2(s') \supset \\
&\qquad\qquad (\exists s'').succ^*(s', s'') \wedge C_2(s'').
\end{aligned}$$

Discussion

Ours can be considered as a symbolic model checking approach without BDDs, similar to the approach described in (Biere *et al.* 1999). In fact we show how to reduce model checking to entailment in a decidable subset of the situation calculus. An early work heading in this direction is reported in (Rajan, Shankar, & Srivas 1995); unfortunately, lack of technical detail does not allow a comparison with our approach.

Notice that (Reiter 2001) gives an implementation technique for BATs such as the one of Section "An Example". This technique justifies a straightforward translation of the BATs to a form suitable for a Prolog implementation. This technique could be applied here. This would amount to implementing a predicate, e.g. $checkCTL(\phi)$, where ϕ is a CTL formula, for checking whether the BAT entails $Q_\phi(S_0)$. The same technique could be used to simulate the system modeled by the BAT. Since CTL properties involve the predicate $\sqsubset$, any interpreter for checking these properties will necessarily be non-Markovian (Gabaldon 2002) meaning that effects of actions are explained by taking into account all past situations. All this however remains to be accounted for.

A perceived advantage of our framework is the richness of the situation calculus which is more expressive than branching time temporal logic (Pinto 1994). A systematic study of fragments richer than the one considered in this paper remains to be undertaken. It remains also to see how our framework can be turned into a practical tool using well-known automata theoretic semantics for the situation calculus in the style of (Vardi & Wolper 1986).

The fragment ${\mathcal{L}_0}^0$ is powerful enough to express relatively realistic systems. In general however, we can define fragments ${\mathcal{L}_j}^i$, with increasing indexes i and j. What is the exact expressive power of ${\mathcal{L}_0}^0$? What do we gain in expressive power with the fragments ${\mathcal{L}_j}^i,\ i = 1, 2, \cdots,\ j = 1, 2, \cdots$? All these questions are worth pursuing.

References

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without bdds. In *Proceedings of TACAS/ETAPS'99*, 193–207. Berlin: Springer Verlag.

Clarke, E.M. Emerson, E., and Sistla, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8:244–263.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking*. Cambridge, MA: MIT Press.

De Giacomo, G.; Ternovskaia, E.; and Reiter, R. 1997. Non-terminating processes in the situation calculus. *AAAI'97 Workshop*.

Emerson, E., and Trefler, R. 1999. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In Pierre, L., and Kropf, T., eds., *Correct Hardware Design and Verification Methods. Proceedings of the 10th IFIP WG*, 142–156. Springer Verlag. LNC 1703.

Gabaldon, A. 2002. Non-markovian control in the situation calculus. In *Eighteenth national conference on Artificial intelligence*, 519–524. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Harel, D.; Tiuryn, J.; and Kozen, D. 2000. *Dynamic Logic*. Cambridge, MA, USA: MIT Press.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12(10):576–580.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions* 31(1-3):59–83.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University.

Pinto, J. 1994. *Temporal Reasoning in the Situation Calculus*. Ph.D. Dissertation, Department of Computer Science, University of Toronto, Toronto.

Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM* 46(3):325–364.

Plotkin, G. 1981. A structural approach to operational semantics, tr-daimi-fn 19. Technical report, Comp. Science Dpt., Aarhus University.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on foundations of Computer Science*, 46–57. IEEE Computer Society.

Pnueli, A. 1981. A temporal logic of concurrent programs. *Theoretical Computer Science* 13:45–60.

Rajan, S.; Shankar, N.; and Srivas, M. 1995. An integration of model checking with automated proof checking. In Wolper, P., ed., *Proceedings of the 7th International Conference on Computer Aided Verification*, 84–97. Springer Verlag. LNC 939.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. Cambridge: MIT Press.

Ternovskaia, E. 1999. Automata theory for reasoning about actions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 153–158.

Vardi, M., and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press.

Wolper, P. 1998. The algorithmic verification of reactive systems. 1998 francqui chair lectures given at the fundp (namur). Lecture Notes, http://www.montefiore.ulg.ac.be/~pw/cours/.

4.2 Designing a FLUX Agent for the Dynamic Wumpus World

Designing a FLUX Agent for the Dynamic Wumpus World

Michael Thielscher
Department of Computer Science
Dresden University of Technology, Germany
mit@inf.tu-dresden.de

Abstract

Following an earlier proposal by Michael Genesereth, the Wumpus World has been described in (Russell & Norvig 2003) as an example for autonomous agents that need to reason logically about their actions and sensor information they acquire. The dynamic Wumpus World extends the original specification by several challenging aspects: a more complex environment, dynamic changes, and unexpected action failure. In (Thielscher 2005b) we have specified an agent for the simple Wumpus World using the high-level action programming language FLUX. In this paper we present the design of a FLUX agent for the extended problem with the help of advanced features such as dynamic state properties and a solution to the qualification problem.

Introduction

Cognitive Robotics (McCarthy 1958; Lespérance *et al.* 1994) is concerned with the problem of endowing agents with the high-level cognitive capability of reasoning. Intelligent agents rely on this ability when drawing inferences from sensor data acquired over time, when acting under incomplete information, and in order to exhibit plan-oriented behavior. For this purpose, agents form a mental model of their environment, which they constantly update to reflect the changes they have effected and the sensor information they have acquired. A simulated environment, the Wumpus World as defined in (Russell & Norvig 2003) is a good example of a problem for controlling an agent that needs to choose its actions not only on the basis of the current status of its sensors but also on the basis of what it has previously observed or done. Moreover, some properties of the environment can be observed only indirectly, which requires the agent to logically combine observations made at different stages. The dynamic Wumpus World[1] extends the original problem by a variety of challenging features: a more complex environment with walls between cells, dynamic changes, and unexpected action failures which are not immediately recognizable.

In (Thielscher 2005b) we have specified an agent for the simple Wumpus World using the high-level action programming language FLUX (Thielscher 2005a), which supports the design of intelligent agents that reason about their actions on the basis of the fluent calculus (Thielscher 1999). A constraint logic program, FLUX comprises a method for encoding incomplete states along with a technique of updating these states according to a declarative specification of the elementary actions and sensing capabilities of an agent. Incomplete states are represented by lists (of fluents) with variable tail, and negative and disjunctive state knowledge is encoded by constraints. In a series of experiments with a grid of fixed size (Sardina & Vassos 2005), the runtime behavior of the FLUX agent, with its efficient constraint solving mechanism, turned out to be superior to that of a similar GOLOG agent for the Wumpus World. Moreover, thanks to a solution to the computational frame problem under incomplete information, paired with the inference technique of progression, experiments described in (Thielscher 2005b) showed that the FLUX agent scaled up well.

In this paper we present the design of a FLUX agent for the dynamic Wumpus World, which requires to combine three extensions of the original framework:

- We use dynamic state properties in FLUX, which may change independent of the actions of the agent; hence, the agent can rely on them only at the time when they are explicitly sensed.
- We use the notion of implicational state constraints in FLUX (Thielscher 2005c) in order to represent conditional dependencies among fluents that arise from the fact that in the extended Wumpus World the presence of a pit can only be sensed if it is not obstructed by a wall.
- We address the Qualification Problem (McCarthy 1977), which arises whenever the successful execution of actions cannot be predicted with certainty. Agents for the dynamic Wumpus World rely on a solution to this problem in order to be able both to realize that some of their foregoing actions must have failed and to recover from this (Thielscher 2001).

The contribution of this paper is not to provide a new theoretical result, but rather to demonstrate how an existing knowledge representation language can be used to axiomatize a non-trivial domain, and how the action programming language and system FLUX can be used to design an intelligent agent that employs its ability to reason about actions to effectively act in this complex environment.

[1] www.cl.inf.tu-dresden.de/~mit/LRAPP/

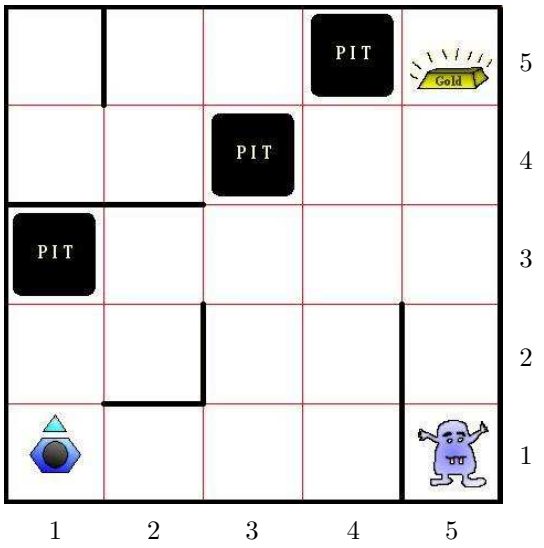


Figure 1: An example scenario in a dynamic Wumpus World where the 5×5 -cave features three pits, gold in cell $(5, 5)$, and the Wumpus currently in cell $(5, 1)$. The agent is in its home square $(1, 1)$ currently facing north.

The paper is organized as follows. The next section contains an informal description of the dynamic Wumpus World. Thereafter we axiomatize this environment as a fluent calculus theory, which provides the necessary background knowledge for an agent to reason about the effects of its actions in this world. We then describe a strategy for a FLUX agent for the dynamic Wumpus World, and in the final section we report on some initial experiments with our program.

The Dynamic Wumpus World

Just like in the simple Wumpus World, the agent moves in a regular grid of cells, some of which contain bottomless pits, which are to be avoided, and somewhere in the grid there is a hostile creature called Wumpus and a heap of gold, which the agent should find and bring home. The agent can indirectly sense the presence of a pit and the Wumpus by noticing a breeze and a stench, respectively, next to these cells, and it can sense the gold once it enters the right location. The agent has exactly one arrow which it can shoot at any time in any direction and which kills the Wumpus if the latter happens to be somewhere on the trajectory.

The structure of the environment in the dynamic Wumpus World is, however, a more complex one, where cells may be separated by walls; see Figure 1 for an example. A wall can be sensed only when the agent stands next to it and faces it. Furthermore, the Wumpus may make arbitrary moves (synchronously with any physical action of the agent). The most challenging feature of the extended problem is that the action of going forward to an adjacent cell may fail with a small likelihood, and the agent has no direct means to notice this failure. Suppose, for example, the agent has acquired complete knowledge of the wall structure in Figure 1 and happens to be in cell $(5, 3)$, from where it attempts to take a step northward. Even if the agent afterwards turns to check the walls around its current location, it is impossible to decide whether it has actually reached cell $(5, 4)$, for the wall structure in the two cells are identical. Only by making a further step northward to $(5, 5)$ (and verifying that it now faces a wall) can the agent be sure that its actions were indeed successful. Therefore, an agent that maintains an internal model of the environment has to take into account that this model may be erroneous.

The Background Theory

Fluents and states

In the fluent calculus, states are axiomatized on the basis of atomic components which are modeled as functions into the pre-defined sort FLUENT. The FLUX agent for the original Wumpus World described in (Thielscher 2005b) employs a systematic exploration strategy based on auxiliary parameters encoding the cells that it has visited and the path it has taken in order to backtrack from regions that have been completely explored. These parameters were kept outside the internal world model of the agent. In the dynamic Wumpus World, however, the agent's world model may be erroneous due to unnoticed failures when moving around. A successful recovery from action failure then requires not only to re-adjust its own positions, but also to revise its belief regarding the cells it has visited and the current path. These parameters must therefore be part of the internal state representation and hence be modeled as additional fluents. Because the Wumpus moves around freely, a further difference to the state representation in (Thielscher 2005b) is that the location of the creature can no longer be modeled as a fluent (to which the frame assumption applies); rather it has to be treated as a dynamic, situation-dependent property. The

Name	Type	Meaning
At	$\mathbb{N} \times \mathbb{N} \mapsto$ FLUENT	position of the agent
Facing	$\{1,2,3,4\} \mapsto$ FLUENT	orientation of the agent
Has	$\{Arrow, Gold\} \mapsto$ FLUENT	possessions of the agent
Gold	$\mathbb{N} \times \mathbb{N} \mapsto$ FLUENT	location of the gold
Pit	$\mathbb{N} \times \mathbb{N} \mapsto$ FLUENT	cells containing a pit
Dead	FLUENT	Wumpus is dead
Wall	$\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mapsto$ FLUENT	walls between two cells
Visited	$\mathbb{N} \times \mathbb{N} \mapsto$ FLUENT	visited cells
Backtrack	$\mathbb{N} \times \mathbb{N} \times \{1,2,3,4\} \mapsto$ FLUENT	element of the current path

Figure 2: The fluents for the dynamic Wumpus World. The numbers $1 \ldots 4$ encode the four directions north, east, south, and west. An instance $Backtrack(x, y, d)$ is true if the agent entered square (x, y) by going into direction d.

table in Figure 2 lists the fluents that we use to model the dynamic Wumpus World, the first six of which are adopted from the original axiomatization in (Thielscher 2005b).

States in the fluent calculus are formalized using the pre-defined sort STATE. Every FLUENT is of this sort, representing a singleton state, and the special function $\circ$: STATE $\times$ STATE $\mapsto$ STATE is used to compose sub-states. The special constant $\emptyset$: STATE denotes the empty state. A fluent is then defined to hold in a state just in case the latter contains it:[2]

$$Holds(f, z) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z'$$

This definition is accompanied by the foundational axioms of the fluent calculus, which ensure that a state can be identified with the fluents that hold in it; see, e.g., (Thielscher 2005a). Let Z_0 denote the initial state of the agent, then the agent may know the following:

$$\begin{array}{l} Holds(At(x,y), Z_0) \equiv x = 1 \wedge y = 1 \\ Holds(Facing(d), Z_0) \equiv d = 1 \\ Holds(Has(Arrow), Z_0) \wedge \neg Holds(Has(Gold), Z_0) \\ \neg Holds(Dead, Z_0) \wedge \neg Holds(Pit(1,1), Z_0) \\ Holds(Visited(x,y), Z_0) \equiv x = 1 \wedge y = 1 \\ (\exists x, y)\, Holds(Gold(x,y), Z_0) \end{array} \quad (1)$$

Actions and situations

Adopted from the situation calculus (McCarthy 1963; Reiter 2001), actions are modeled in the fluent calculus by terms into the pre-defined sort ACTION. Sequences of actions are represented by situations (sort SIT) with the help of a constant S_0 : SIT, denoting the initial situation, and the function Do : ACTION $\times$ SIT $\mapsto$ SIT, denoting the successor situation reached after performing an action in a situation. Situations and states are related by the pre-defined function $State$: SIT $\mapsto$ STATE, accompanied by the definition[3]

$$Holds(f, s) \stackrel{\text{def}}{=} Holds(f, State(s))$$

[2]Throughout the paper, variables of sort FLUENT and STATE will be denoted by the variables f and z, respectively, possibly with sub- or superscript.

[3]Throughout the paper, variables of sort ACTION and SIT will be denoted by the variables a and s, respectively, possibly with sub- or superscript.

The agent in the dynamic Wumpus World can perform the same actions as in the original problem; see Figure 3. As in the situation calculus, preconditions of actions are axiomatized with the help of the predicate $Poss$: ACTION $\times$ SIT. In the Wumpus World, all action can always be executed with the exception of *Go*, *Shoot*, and *Exit*, whose preconditions are as follows:

$$\begin{array}{rcl} Poss(Go, s) & \equiv & (\exists d, x, y, x', y')\, (Holds(At(x,y), s) \wedge \\ & & \quad Holds(Facing(d), s) \wedge \\ & & \quad Adjacent(x, y, d, x', y') \wedge \\ & & \quad \neg Holds(Wall(x, y, x', y'), s)\,) \end{array}$$

$$Poss(Shoot, s) \equiv Holds(Has(Arrow), s)$$

$$Poss(Exit, s) \equiv Holds(At(1,1), s)$$

The auxiliary predicate $Adjacent(x, y, d, x', y')$ means that (x', y') is adjacent to (x, y) in direction d.

Effects of actions are specified with the help of an axiomatic definition for removal and addition of fluents:

$$\begin{array}{rcl} z_1 - f = z_2 & \stackrel{\text{def}}{=} & (z_2 = z_1 \vee z_2 \circ f = z_1) \\ & & \quad \wedge \neg Holds(f, z_2) \\ z_1 + f = z_2 & \stackrel{\text{def}}{=} & z_2 = z_1 \circ f \end{array}$$

Based on this definition, the frame problem is solved in the fluent calculus by so-called state update axioms, which define the effects of an action a in a situation s as the difference between the current $State(s)$ and its successor $State(Do(a, s))$; for example,

$$\begin{array}{l} Poss(Grab, s) \supset \\ \quad (\exists x, y)\, (Holds(At(x,y), s) \wedge Holds(Gold(x,y), s) \wedge \\ \quad\quad State(Do(Grab, s)) = State(s) + Has(Gold)\,) \\ \quad \vee \\ \quad \neg(\exists x, y)\, (Holds(At(x,y), s) \wedge Holds(Gold(x,y), s)) \wedge \\ \quad\quad State(Do(Grab, s)) = State(s) \end{array}$$

$$Poss(Exit, s) \supset State(Do(Exit, s)) = State(s) - At(1,1)$$

Sensor information may provide the agent with additional knowledge of a successor state. We represent the result of sensing in the dynamic Wumpus World with the help of additional situation-dependent predicates:

$$Stench, Breeze, Glitter, Bump, Scream : \text{SIT}$$

Name	Type	Meaning
Go	ACTION	go forward to the adjacent cell
TurnRight	ACTION	make a quarter turn clockwise
TurnLeft	ACTION	make a quarter turn counterclockwise
Grab	ACTION	grab the gold
Shoot	ACTION	shoot the arrow
Exit	ACTION	exit the cave

Figure 3: The actions for the dynamic Wumpus World.

Because the Wumpus moves around freely and its position is not represented as a fluent (which would persists by default), the predicate $\mathit{Stench}(s)$ will only be used for the definition of the agent's strategy. Predicate $\mathit{Breeze}(s)$ provides knowledge as to whether one of the four cells next to the agent's location contains a pit, provided there is no wall in between:

$$\begin{aligned}\mathit{Breeze}(s) \equiv (\exists x, y, x', y', d)\,(&\mathit{Holds}(\mathit{At}(x, y), s) \wedge \\ &\mathit{Holds}(\mathit{Pit}(x', y'), s) \wedge \\ &\mathit{Adjacent}(x, y, d, x', y') \wedge \\ &\neg\mathit{Holds}(\mathit{Wall}(x, y, x', y'), s)\,)\end{aligned}$$

A glitter indicates that gold is at the current location:

$$\begin{aligned}\mathit{Glitter}(s) \equiv (\exists x, y)\,(&\mathit{Holds}(\mathit{At}(x, y), s) \wedge \\ &\mathit{Holds}(\mathit{Gold}(x, y), s)\,)\end{aligned}$$

A bump is perceived in case the agent faces a wall:

$$\begin{aligned}\mathit{Bump}(s) \equiv (\exists x, y, d, x', y')\,(&\mathit{Holds}(\mathit{At}(x, y), s) \wedge \\ &\mathit{Holds}(\mathit{Facing}(d), s) \wedge \\ &\mathit{Adjacent}(x, y, d, x', y') \wedge \\ &\mathit{Holds}(\mathit{Wall}(x, y, x', y'), s)\,)\end{aligned}$$

For the sake of simplicity, we assume that the knowledge of the five sensing predicates is given to the agent whenever it performs one of the actions *TurnRight*, *TurnLeft*, *Shoot*, or *Go*.[4] The state update axioms for the first three of these actions are as follows:

$$\begin{aligned}&\mathit{Poss}(\mathit{TurnRight}, s) \supset \\ &\quad(\exists d)\,(\mathit{Holds}(\mathit{Facing}(d), s) \wedge \\ &\qquad\mathit{State}(\mathit{Do}(\mathit{TurnRight}, s)) = \\ &\qquad\quad\mathit{State}(s) - \mathit{Facing}(d) + \mathit{Facing}(d \bmod 4 + 1)\,)\end{aligned}$$

$$\begin{aligned}&\mathit{Poss}(\mathit{TurnLeft}, s) \supset \\ &\quad(\exists d)\,(\mathit{Holds}(\mathit{Facing}(d), s) \wedge \\ &\qquad\mathit{State}(\mathit{Do}(\mathit{TurnRight}, s)) = \\ &\qquad\quad\mathit{State}(s) - \mathit{Facing}(d) + \mathit{Facing}((d + 2) \bmod 4 + 1)\,)\end{aligned}$$

$$\begin{aligned}&\mathit{Poss}(\mathit{Shoot}, s) \supset \\ &\quad\mathit{Scream}(\mathit{Do}(\mathit{Shoot}, s)) \wedge \\ &\qquad\mathit{State}(\mathit{Do}(\mathit{Shoot}, s)) = \mathit{State}(s) - \mathit{Has}(\mathit{Arrow}) + \mathit{Dead} \\ &\quad\vee \\ &\quad\neg\mathit{Scream}(\mathit{Do}(\mathit{Shoot}, s)) \wedge \\ &\qquad\mathit{State}(\mathit{Do}(\mathit{Shoot}, s)) = \mathit{State}(s) - \mathit{Has}(\mathit{Arrow})\end{aligned}$$

[4] This is conceptually simpler than the use of an extension of the fluent calculus which is based on an explicit model of the knowledge of an agent and which allows to explicitly reason about the effects of sensing actions (Thielscher 2000).

The last axiom uses the sensor predicate $\mathit{Scream}(s)$, which is true if the agent hears a scream, that is, in case the arrow hit the Wumpus.

Abnormalities

The most challenging aspect of the dynamic Wumpus World is the possibility of unexpected action failure. Specifically, when the agent attempts to go forward, it may actually remain where it was. The main difficulty stems from the fact that this is usually not immediately recognizable. This is an instance of the general Qualification Problem (McCarthy 1977), which arises whenever unexpected circumstances, albeit unlikely, may prevent an autonomous agent from performing the intended actions. Planning and acting under this proviso requires the agent to rigorously assume away, by default, all of the possible but unlikely *abnormal qualifications* of its actions, lest the agent is unable to make decisions which are perfectly rational although they cannot guarantee success. We adopt here a solution to this problem where a fluent calculus axiomatization is embedded in a so-called default theory (Reiter 1980; Thielscher 2001). To this end, let $\mathbf{Ab}(s)$ be a situation-dependent predicate denoting the "abnormality" that a *Go*-action fails in situation s. The state update axiom for this action then comprises both the regular and the abnormal effect:

$$\begin{aligned}&\mathit{Poss}(\mathit{Go}, s) \supset \\ &\quad\neg\mathbf{Ab}(s) \wedge (\exists d, x, y, x', y')\,(\mathit{Holds}(\mathit{At}(x, y), s) \wedge \\ &\qquad\mathit{Holds}(\mathit{Facing}(d), s) \wedge \\ &\qquad\mathit{Adjacent}(x, y, d, x', y') \wedge \\ &\qquad[\neg\mathit{Holds}(\mathit{Visited}(x', y'), s) \wedge \mathit{State}(\mathit{Do}(\mathit{Go}, s)) = \\ &\qquad\quad\mathit{State}(s) - \mathit{At}(x, y) \\ &\qquad\qquad + \mathit{At}(x', y') + \mathit{Visited}(x', y') \\ &\qquad\qquad\qquad + \mathit{Backtrack}(x', y', d) \\ &\qquad\vee \mathit{Holds}(\mathit{Visited}(x', y'), s) \wedge \mathit{State}(\mathit{Do}(\mathit{Go}, s)) = \\ &\qquad\quad\mathit{State}(s) - \mathit{At}(x, y) + \mathit{At}(x', y')\,]\,) \\ &\quad\vee \\ &\quad\mathbf{Ab}(s) \wedge \mathit{State}(\mathit{Do}(\mathit{Go}, s)) = \mathit{State}(s)\end{aligned} \tag{2}$$

The entire fluent calculus axiomatization Σ is then accompanied by a single default rule:

$$\frac{:\ \neg\mathbf{Ab}(s)}{\neg\mathbf{Ab}(s)} \tag{3}$$

Reasoning with a default theory is based on the notion of *extensions*, which—thanks to the simple structure of our default (Reiter 1980)—can be defined as maximally consistent

sets

$$\Sigma \cup \{\neg \mathbf{Ab}(\sigma) : \sigma \text{ ground SIT term}\}$$

This allows agents to predict the success of a *Go*-action by default, but also to find explanations for (and to recover from) unexpected observations. For example, suppose for the sake of argument that the agent is already aware of the wall between cells $(2,1)$ and $(2,2)$ (cf. Figure 1). Let S denote the current situation where the agent is at $(1,1)$ and faces north. Suppose further that the agent turns right, then attempts to go to square $(1,2)$, and finally turns left to double-check the wall:

$$S' = \mathit{Do}(\mathit{TurnLeft}, \mathit{Do}(\mathit{Go}, \mathit{Do}(\mathit{TurnRight}, S)))$$

Our default theory then contains a unique extension, where all instances of **Ab** are false and which thus entails

$$\mathit{Holds}(\mathit{At}(2,1), S') \wedge \mathit{Holds}(\mathit{Facing}(1), S')$$

and hence

$$\mathit{Bump}(S')$$

due to the wall between $(2,1)$ and $(2,2)$. If, however, the observation

$$\neg \mathit{Bump}(S')$$

is added to the axiomatization, the default theory thus augmented admits a unique extension again, in which all instances of **Ab** are false except for $\mathbf{Ab}(\mathit{Do}(\mathit{TurnRight}, S))$. This together with the state update axioms, in particular the second disjunct in (2), entails

$$\mathit{Holds}(\mathit{At}(1,1), S')$$

This shows how an agent can explain an observation which contradicts its expectations, and how it can automatically recover by finding a minimal set of positive **Ab**-instances which are consistent with the observations.

FLUX

FLUX (Thielscher 2005a; 2005d) is a high-level programming method for the design of intelligent agents that reason about their actions on the basis of the fluent calculus (Thielscher 1999). A constraint logic program, FLUX comprises a method for encoding incomplete states along with a technique of updating these states according to a declarative specification of the elementary actions and sensing capabilities of an agent. Incomplete states are represented by lists (of fluents) with variable tail, and negative and disjunctive state knowledge is encoded by constraints. The incomplete initial state knowledge of the agent in the dynamic Wumpus World (cf. (1)), for example, is encoded in FLUX by the following clause:

```
init(Z0) :-
  Z0 = [at(1,1),facing(1),has(arrow),
        visited(1,1),gold(X,Y) | Z],
  not_holds_all(at(_,_),Z),
  not_holds_all(facing(_),Z),
  not_holds_all(visited(_,_),Z),
  not_holds(has(gold),Z),
  not_holds(dead,Z),
  not_holds(pit(1,1),Z).
```

Constraint `not_holds(f,z)` encodes the fluent calculus formula $\neg \mathit{Holds}(f,z)$, and `not_holds_all(f,z)` stands for $(\forall \vec{x}) \neg \mathit{Holds}(f,z)$, where $\vec{x}$ are the variables in f.

State update axioms in the fluent calculus can be directly translated into FLUX as clauses which define the predicate `state_update(Z1,A,Z2,Y)` with the intended meaning that performing action a in state z_1 and sensing $\vec{y}$ yields updated state z_2. As an example, we give the specification of the regular effects of action *Go* with sensor input for breeze, glitter, and wall—the other state update axioms are encoded in a similar fashion:

```
state_update(Z1,go,Z2,[Br,Gl,Bm]) :-
  holds(at(X,Y),Z1),
  holds(facing(D),Z1),
  adjacent(X,Y,D,X1,Y1),
  ( not_holds(visited(X1,Y1),Z1),
    update(Z1,[at(X1,Y1),
               visited(X1,Y1),
               backtrack(X1,Y1,D)],
              [at(X,Y)],Z2)
    ;
    holds(visited(X1,Y1),Z1),
    update(Z1,[at(X1,Y1)],
              [at(X,Y)],Z2) ),
  breeze(X1,Y1,Br,Z2),
  glitter(X1,Y1,Gl,Z2),
  bump(X1,Y1,D,Bm,Z2).
```

Standard FLUX predicate `update(Z1,P,N,Z2)` means that the update of incomplete state z_1 by positive and negative effects p and n, respectively, results in state z_2.

Due to the possible existence of walls, the evaluation of the sensor input in the dynamic Wumpus World requires to use a recent extension of FLUX which allows to express conditional state knowledge in form of a state constraint `if_then_holds(f,g,z)`, encoding the fluent calculus formula $\mathit{Holds}(f,z) \supset \mathit{Holds}(g,z)$ (Thielscher 2005c):

```
breeze(X,Y,Percept,Z) :-
  XE#=X+1, XW#=X-1, YN#=Y+1, YS#=Y-1,
  ( Percept=false,
      if_then_holds(pit(XE,Y),
                    wall(X,Y,XE,Y),Z),
      if_then_holds(pit(XW,Y),
                    wall(X,Y,XW,Y),Z),
      if_then_holds(pit(X,YN),
                    wall(X,Y,X,YN),Z),
      if_then_holds(pit(X,YS),
                    wall(X,Y,X,YS),Z)
    ; Percept=true ).
```

The current expressiveness of FLUX does not allow to give a complete account of the conditional knowledge of pits that follows from sensing a breeze. Fortunately, this is not necessary in order that the agent functions effectively, because the agent will always enter a square only if this is *known* to be safe. Hence, there is no need to distinguish between not knowing whether a cell is safe and knowing for sure that there is a pit!

The other two sensors are encoded in a straightforward manner:

```
glitter(X,Y,Percept,Z) :-
  Percept=false,
    not_holds(gold(X,Y),Z)
  ;
  Percept=true,
    holds(gold(X,Y),Z).

bump(X,Y,D,Percept,Z) :-
  adjacent(X,Y,D,X1,Y1),
  ( Percept=false,
      not_holds(wall(X,Y,X1,Y1),Z),
      not_holds(wall(X1,Y1,X,Y),Z)
    ;
    Percept=true,
      holds(wall(X,Y,X1,Y1),Z),
      holds(wall(X1,Y1,X,Y),Z) ).
```

The state update axiom given above merely encodes the normal effect of the *Go*-action. The solution to the Qualification Problem in FLUX (Thielscher 2005d) requires to encode the abnormal effects of actions by an additional clause; in our case (cf. (2)),

```
ab_state_update(Z1,go,Z2,
                [Br,Gl,Bm]) :-
  Z2 = Z1,
  holds(at(X,Y),Z2),
  holds(facing(D),Z2),
  breeze(X,Y,Br,Z2),
  glitter(X,Y,Gl,Z2),
  bump(X,Y,D,Bm,Z2).
```

The application of default rule (3) is modeled in FLUX as follows: Predicate `state_update` is used by default whenever the world model gets updated after an action has been performed (via predicate `execute`, see below). Only when the agent, upon executing an action, makes an observation which is inconsistent with the world model thus updated, predicate `ab_state_update` can be additionally used in order to find a sequence of updates which is consistent with the sequence of observations that have been made; for details we refer to (Thielscher 2005d).

The Strategy

FLUX allows to define complex behaviors by means of strategy programs. These strategies are independent of the background specification, which allows to devise and compare different strategies. In the following, we describe a specific strategy for the dynamic Wumpus World, which is similar to the systematic exploration defined in (Thielscher 2005b) for the simple environment.

After initializing the world model, the agent executes a loop where it either grabs the gold, if possible, and goes home, or it explores a new cell, if possible, and otherwise it backtracks. If the latter is not possible, too, then the agent believes it is at its home square and exits (without having found the gold). The latter may fail, because of an erroneous world model, in which case the agent remains somewhere in the environment and continues with its revised model:

```
main_loop(Z) :-
  knows_val([X,Y],at(X,Y),Z),
  ( knows(gold(X,Y),Z) ->
      execute(grab,Z,Z1), go_home(Z1)
    ;
    explore(X,Y,Z,Z1)
      -> main_loop(Z1)
    ;
    backtrack(X,Y,Z,Z1)
      -> main_loop(Z1)
    ;
    execute(exit,Z,Z1),
    ( knows_val([X1,Y1],at(X1,Y1),Z1)
        -> main_loop(Z1)
         ; true ) ).
```

Here, the standard FLUX predicate `knows_val(X,F,Z)` means that the agent knows arguments $\vec{x}$ of fluent f so that f holds in state z; predicate `knows(F,Z)` is true if fluent f is known to hold in state z; and predicate `execute(A,Z1,Z2)` means that the actual execution of action a in state z_1 leads to state z_2.

The agent can take an exploration step if it is adjacent to a cell which has not been visited, which is not blocked by a wall, and which is known not to house a pit. If these conditions are satisfied, then the agent enters this cell and makes three turns in order to gain information about the wall structure. The agent needs this information to decide whether it can continue with the exploration. Knowledge of the wall structure also helps with locating the pits in case a breeze is being sensed. But even when the agent knows the complete wall structure, double-checking the presence of walls helps with recognizing a failed *Go*-action as early as possible:

```
explore(X,Y,Z1,Z7) :-
  wumpus_alert(X,Y,Z1,Z2),
  adjacent(X,Y,D,X1,Y1),
  knows_not(visited(X1,Y1),Z2),
  knows_not(wall(X,Y,X1,Y1),Z2),
  knows_not(pit(X1,Y1),Z2)
  ->
  turn_to(D,Z2,Z3),
  execute(go,Z3,Z4),
  execute(turn_left,Z4,Z5),
  execute(turn_left,Z5,Z6),
  execute(turn_left,Z6,Z7).
```

Here, the standard FLUX predicate `knows_not(F,Z)` means that the agent knows that fluent f is false in state z. The auxiliary predicate `adjacent(X,Y,D,X1,Y1)` defines cell (x_1, y_1) to be adjacent to cell (x, y) in direction d. Auxiliary predicate `wumpus_alert` defines the reaction in case the agent senses a stench while it knows that the Wumpus is still alive. Whenever this happens, our agent either turns into a direction with no wall and shoots the arrow, in the hope of hitting the Wumpus. In case the agent has already used its arrow unsuccessfully, then upon sensing a stench it backtracks to the previous location (we omit the details). Finally, auxiliary predicate `turn_to(D,Z1,Z2)` means to turn into direction d.

If no exploration step is possible, the agent backtracks according to the following definition:

```
backtrack(X,Y,Z1,Z3) :-
  knows_val([D],backtrack(X,Y,D),Z1),
  R is (D+1) mod 4 + 1,
  turn_to(R,Z1,Z2),
  execute(go,Z2,Z3).
```

(Note that `R` is the reverse of direction `D`). According to its definition, predicate `backtrack(X,Y,Z1,Z3)` is false if state z_1 does not contain a fluent *Backtrack*(x, y, d), which means that the agent is in cell $(1, 1)$ (or rather that it believes it is).

Once the agent has found the gold, it uses the backtracking information to find its way to the home square. As above, if the agent believes it is at its home square then it exits, which may fail because of an erroneous world model, in which case the agent remains somewhere in the environment and continues with backtracking:

```
go_home(Z) :-
  knows_val([X,Y],at(X,Y),Z),
  ( backtrack(X,Y,Z,Z1)
      -> go_home(Z1)
    ;
    execute(exit,Z,Z1),
    ( knows_val([X1,Y1],at(X1,Y1),Z1)
        -> go_home(Z1)
         ; true ) ).
```

Discussion

In order to test the agent program, we ran a series of experiments with the scenario depicted in Figure 1. Of a total of 30 runs, 47% were successful in that the agent managed to locate and grab the gold and to exit safely. In 30% of the test cases, the agent stayed alive but did not find the gold, because the Wumpus came in the way and the agent did not guess its location correctly when shooting the arrow. In 23% of the runs the Wumpus killed the agent by moving into its location.

The agent was always able to recover from action failure. In some cases, where one or more *Go* actions failed at early stages, the agent built up a largely erroneous model of the environment and it took until the very end, when the agent intended to exit, to detect the abnormalities. Generally, the later an abnormality was detected, the longer it took to infer the correct position because the space of possible explanations is exponential in the total number of *Go* actions. One way to improve the computational behavior of this recovery process would be to begin with searching for explanations in which just $k = 1$ failure occurs, and to iteratively increase k until a consistent sequence of updates is found. Under the assumption that failures are rare, this would avoid computing a large portion of the search space. Generally speaking, abnormalities should only be used for the specification of truly exceptional effects of actions. If the *Go* action in the Wumpus World did not have a high chance of succeeding, then it had better be specified by a nondeterministic state update axiom, lest the agent has to frequently recover from an erroneous world model.

References

Lespérance, Y.; Levesque, H.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. 1994. A logical approach to high-level robot programming—a progress report. In Kuipers, B., ed., *Control of the Physical World by Intelligent Agents, Papers from the AAAI Fall Symposium*, 109–119.

McCarthy, J. 1958. Programs with Common Sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, volume 1, 77–84. (Reprinted in: (McCarthy 1990)).

McCarthy, J. 1963. *Situations and Actions and Causal Laws*. Stanford University, CA: Stanford Artificial Intelligence Project, Memo 2.

McCarthy, J. 1977. Epistemological problems of artificial intelligence. In Reddy, R., ed., *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1038–1044. Cambridge, MA: MIT Press.

McCarthy, J. 1990. *Formalizing Common Sense*. Norwood, New Jersey: Ablex. (Edited by V. Lifschitz).

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice-Hall.

Sardina, S., and Vassos, S. 2005. The wumpus world in IndiGolog: A preliminary report. In Morgenstern, L., and Pagnucco, M., eds., *Proceedings of the Workshop on Nonmonotonic Reasoning, Action and Change at IJCAI*, 90–95.

Thielscher, M. 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence* 111(1–2):277–299.

Thielscher, M. 2000. Representing the knowledge of a robot. In Cohn, A.; Giunchiglia, F.; and Selman, B., eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 109–120. Breckenridge, CO: Morgan Kaufmann.

Thielscher, M. 2001. The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence* 131(1–2):1–37.

Thielscher, M. 2005a. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming* 5(4–5):533–565.

Thielscher, M. 2005b. A FLUX agent for the Wumpus World. In Morgenstern, L., and Pagnucco, M., eds., *Proceedings of the Workshop on Nonmonotonic Reasoning, Action and Change at IJCAI*, 104–108.

Thielscher, M. 2005c. Handling implicational and universal quantification constraints in flux. In van Beek., ed., *Proceedings of the International Conference on Principle and Practice of Constraint Programming (CP)*, volume 3709 of *LNCS*, 667–681. Sitges, Spain: Springer.

Thielscher, M. 2005d. *Reasoning Robots: The Art and Science of Programming Robotic Agents*, volume 33 of *Applied Logic Series*. Kluwer.

4.3 A Semantics for ADL as Progression in the Situation Calculus

A Semantics for ADL as Progression in the Situation Calculus

Jens Claßen and **Gerhard Lakemeyer**
Department of Computer Science
RWTH Aachen
52056 Aachen
Germany
⟨classen|gerhard⟩@cs.rwth-aachen.de

Abstract

Lin and Reiter were the first to propose a purely declarative semantics of STRIPS by relating the update of a STRIPS database to a form of progression in the situation calculus. In this paper we show that a corresponding result can be obtained also for ADL. We do so using a variant of the situation calculus recently proposed by Lakemeyer and Levesque. Compared to Lin and Reiter this leads to a simpler technical treatment, including a new notion of progression.

Introduction

Lin and Reiter (Lin & Reiter 1997) were the first to propose a purely declarative semantics of STRIPS (Fikes & Nilsson 1971) by relating the update of a STRIPS database to a form of progression of a corresponding situation-calculus theory. More precisely, they show that when translating STRIPS planning problems into basic action theories of Reiter's situation calculus (Reiter 2001), then the STRIPS mechanism of adding and deleting literals after an action A is performed is correct in the sense that the conclusions about the future that can be drawn using the updated theory are the same as those drawn from the theory before the update.

Given that today's planning languages like PDDL (Fox & Long 2003) go well beyond STRIPS, it seems natural to ask whether Lin and Reiter's results can be extended along these lines. One advantage would be to have a uniform framework for specifying the semantics of planning languages. Perhaps more importantly, as we will argue in more detail at the end of the paper, this would also provide a foundation to bring together the planning and action language paradigms, which have largely developed independently after the invention of STRIPS.

In this paper we propose a first step in this direction by considering the ADL fragment of PDDL (Pednault 1989; 1994). In contrast to Lin and Reiter, we use a new variant of the situation calculus called $\mathcal{ES}$ recently proposed by Lakemeyer and Levesque (Lakemeyer & Levesque 2004). This has at least two advantages: for one, there is no need to switch the language when translating formulas of the planning language into the new situation calculus because there are no situation terms to worry about (in $\mathcal{ES}$, situations occur only in the semantics); for another, semantic definitions like progression become simpler as it is no longer necessary to consider arbitrary first-order structures but only certain ones over a fixed universe of discourse. As Lakemeyer and Levesque recently showed (Lakemeyer & Levesque 2005), these simplifications do not lead to a loss of expressiveness. In fact, they show that second-order $\mathcal{ES}$ captures precisely the non-epistemic fragment of the situation calculus and the action language Golog (Levesque *et al.* 1997).[1]

The main technical contributions of this paper are the following: we show how to translate an ADL problem description into a basic action theory of $\mathcal{ES}$; we develop a notion of progression, which is similar to that of Lin and Reiter but also simpler given the semantics underlying $\mathcal{ES}$; finally, we establish that updating an ADL database (called a *state*) after performing an action is correct in the sense that the resulting state corresponds precisely to progressing the corresponding basic action theory. The result is obtained for both closed and open-world states.

With the exception of Lin and Reiter (Lin & Reiter 1997), the approaches to giving semantics to planning languages have all been meta-theoretic. When Pednault introduced ADL (1989; 1994), he provided a semantics that defined operators as mappings between first-order structures that are defined by additions and deletions of tuples to the relations and functions of that structures. He presented a method of deriving a situation calculus axiomatization from ADL operator schema, but did not show the semantic correspondence between the two. Despite the fact that PDDL was built upon ADL, it was not until PDDL2.1 that a formal semantics was provided. The focus in (Fox & Long 2003) was more on formalizing the meaning of the newly introduced temporal extensions and concurrent actions; nonetheless, the predicate-logic subset of Fox and Long's semantics represents a generalization of Lifschitz' state transition semantics for STRIPS (1986). However, they compile conditional effects into the preconditions of the operators, propositionalize quantifiers and only consider the case of complete state descriptions. An exhaustive study of the expressiveness and compilability of different subsets of the propositional version of ADL is given in (Nebel 2000).

The paper proceeds as follows. We first introduce $\mathcal{ES}$ and show how basic action theories are formulated in this logic.

[1] The correspondence with the full situation calculus is close but not exact.

Next, we define ADL problem descriptions and provide a formal semantics by mapping them into basic action theories. We then define progression and establish the correctness of updating an ADL state with respect to progression. Before concluding, we give an outlook on applying the results to combine planning and the action language Golog.

The Logic $\mathcal{ES}$

For the purpose of this paper, we only need the objective (i.e. non-epistemic), first-order subset of $\mathcal{ES}$.

The Language

The language consists of formulas over symbols from the following vocabulary:

- variables $V = \{x_1, x_2, \ldots, y_1, y_2, \ldots, a_1, a_2, \ldots\}$;
- fluent predicates of arity k: $F^k = \{F_1^k, F_2^k, \ldots\}$; for example, *At*;
  we assume this list includes the distinguished predicate *Poss*;
- rigid functions of arity k: $G^k = \{g_1^k, g_2^k, \ldots\}$; for example, *paycheck*, *moveB*;
- connectives and other symbols: $=$, $\wedge$, $\neg$, $\forall$, $\Box$, round and square parentheses, period, comma.

For simplicity, we do not include rigid (non-fluent) predicates or fluent (non-rigid) functions. The *terms* of the language are the least set of expressions such that

1. Every first-order variable is a term;
2. If $t_1, \ldots, t_k$ are terms and $g \in G^k$, then $g(t_1, \ldots, t_k)$ is a term.

We let R denote the set of all ground terms. For simplicity, instead of having variables of the *action* sort distinct from those of the *object* sort as in the situation calculus, we lump both of these together and allow ourselves to use any term as an action or as an object. Finally, the *well-formed formulas* of the language form the least set such that

1. If $t_1, \ldots, t_k$ are terms and $F \in F^k$, then $F(t_1, \ldots, t_k)$ is an (atomic) formula;
2. If t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula;
3. If t is a term and α is a formula, then $[t]\alpha$ is a formula;
4. If α and β are formulas, then so are $(\alpha \wedge \beta)$, $\neg\alpha$, $\forall x.\alpha$, $\Box\alpha$.

We read $[t]\alpha$ as "α holds after action t" and $\Box\alpha$ as "α holds after any sequence of actions". As usual, we treat $\exists x.\alpha$, $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, and $(\alpha \equiv \beta)$ as abbreviations. We call a formula without free variables a *sentence*.

In the following, we will call a sentence *fluent*, when it does not contain $\Box$ and $[t]$ operators and does not mention *Poss*. In addition, we introduce the following special notation: A *type* τ is a symbol from F^1, i.e. a unary predicate. Then we define:

$$\forall x{:}\tau.\, \phi \stackrel{def}{=} \forall x.\, \tau(x) \supset \phi$$

We will often use the vector notation to refer to a tuple of terms ($\vec{t}$) or types ($\vec{\tau}$). If $\vec{r}$ denotes $r_1, \ldots, r_k$ and $\vec{t}$ stands for $t_1, \ldots, t_k$, then $(\vec{r} = \vec{t})$ means $(r_1 = t_1) \wedge \cdots \wedge (r_k = t_k)$. Further, $\vec{\tau}(\vec{t})$ serves as an abbreviation for $\tau_1(t_1) \wedge \cdots \wedge \tau_k(t_k)$.

The semantics

Intuitively, a world w will determine which fluents are true, but not just initially, also after any sequence of actions. Formally, let P denote the set of all pairs $\sigma{:}\rho$ where $\sigma \in R^*$ is considered a sequence of actions, and $\rho = F(r_1, \ldots, r_k)$ is a ground fluent atom from F^k. A *world* then is a mapping from P to truth values $\{0, 1\}$.

First-order variables are interpreted substitutionally over the rigid terms R, that is, R is treated as being isomorphic to a fixed universe of discourse. This is similar to $\mathcal{L}$ (Levesque & Lakemeyer 2001), where standard names are used as the domain.

Here is the complete semantic definition: Given a world w, for any formula α with no free variables, we define $w \models \alpha$ as $w, \langle\,\rangle \models \alpha$ where $\langle\,\rangle$ denotes the empty action sequence and

$w, \sigma \models F(r_1, \ldots, r_k)$ iff $w[\sigma{:}F(r_1, \ldots, r_k)] = 1$;
$w, \sigma \models (r_1 = r_2)$ iff r_1 and r_2 are identical;
$w, \sigma \models (\alpha \wedge \beta)$ iff $w, \sigma \models \alpha$ and $w, \sigma \models \beta$;
$w, \sigma \models \neg\alpha$ iff $w, \sigma \not\models \alpha$;
$w, \sigma \models \forall x.\, \alpha$ iff $w, \sigma \models \alpha^x_r$, for every $r \in R$;
$w, \sigma \models [r]\alpha$ iff $w, \sigma \cdot r \models \alpha$;
$w, \sigma \models \Box\alpha$ iff $w, \sigma \cdot \sigma' \models \alpha$, for every $\sigma' \in R^*$;

The notation α^x_t means the result of simultaneously replacing all free occurrences of the variable x by the term t; $\sigma_1 \cdot \sigma_2$ denotes the concatenation of the two action sequences.

When Σ is a set of sentences and α is a sentence, we write $\Sigma \models \alpha$ (read: Σ logically entails α) to mean that for every w, if $w \models \alpha'$ for every $\alpha' \in \Sigma$, then $w \models \alpha$. Finally, we write $\models \alpha$ (read: α is valid) to mean $\{\} \models \alpha$.

Basic Action Theories

As shown in (Lakemeyer & Levesque 2004), we are able to define basic action theories in a way very similar to those originally introduced by Reiter:

Given a set $\mathcal{F}$ of fluent predicates, a set of sentences Σ is called a *basic action theory* over $\mathcal{F}$ iff it only mentions the fluents in $\mathcal{F}$ and is of the form $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, where

- Σ_0 is a finite set of fluent sentences,
- Σ_{pre} is a singleton of the form[2] $\Box\mathit{Poss}(a) \equiv \pi$, where π is fluent with a being the only free variable;
- Σ_{post} is a finite set of successor state axioms of the form[3] $\Box[a]F(\vec{x}) \equiv \gamma_F$, one for each fluent $F \in \mathcal{F} \setminus \{\mathit{Poss}\}$, where γ_F is a fluent sentence whose free variables are among $\vec{x}$ and a.

[2] We follow the convention that free variables are universally quantified from the outside. We also assume that $\Box$ has lower syntactic precedence than the logical connectives, so that $\Box\mathit{Poss}(a) \equiv \pi$ stands for $\forall a.\Box(\mathit{Poss}(a) \equiv \pi)$.

[3] The $[t]$ construct has higher precedence than the logical connectives. So $\Box[a]F(\vec{x}) \equiv \gamma_F$ abbreviates $\forall a.\Box([a]F(\vec{x}) \equiv \gamma_F)$.

The idea is that Σ_0 represents the initial database, Σ_{pre} is one large precondition axiom and Σ_{post} the set of successor state axioms for all fluents in $\mathcal{F}$ (incorporating Reiter's solution (1991) to the frame problem).

The ADL subset of PDDL

ADL was proposed by Pednault (1989) as a planning formalism that constitutes a compromise between the highly expressive situation calculus on the one hand and the computationally more beneficial STRIPS language on the other. Recently, it has been used as the basis for the definition of a general planning domain definition language called PDDL (Ghallab *et al.* 1998; Gerevini & Long 2005a).

Here, we are interested in the ADL subset of PDDL, i.e. the language we obtain by only allowing the `:adl` requirement to be set. This implies that, beyond the definition of standard STRIPS operators, equality is supported as built-in predicate and preconditions may contain negation, disjunction and quantification (therefore they are normal first-order formulas using the domain's predicates together with the action's object parameters and the domain objects as the only function symbols). Further, conditional effects are allowed and objects may be typed.

ADL Operators: The General Form

Formally, an *ADL operator* A is given by a triple $(\vec{y}{:}\vec{\tau}, \pi_A, \epsilon_A)$, where $\vec{y}{:}\vec{\tau}$ is a list of variable symbols with associated types[4], π_A is a precondition formula with free variables among $\vec{y}$ and ϵ_A is an effect formula with free variables among $\vec{y}$. The $\vec{y}$ are the action's *parameters*, π_A is called the *precondition* and ϵ_A the *effect* of A, for short. The name of the operator A has to be a symbol from G^p (the function symbols of arity p), where p is the number of parameters $\vec{y}$ of A (possibly zero).

A *precondition formula* is of the following form: An atomic formula $F(\vec{t})$ is a precondition formula, if each of the t_i is either a variable or constant (i.e. terms are not nested). Similarly, an equality atom $(t_1 = t_2)$ is a precondition formula, if each t_i is a variable or a constant. If ϕ_1 and ϕ_2 are precondition formulas, then so are $\phi_1 \wedge \phi_2$, $\neg\phi_1$ and $\forall x{:}\tau.\phi_1$[5].

The *effect formulas* are defined as follows: An atomic formula $F(\vec{t})$ is an effect formula, if each of the t_i is either a variable or a constant. Similarly, a negated atomic formula $\neg F(\vec{t})$ is an effect formula, if each t_i is a variable or a constant. If ψ_1 and ψ_2 are effect formulas, then $\psi_1 \wedge \psi_2$ and $\forall x{:}\tau.\psi_1$ are as well. If γ is a precondition formula and ψ is an effect formula not containing "$\Rightarrow$" and "$\forall$", then $\gamma \Rightarrow \psi$ is an effect formula.

Therefore, effect formulas are always conjunctions of single effects. An effect of the form $\gamma \Rightarrow \psi$ is called a *conditional effect*. Nesting of conditional effects is disallowed.

[4] $\vec{y}{:}\vec{\tau}$ is to be understood as a list of pairs $y_i{:}\tau_i$.

[5] Recall that $\vee$, $\exists$ etc. are treated as abbreviations, therefore disjunction and existential quantification is allowed as well.

ADL Operators: The Normal Form

We say that an ADL operator A is in *normal form*, if its effect ϵ_A looks as follows:

$$\begin{array}{l} \bigwedge_{F_j} \forall \vec{x_j}{:}\vec{\tau_{F_j}}.\ (\gamma^{+}_{F_j,A}(\vec{x_j}) \Rightarrow F_j(\vec{x_j})) \wedge \\ \bigwedge_{F_j} \forall \vec{x_j}{:}\vec{\tau_{F_j}}.\ (\gamma^{-}_{F_j,A}(\vec{x_j}) \Rightarrow \neg F_j(\vec{x_j})) \end{array} \quad (1)$$

We mean here that for each F_j, there is *at most* one conjunct $\cdots \Rightarrow F_j(\vec{x})$ and also at most one conjunct $\cdots \Rightarrow \neg F_j(\vec{x})$; neither is it required that there are conjuncts for all predicates of a theory nor is the ordering important.

ADL Problem Descriptions

A problem description for ADL now is given by:

1. a finite list of types $\tau_1, \ldots, \tau_l$, *Object* (*Object* is a special type that has to be always included);
   - along with this a finite list of statements of the form
     $$\tau_i{:}(\text{either } \tau_{i_1} \cdots \tau_{i_{k_i}}) \quad (2)$$
     defining some of the types as compound[6]types; a *primitive type* is one that does not appear on the left-hand side of such a definition and is distinct from *Object*;
2. a finite list of fluent predicates $F_1, \ldots, F_n$;
   - associated with each F_j a list of types $\tau_{j_1}, \ldots, \tau_{j_{k_j}}$ (one for each argument of F_j)
3. a finite list of objects with associated primitive types $o_1{:}\tau_{o_1}, \ldots, o_k{:}\tau_{o_k}$ (object symbols are taken from G^0);
4. a finite list of ADL operators $A_1, \ldots, A_m$ (with associated descriptions in the above general form);
5. an initial state I (see below) and
6. a goal description G in form of a precondition formula.

I and G may only contain the symbols from items 1 to 3; the formulas in the descriptions of the A_i have to be constructed using only these symbols and the respective operator's parameters. We further require that all the symbols are distinct. In particular, this forbids using a type also as an F_j and using an object also as an A_i.

The purpose of the *Object* type is to serve as a dummy whenever an action argument, predicate argument or object is not required to be of any specific type. All objects o_i are implicitly of this type; *Object* is a super-type of all other types. Therefore, it is not allowed to appear anywhere in an "either" statement.

In the case of closed-world planning, the initial state description I is simply given by a finite set of ground fluent atoms $F(\vec{r})$; the truth value of non-appearing atoms is assumed to be FALSE. When we are doing open-world planning, I is defined by a finite set of ground atoms $F(\vec{r})$ and negated ground atoms $\neg F(\vec{r})$ (literals); non-appearing atoms are assumed to have an initially unknown truth value (I is a *belief state*, i.e. a representation of a *set* of possible world states).

[6] τ_i is to be understood as the union of the τ_{i_j}. For simplicity, we assume that these definitions do not contain cycles, although one can think of examples where this would make sense (e.g. defining two types as equal).

Example

For illustration, let's consider a variant of Pednault's well-known briefcase example (Pednault 1988; Ghallab *et al.* 1998) dealing with transporting objects between home and work using a briefcase. We have the following problem description:

1. Types:

   Object, *Item*, *Location*

   The *Object* type is the general superclass introduced above. *Item*s are objects that may be transported. A *Location* is a place where we may move objects to.

2. Predicates with associated types of arguments:

   $At(\textit{Item}, \textit{Location}), In(\textit{Item})$

   $At(x_1, x_2)$ denotes that item x_1 currently is at location x_2, $In(x_1)$ means that x_1 is in the briefcase.

3. Objects with associated types:

   briefcase:*Item*, *paycheck*:*Item*, *dictionary*:*Item*
   office:*Location*, *home*:*Location*

4. Operators:

   moveB =
   $(\langle y_1$:*Location*, y_2:*Location*$\rangle$,
   $At(\textit{briefcase}, y_1) \wedge \neg(y_1 = y_2)$,
   $At(\textit{briefcase}, y_2) \wedge \neg At(\textit{briefcase}, y_1) \wedge$
   $\forall z$:*Item*. $In(z) \Rightarrow At(z, y_2) \wedge \neg At(z, y_1))$

   The briefcase can be moved from y_1 to y_2 if it is at the starting location y_1 and y_1 is not identical to the destination y_2. After moving, the briefcase is at the destination and no longer at the starting location, which equally holds for everything that is in the briefcase.

   putInB =
   $(\langle y_1$:*Item*, y_2:*Location*$\rangle$,
   $\neg(y_1 = \textit{briefcase})$,
   $At(y_1, y_2) \wedge At(\textit{briefcase}, y_2) \Rightarrow In(y_1))$

   This operator allows to put something into the briefcase, if it is not the briefcase itself. When applied to an object that is not at the same location as the briefcase, the action has no effect.

   takeOutOfB =
   $(\langle y_1$:*Item*$\rangle, In(y_1), \neg In(y_1))$

   Something is removed from the briefcase.

   emptyB =
   $(\langle\rangle$, TRUE, $\forall z$:*Item*. $In(z) \Rightarrow \neg In(z))$

   Everything is removed from the briefcase.

5. Initial State (in a closed world):

   $I = \{At(\textit{briefcase}, \textit{home}), At(\textit{paycheck}, \textit{home}),$
   $At(\textit{dictionary}, \textit{home}), In(\textit{paycheck})\}$

6. Goal description:

   $G = At(\textit{briefcase}, \textit{office}) \wedge At(\textit{dictionary}, \textit{office}) \wedge$
   $At(\textit{paycheck}, \textit{home})$

Mapping ADL to $\mathcal{ES}$

In this section, we generalize the approach of (Lin & Reiter 1997) for STRIPS to show that applying ADL operators can as well be expressed as a certain form of first-order progression in the situation calculus $\mathcal{ES}$. Below, we will construct, given an ADL problem description in normal form, a corresponding basic action theory Σ. The restriction to normal-form ADL is no loss of generality, as the following theorem shows.

Theorem 1 *The operators of an ADL problem description can always be transformed into an equivalent normal form.*

Here are the operators from the example, put into normal form:

moveB =
$(\langle y_1$:*Location*, y_2:*Location*$\rangle$,
$At(\textit{briefcase}, y_1) \wedge \neg(y_1 = y_2)$,
$\forall x_1$:*Item*. $\forall x_2$:*Location*.
$((x_1 = \textit{briefcase} \vee In(x_1)) \wedge x_2 = y_2)$
$\Rightarrow At(x_1, x_2) \wedge$
$\forall x_1$:*Item*. $\forall x_2$:*Location*.
$((x_1 = \textit{briefcase} \vee In(x_1)) \wedge x_2 = y_1)$
$\Rightarrow \neg At(x_1, x_2))$

putInB =
$(\langle y_1$:*Item*, y_2:*Location*$\rangle$,
$\neg(y_1 = \textit{briefcase})$,
$\forall x_1$:*Item*. $At(x_1, y_2) \wedge At(\textit{briefcase}, y_2) \Rightarrow In(x_1))$

takeOutOfB =
$(\langle y_1$:*Item*$\rangle$,
$In(y_1)$,
$\forall x_1$:*Item*. $(x_1 = y_1) \Rightarrow \neg In(x_1))$

emptyB =
$(\langle\rangle$,
TRUE,
$\forall x_1$:*Item*. $In(x_1) \Rightarrow \neg In(x_1))$

The Successor State Axioms Σ_{post}

It is not a coincidence that the normal form (1) resembles Reiter's (1991) normal form effect axioms which are combined out of individual positive and negative effects and which he then uses to construct his successor state axioms as a solution to the frame problem. Generalizing his approach (also applied in (Pednault 1994)), we transform a set of operator descriptions to a set of successor state axioms as follows, assuming (without loss of generality by Theorem 1) that all operators are given in normal form. Let

$$\gamma^+_{F_j} \stackrel{def}{=} \bigvee_{\gamma_{F_j,A_i} \in NF(A_i)} \exists \vec{y_i}.a = A_i(\vec{y_i}) \wedge \gamma^+_{F_j,A_i} \qquad (3)$$

By "$\gamma_{F_j,A_i} \in NF(A_i)$" we mean that there only is a disjunct for A_i, $1 \leq i \leq m$ if there really exists a γ_{F_j,A_i} in the normal form of the effect of A_i. Recall that the normal form did not require that there is a $\forall \vec{x_j}$:$\tau_{\vec{F_j}}$. $(\gamma^+_{F_j,A}(\vec{x_j}) \Rightarrow$

$F_j(\vec{x_j})$) for every F_j of the domain. We only obtain ones for F_j that did already appear[7] positively in the original effect of A.

Using a similar definition for $\gamma^-_{F_j}$, we get the successor state axiom for F_j:

$$\Box[a]F_j(\vec{x_j}) \equiv \gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j} \quad (4)$$

Differing from the usual construction, we introduced the conjunct $\tau_{\vec{F_j}}(\vec{x_j})$ to ensure that F_j can only become true for instantiations of the $\vec{x_j}$ that are consistent with the type definitions for F_j's arguments.

For each type τ_i, we additionally include a successor state axiom

$$\Box[a]\tau_i(x) \equiv \tau_i(x) \quad (5)$$

to define it as a situation-independent predicate (recall that by the definition of the semantics, all predicates are initially assumed to be fluent).

In the example, we get

$$\gamma^+_{At} = \exists y_1.\exists y_2.a = \mathit{moveB}(y_1,y_2) \wedge ((x_1 = \mathit{briefcase} \vee \mathit{In}(x_1)) \wedge x_2 = y_2) \quad (6)$$

$$\gamma^-_{At} = \exists y_1.\exists y_2.a = \mathit{moveB}(y_1,y_2) \wedge ((x_1 = \mathit{briefcase} \vee \mathit{In}(x_1)) \wedge x_2 = y_1) \quad (7)$$

$$\gamma^+_{In} = \exists y_1.\exists y_2.a = \mathit{putInB}(y_1,y_2) \wedge (\mathit{At}(x_1,y_2) \wedge \mathit{At}(\mathit{briefcase},y_2)) \quad (8)$$

$$\gamma^-_{In} = \exists y_1.a = \mathit{takeOutOfB}(y_1) \wedge (x_1 = y_2) \vee a = \mathit{emptyB} \wedge \mathit{In}(x_1) \quad (9)$$

Notice that, as stated above, not all operators are mentioned in γ^+_{At}, but only those that possibly cause a positive truth value for *At*. Therefore, the construction presented here still incorporates a solution to the frame problem. Our Σ_{post} now consists of the following sentences:

$\Box[a]\mathit{At}(x_1,x_2) \equiv \gamma^+_{At} \wedge \mathit{Item}(x_1) \wedge \mathit{Location}(x_2) \vee \mathit{At}(x_1,x_2) \wedge \neg\gamma^-_{At}$

$\Box[a]\mathit{In}(x_1) \equiv \gamma^+_{In} \wedge \mathit{Item}(x_1) \vee \mathit{In}(x_1) \wedge \neg\gamma^-_{In}$

$\Box[a]\mathit{Object}(x) \equiv \mathit{Object}(x)$

$\Box[a]\mathit{Item}(x) \equiv \mathit{Item}(x)$

$\Box[a]\mathit{Location}(x) \equiv \mathit{Location}(x)$

The Precondition Axiom Σ_{pre}

Further, a precondition axiom can be obtained in a similar fashion, that is a case distinction for all operators of the problem domain:

$$\pi \stackrel{def}{=} \bigvee_{1\le i\le m} \exists \vec{y_i}{:}\vec{\tau_i}.a = A_i(\vec{y_i}) \wedge \pi_{A_i} \quad (10)$$

The types $\vec{\tau_i}$ are those stated in the parameter list of A_i, and π_{A_i} simply is the unmodified precondition for the operator A_i. In our example, we obtain:

$$\begin{aligned}\pi = {} & \exists y_1{:}\mathit{Location}.\exists y_2{:}\mathit{Location}.a = \mathit{moveB}(y_1,y_2) \wedge \\ & \mathit{At}(\mathit{briefcase},y_1) \wedge \neg(y_1 = y_2) \vee \\ & \exists y_1{:}\mathit{Item}.\exists y_2{:}\mathit{Location}. \quad a = \mathit{putInB}(y_1,y_2) \wedge \\ & \neg(y_1 = \mathit{briefcase}) \vee \\ & \exists y_1{:}\mathit{Item}. \quad a = \mathit{takeOutOfB}(y_1) \wedge \\ & \mathit{In}(y_1) \vee \\ & \quad a = \mathit{emptyB} \wedge \\ & \text{TRUE}\end{aligned} \quad (11)$$

The Initial Description Σ_0

Finally, we are left with defining the initial description Σ_0. Here, we not only have to encode the information about the initial state of the world, but also everything that is concerned with the typing of objects. For all "either" statements of the form (2), we need a sentence

$$\tau_i(x) \equiv \tau_{i_1}(x) \vee \cdots \vee \tau_{i_{k_i}}(x) \quad (12)$$

in Σ_0. Further, we include

$$F_j(x_{j_1},\ldots,x_{j_{k_j}}) \supset \tau_{j_1}(x_{j_1}) \wedge \cdots \wedge \tau_{j_{k_j}}(x_{j_{k_j}}) \quad (13)$$

for each type declaration of predicate arguments. Next, for each primitive type τ_i such that $o_{j_1},\ldots,o_{j_{k_i}}$ are all objects declared to be of type τ_i, we include the sentence

$$\tau_i(x) \equiv x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}} \quad (14)$$

The final sentence needed for translating the type definitions is

$$\mathit{Object}(x) \equiv \tau_1(x) \vee \cdots \vee \tau_l(x) \quad (15)$$

Although the above sentences in themselves only establish type consistency in the initial situation (there are no $\Box$ operators here), the special form of Σ_{post} defined earlier ensures that these facts will remain true in successor situations. More precisely, we have here an example where state constraints are resolved by compiling them into successor state axioms (Lin & Reiter 1994).

We now come to the encoding of the actual initial world state. In the case of a closed world, we include for each F_j the sentence

$$F_j(\vec{x_j}) \equiv \vec{x_j} = \vec{o_1} \vee \cdots \vee \vec{x_j} = \vec{o_{k_o}} \quad (16)$$

assuming that $F_j(\vec{o_1}),\ldots,F_j(\vec{o_{k_o}})$ are all the atoms in I mentioning F_j. If we are however dealing with an open-world problem, we instead include the sentence

$$\vec{x_j} = \vec{o_1} \vee \cdots \vee \vec{x_j} = \vec{o_{k_o}} \supset F_j(\vec{x_j}), \quad (17)$$

where $F_j(\vec{o_1}),\ldots,F_j(\vec{o_{k_j}})$ are all the positive literals in I using F_j; and the sentence

$$\vec{x_j} = \vec{o_1} \vee \cdots \vee \vec{x_j} = \vec{o_{k_o}} \supset \neg F_j(\vec{x_j}) \quad (18)$$

when $\neg F_j(\vec{o_1}),\ldots,\neg F_j(\vec{o_{k_o}})$ are all the negative literals in I using F_j.

[7] more precisely for those F_j appearing positively in ϵ_A outside of the antecedent γ of a conditional effect $\gamma \Rightarrow \psi$

In our closed-world example, we end up with a Σ_0 consisting of:

$$\begin{aligned}
\mathit{At}(x_1,x_2) &\supset (\mathit{Item}(x_1) \wedge \mathit{Location}(x_2))\\
\mathit{In}(x_1) &\supset \mathit{Item}(x_1)\\
\mathit{Item}(x) &\equiv ((x=\mathit{briefcase}) \vee (x=\mathit{paycheck}) \vee\\
&\quad (x=\mathit{dictionary}))\\
\mathit{Location}(x) &\equiv ((x=\mathit{office}) \vee (x=\mathit{home}))\\
\mathit{Object}(x) &\equiv (\mathit{Item}(x) \vee \mathit{Location}(x))\\
\mathit{At}(x_1,x_2) &\equiv ((x_1=\mathit{briefcase} \wedge x_2=\mathit{home}) \vee\\
&\quad (x_1=\mathit{paycheck} \wedge x_2=\mathit{home}) \vee\\
&\quad (x_1=\mathit{dictionary} \wedge x_2=\mathit{home}))\\
\mathit{In}(x_1) &\equiv (x_1=\mathit{paycheck})
\end{aligned}$$

Correctness

Finally, we will show the correspondence between the state-transitional semantics for ADL of adding and deleting literals and first-order progression in $\mathcal{ES}$. The following definition is derived from Lin and Reiter's original proposal of progression, but is simpler due to the fact that we do not need to consider arbitrary first-order structures.

A set of sentences Σ_r is a *progression* of Σ_0 through a ground term r (wrt Σ_{pre} and Σ_{post}) iff:

1. all sentences in Σ_r are fluent in $\langle r\rangle$;
2. $\Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \models \Sigma_r$;
3. for every world w_r with $w_r \models \Sigma_r \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, there is a world w with $w \models \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$ such that:

   $$w_r, r\cdot\sigma \models F(\vec{t}) \quad \text{iff} \quad w, r\cdot\sigma \models F(\vec{t})$$

   for all $\sigma \in R^*$ and all primitive formulas $F(\vec{t})$ such that $F \in \mathcal{F}$ (including *Poss*).

A formula that is *fluent in* $\langle r\rangle$ is one which is equivalent to $[r]\phi$ for some fluent[8] formula ϕ, i.e. it only talks about the fluents' values in the situation $\langle r\rangle$. Intuitively, for an observer standing in the situation after r was performed (and only looking "forward" in time), it is impossible to distinguish between a world w satisfying the original theory Σ and a world w_r that satisfies $\Sigma_r \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$.

We now want to address the issue of how to obtain such a progression. The result will be that for a basic action theory that is a translation from an ADL problem (and therefore the member of a restricted subclass of the general form of action theories), it is quite easy to construct such a set. Given an ADL problem description and an action $A(\vec{p})$ (i.e. an operator and object symbols as instantiations for A's parameters), we make, under the condition that $\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{Poss}(A(\vec{p}))$, the following modifications to the state description I:

- in the case of closed-world planning:
  - for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent
    if $\Sigma_0 \models \gamma^{+\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ A(\vec{p})}$: add $F_j(\vec{o})$
  - for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent
    $\Sigma_0 \models \gamma^{-\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ A(\vec{p})}$: delete $F_j(\vec{o})$
- in the case of open-world planning:
  - for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent
    $\Sigma_0 \models \gamma^{+\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ A(\vec{p})}$: add $F_j(\vec{o})$, delete $\neg F_j(\vec{o})$
  - for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent
    $\Sigma_0 \models \gamma^{-\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ A(\vec{p})}$: add $\neg F_j(\vec{o})$, delete $F_j(\vec{o})$

If we denote the set of literals to be added by $Adds$ and the ones to be deleted by $Dels$, then the new state description is

$$I' = (I \cup Adds) \setminus Dels.$$

Here it is assumed that we only consider symbols (objects, fluents, operators) that appear in the given problem description, which yields only finitely many combinations. The fact that we only have to check type-consistent atoms further restricts the number of atoms to be treated. Formally, $F_j(\vec{o})$ being *type-consistent* means that $\Sigma_0 \cup \{\tau_{\vec{F_j}}(\vec{o})\}$ is satisfiable.

Theorem 2 *Let I' be the set obtained in the above construction applied to a given (closed- or open-world) ADL problem description and a ground action $r = A(\vec{p})$. Further let*

$$\Sigma_r = \{[r]\psi \mid \psi \in \Sigma_0(I')\},$$

where $\Sigma_0(I')$ means the result of applying the constructions in (12)-(18) to I' instead of I. For all fluent predicates F_j in the problem description, let the consistency condition

$$\models (\gamma^+_{F_j} \wedge \gamma^-_{F_j})^a_r$$

hold. Then Σ_r is a progression of Σ_0 through r

- *in the closed-world case;*
- *in the open-world case only under the additional condition that whenever for some $\gamma^*_{F_j}$ (where $* \in \{+,-\}$) it holds that $\Sigma_0 \cup \{\gamma^{*\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ r}\}$ is satisfiable, then*

$$\Sigma_0 \models \gamma^{*\ \vec{x_j}\ a}_{F_j\ \vec{o}\ \ r}.$$

For space reasons, we will not present a proof here. The main reason for being able to establish the result lies in the finiteness of the domain to be considered. Whereas $\mathcal{ES}$'s semantics assumes a domain with countably infinite many objects and actions, PDDL, as a language that is used in practical implementations, only allows problem domains with a finite number of operators and items. We utilize the typing constructs to reconcile these two views.

The additional condition for open-world problems can be illustrated with a small example: consider an operator $A = (\emptyset, \text{TRUE}, P \Rightarrow Q_1 \wedge \neg P \Rightarrow Q_2)$ and an open world initial state description of $I = \emptyset$. Applying A to I leads to a situation whose state is described by $Q_1 \vee Q_2$, since it is both possible that P holds and that it does not hold. Obviously, the resulting state is not representable by a set of literals,

[8] Recall that our terminology contains both the notions of fluent predicates (like *At*) as well as that of fluent formulas (e.g. $\exists x.\mathit{At}(x,\mathit{home}) \wedge \mathit{In}(x)$); they should not be confused.

therefore we cannot apply the above progression scheme. In fact, mainly because of such undefined states, the open-world requirement is not included in the PDDL language definition anymore since version 2.1 (Fox & Long 2003), restricting its application to purely closed-world planning.

Notice that in the closed-word case, our special form of basic action theories constitutes a proper subclass of what is called "relatively complete databases" in (Lin & Reiter 1997). It is therefore not surprising that a progression of such theories exists. Theorem 2 however additionally establishes that our class of action theories is also *closed under progression*, since the progression result Σ_r is of the same form as the original Σ_0. Progression steps may thus be applied iteratively.

On the other hand, in both the open- and closed-world case, Lin and Reiter's progression for "strongly context-free" theories (for which they show the correspondence to STRIPS) is a special case of our result. This agrees with what one would expect from the fact that ADL action descriptions can be viewed as a generalization of STRIPS operators.

Now let us return to our example again to see how the closed-world progression works in this case. We assume that we want to progress through the action $\mathit{moveB}(\mathit{home}, \mathit{office})$ (abbreviated as m). The first thing to notice is that

$$\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{Poss}(m)$$

iff, using (11), unique names for actions and the fact that *home* and *office* are both *Location*s,

$$\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{At}(\mathit{briefcase}, \mathit{home}) \wedge \neg(\mathit{home} = \mathit{office})$$

iff, with unique names for objects (recall that our semantics does not distinguish between objects and actions)

$$\Sigma_0 \cup \Sigma_{\text{pre}} \models \mathit{At}(\mathit{briefcase}, \mathit{home})$$

which is obviously the case, therefore we may proceed. The reader may verify (considering (6) and (7)) that

- $\Sigma_0 \models \gamma^{+}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{briefcase}}{}^{x_2}_{\mathit{office}}$
- $\Sigma_0 \models \gamma^{+}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{paycheck}}{}^{x_2}_{\mathit{office}}$
- $\Sigma_0 \models \gamma^{-}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{briefcase}}{}^{x_2}_{\mathit{home}}$
- $\Sigma_0 \models \gamma^{-}_{\mathit{At}}{}^{a}_{m}{}^{x_1}_{\mathit{paycheck}}{}^{x_2}_{\mathit{home}}$

and that these are all type-consistent instantiations for x_1, x_2 such that $\gamma^{+}_{\mathit{At}}{}^{a}_{m}$ respectively $\gamma^{-}_{\mathit{At}}{}^{a}_{m}$ are entailed by Σ_0. Because there are no disjuncts for *moveB* in (8) and (9), $\gamma^{+}_{\mathit{In}}{}^{a}_{m}$ and $\gamma^{-}_{\mathit{In}}{}^{a}_{m}$ are not entailed for any instantiation of x_1. The new initial state then is

$$I' = \{\mathit{At}(\mathit{dictionary}, \mathit{home}),\ \mathit{In}(\mathit{paycheck}),\ \mathit{At}(\mathit{briefcase}, \mathit{office}),\ \mathit{At}(\mathit{paycheck}, \mathit{office})\}.$$

We obtain the progression Σ_m consisting of

$$\begin{array}{ll}
[m](\mathit{At}(x_1, x_2) & \supset (\mathit{Item}(x_1) \wedge \mathit{Location}(x_2))) \\
[m](\mathit{In}(x_1) & \supset \mathit{Item}(x_1)) \\
[m](\mathit{Item}(x) & \equiv ((x = \mathit{briefcase}) \vee (x = \mathit{paycheck}) \vee \\
 & \quad (x = \mathit{dictionary}))) \\
[m](\mathit{Location}(x) & \equiv ((x = \mathit{office}) \vee (x = \mathit{home}))) \\
[m](\mathit{Object}(x) & \equiv (\mathit{Item}(x) \vee \mathit{Location}(x))) \\
[m](\mathit{At}(x_1, x_2) & \equiv ((x_1 = \mathit{dictionary} \wedge x_2 = \mathit{home}) \vee \\
 & \quad (x_1 = \mathit{briefcase} \wedge x_2 = \mathit{office}) \vee \\
 & \quad (x_1 = \mathit{paycheck} \wedge x_2 = \mathit{office}))) \\
[m](\mathit{In}(x_1) & \equiv (x_1 = \mathit{paycheck}))
\end{array}$$

Notice that the only changes, compared to Σ_0, are the "$[m]$" in front of each formula (to denote the situation after m has been performed) and the new instances for *At*.

Outlook: Embedding ADL planning in Golog

The situation calculus (and, as (Lakemeyer & Levesque 2005) showed, also $\mathcal{ES}$) constitutes the foundation[9] on which the semantics of the agent programming language Golog (Levesque *et al.* 1997) is defined. The language gives a programmer the freedom to on the one hand specify the agent's behaviour only roughly by using nondeterministic constructs and where it is the system's task to find a deterministic strategy to achieve its goal. On the other hand, the programmer may utilize deterministic constructs known from imperative programming languages. Nonetheless there is some drawback with this general purpose approach which can be illustrated best by considering the following completely nondeterministic Golog program:

$$\mathit{achieve}(\mathit{Goal}) := \textbf{while}\ (\neg \mathit{Goal})\ \textbf{do}\ (\pi a)\ a\ \textbf{endWhile}$$

The program corresponds to the task description of finding sequential plans: As long as the condition Goal is not fulfilled, nondeterministically pick some action a and execute it. Although it is thus possible to do sequential planning in Golog, the performance of the Golog system can usually not compete with current state-of-the-art planners like FF (Hoffmann & Nebel 2001; Hoffmann 2003; Hoffmann & Brafman 2005), LPG (Gerevini *et al.* 2005), HSP2 (Bonet & Geffner 2001a), Fast Downward (Helmert & Richter 2004) or TLPlan (Bacchus & Ady 2001). The reason is that current Golog implementations resolve nondeterminism by a simple backtracking mechanism, whereas planners resort to efficient techniques like heuristic search, e.g. (Bonet & Geffner 2001b).

The idea now is, using the results presented here, to embed ADL-based planners (more precisely planners that take the ADL subset of PDDL as an input language) into Golog, to combine the benefits of both systems. We envision that whenever, during the execution of a Golog program, a planning problem arises (i.e. an $\mathit{achieve}(G)$ subgoal has to be solved), the necessary parts of the current situation and the subgoal are transformed into a planning problem instance and handed over to the planner. Once a solution (a sequence of actions) is found, it is transformed back into Golog and the program execution continues, where PDDL serves in

[9] Valid executions of Golog programs are expressed by a situation calculus (respectively $\mathcal{ES}$) formula that is entailed by the underlying basic action theory.

both cases as an interface language. The results in this paper show that, as long as the action theory underlying the Golog program is obtained by a translation from an ADL problem description, this method is semantically well-founded.

Conclusion

We presented an alternative definition for the semantics of ADL operators as progression in first-order $\mathcal{ES}$ knowledge bases. This establishes the basis for embedding existing state-of-the-art planners that take ADL as an input language, into an interpreter for the robot programming language Golog, to obtain a powerful language that is equally suited for autonomously constructing complete plans (utilizing the planner) and letting the programmer specify preconstructed plans with residual nondeterminism (by means of the usual Golog constructs) to be resolved by the system. Such an embedding into an $\mathcal{ES}$-based Golog interpreter (currently under development) is the focus of future work, as well as giving semantics to larger fragments of PDDL (Edelkamp & Hoffmann 2004; Gerevini & Long 2005b) with features such as numeric fluents, time, or preferences.

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *Proc. IJCAI-2001*, 417–424.

Bonet, B., and Geffner, H. 2001a. Heuristic Search Planner 2.0. *AI Magazine* 22(3):77–80.

Bonet, B., and Geffner, H. 2001b. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Universität Freiburg.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Gerevini, A., and Long, D. 2005a. BNF description of PDDL3.0. `http://zeus.ing.unibs.it/ipc-5/bnf.pdf`.

Gerevini, A., and Long, D. 2005b. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Italy.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2005. Planning with derived predicates through rule-action graphs and relaxed-plan heuristics. Technical report, Universita degli Studi di Brescia, Dipartimento di Elettronica per l Automazione, Brescia, Italy.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language. `ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`.

Helmert, M., and Richter, S. 2004. Fast Downward – making use of causal dependencies in the problem representation. `http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/Proc/downward.pdf`.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS-05*, 71–80.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Lakemeyer, G., and Levesque, H. J. 2004. Situations, si! situation terms, no! In *Proc. KR2004*. AAAI Press.

Lakemeyer, G., and Levesque, H. J. 2005. Semantics for a useful fragment of the situation calculus. In *Proc. IJCAI-05*.

Levesque, H. J., and Lakemeyer, G. 2001. *The Logic of Knowledge Bases*. MIT Press.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Lifschitz, V. 1986. On the semantics of STRIPS. In Georgeff, M. P., and Lansky, A. L., eds., *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, 1–9. Timberline, Oregon: Morgan Kaufmann.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of Logic and Computation* 4(5):655–678.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artif. Intell.* 92(1-2):131–167.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res. (JAIR)* 12:271–315.

Pednault, E. P. D. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4:356–372.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the Situation Calculus. In *Proc. KR1989*, 324–332. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pednault, E. P. D. 1994. ADL and the state-transition model of action. *J. Log. Comput.* 4(5):467–512.

Reiter, R. 1991. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy* 359–380.

Reiter, R. 2001. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. Cambridge, Mass.: MIT Press. The frame problem and the situation calculus.

4.4 Planning ramifications: When ramifications are the norm, not the 'problem'

Planning ramifications: When ramifications are the norm, not the 'problem'

Debora Field
Dept. of Computer Science, University of Liverpool
Liverpool L69 3BX, UK
debora@csc.liv.ac.uk

Allan Ramsay
School of Informatics, University of Manchester
PO Box 88, Manchester M60 1QD, UK
allan.ramsay@manchester.ac.uk

Abstract

This paper discusses the design of a planner whose intended application required us to solve the so-called 'ramification problem'. The planner was designed for the purpose of planning communicative actions, whose effects are famously unknowable and unobservable by the doer/speaker, and depend on the beliefs of and inferences made by the observer/hearer. Our fully implemented model can achieve goals that do not match action effects, but that are rather entailed by them, which it does by reasoning about how to act: state-space planning is interwoven with theorem proving in such a way that a theorem prover uses the effects of actions as hypotheses.[1]

Introduction

Seeing the word 'ramification' so often bound to the word 'problem', it is easy to get the impression from the literature that the ramifications of actions are viewed by the AI planning community as an annoying hindrance to their AI planning ambitions. We, however, see ramifications very differently. They are the focus of our planning ambition and the mechanism of its success. Why? Because we are interested in modelling an every-day human activity which is totally dependent upon the ramifications of actions: human-to-human communication.

As far as communication is concerned, each man (and woman) is an island. I have things I want you to believe, and to this end I do my best to make appropriate signs to you—in writing, speech, smoke signals, facial gestures, and so on. You see my signs, and you decide for yourself what they mean. There is nothing I can do to ensure that you receive the message I want you to get. All I can do is make my signs, and put my trust in the ramifications of my actions.

Consider the human, John. Imagine John's current goal is to get human Sally to believe the proposition *John is kind*. John has no direct access to the environment he wishes to affect—he cannot simply implant *John is kind* into Sally's belief state. John knows that Sally has desires and opinions of her own, and that he will have to plan something that he considers might well lead Sally to infer *John is kind*. This means that when John is planning his action—whether to give Sally some chocolate, pay her a compliment, tell her he is kind, lend her his credit card—he has to consider the many different messages Sally might infer from the one thing John chooses to say or do.

To plan communicative acts is, then, to plan actions by taking into account their possible ramifications. How do we do this? We took a backward-chaining theorem prover, and adapted it for hypothetical reasoning about the effects of actions. Our backward-chaining reasoner essentially says, "I could prove this backwards if you allowed me to introduce these hypotheses". The fully implemented model is thus able to plan to achieve goals that do not match action effects, but that are entailed by them. Our planner was developed by first adapting (Manthey & Bry 1988)'s first-order theorem prover, Satchmo, into a theorem prover for a highly intensional logic (Ramsay 2001), namely, a constructive version of property theory (Turner 1987). To this was added a deduction theory of knowledge and belief (Konolige 1986) so that the planner can reason with its beliefs about the world, including its beliefs about others' beliefs. State-space planning (based on foundational work in classical planning (Newell, Shaw, & Simon 1957; Newell & Simon 1963; Green 1969; McCarthy & Hayes 1969; Fikes & Nilsson 1971)) was then interwoven with theorem proving in such a way as to enable planning for entailed goals.

Satchmo

The theorem prover we present was developed by extending Manthey and Bry's (1988) first-order theorem prover, Satchmo (SATisfiability CHecking by MOdel generation). For model generation we convert the standard form to SEQUENT FORM, where a sequent is a formula of the form $\Gamma \Rightarrow \Delta$ where Γ is $\top$, an atomic formula, or a conjunction of atomic formulae, and Δ is $\bot$, an atomic formula, or a disjunction of atomic formulae. Satchmo was designed for carrying out proof by contradiction, where you show that some formula A follows from a set of assumptions α by converting $\alpha \cup \{\neg A\}$ to normal form, and showing that this set

[1]Initially supported by an EPSRC grant, with recent developments partially funded under EU-grant FP6/IST No. 507019 (*PIPS*: Personalised Information Platform for Health and Life Services).

has no models. The goal is to show that the set of sequents obtained from the assumptions α and the negation $\neg A$ of the goal supports a proof of $\perp$, since this means that no model of α can be a model of $\neg A$, and hence that all models of α are models of A (i.e., $\alpha \models A$).

Model generation proceeds by distinguishing between HORN SEQUENTS, and DISJUNCTIVE SEQUENTS. A Horn sequent has a single literal as its consequent; a disjunctive sequent has a disjunction as its consequent. The proof proceeds in two stages:

(MG-i) Try to prove $\neg A$ by backward chaining among the Horn sequents.

(MG-ii) If this fails, find a disjunctive sequent whose antecedent can be proved using the Horn sequents, but whose consequent cannot. Add each disjunct from the consequent in turn, and try again. The point here is that if we know Γ and $\Gamma \Rightarrow A_1 \vee A_2$, then we know that either A_1 or A_2 must hold. So if we can show that $\perp$ follows from what we know already plus either of A_1 or A_2, then we know that $\perp$ actually follows from what we know already.

A constructive epistemic intensional logic

The original presentation of Satchmo is unsuitable for our purposes, since it assumes a classical version of predicate logic, whereas we use a constructive epistemic logic.

An epistemic logic is necessary because the program needs to be able to do reasoning about what agents believe, including what they believe about the beliefs of other agents. The theorem prover embodies a deduction model of belief (Konolige 1986), rather than a 'possible worlds' model (Hintikka 1962; Kripke 1963). In a deduction model, agents are allowed to have sets of beliefs that are incomplete, inaccurate, and inconsistent, to have imperfect inference strategies, and to *do* inference—which is much more useful for modelling human epistemic reasoning than an agent who is logically omniscient and automatically knows everything that follows from his belief set.

We use a constructive logic, because we consider it is essential for our purpose—to model natural language—as we will now briefly argue. Our argument falls under two themes: (i) We need a theorem prover that can reason with formulae in which propositions are quantified over; (ii) We need a theorem prover that more closely models the way *people* do reasoning, including reasoning about a proposition that a person believes to be false.

A theorem prover for property theory

There are a number of phenomena in natural language that we can cope with if we are allowed to quantify over propositions, but that seem otherwise very hard to capture. Consider the word '*only*'. If S says to H:

I only <u>touched</u> it

(with a voiced stress on *touched*), S is inviting H to compare the action A that S *did* do (*touched*) with some other action A' that S did *not* do, and which is in some way stronger than A (*broke*, perhaps). To represent the meaning of '*only*' we require intensional meaning postulates—intensional, because they comment on the truth or falsity of other parts of the sentence. For '*only*' we need something like:

$$\forall P \forall X(only(P, X) \Rightarrow (P.X \& \exists P'(similar(P, P') \& \neg P'.X)))$$

This axiom says that if $only(P, X)$ holds, where P is some property and X is an arbitrary entity (possibly a property itself), then P does hold of X, but there is some property P' which is similar to P, and which you might have expected to hold of X, but which in fact fails to do so. So then, to adequately represent the meaning of even a common little word like '*only*', not to mention many other natural language constructions, we need to be able to quantify over propositions (Ramsay 1994).

Quantifying over propositions, however, opens the way to the paradoxes of negative self-reference—Russell's set, the Epimenedes paradox, Grelling's paradox, and so on. Therefore, we need some way of preventing this that still admits common natural language usage (e.g., the many self-referring pronouns: '*this*', '*me*', '*himself*' ...), which the earlier classical solutions to this problem do not (e.g., Whitehead and Russell's (1910) THEORY OF TYPES, and MONTAGUE SEMANTICS (Thomason 1974)).

Turner's PROPERTY THEORY (1987) provides a solution to this problem by allowing you to say whatever you want, but then placing constraints on what can be proved. Turner's analysis involves taking a classical treatment of first-order logic, and adding λ-abstraction and β-reduction to it (or at any rate, operations which look extremely like λ-abstraction and β-reduction). It allows propositions and properties to occur as arguments, and thus provides the expressive power we need. It provides an operator for constructing terms corresponding to propositions and properties, so that they can appear in the positions where terms are required (i.e., as predicate arguments), together with a series of predicates for retrieving the truth conditions of such a term. The main advantage of Property Theory over languages like Montague Semantics is that it is a type-free language, and hence provides considerable extra expressive power. Spelling out the meaning of a word like '*only*' requires you to produce *untyped* intensional meaning postulates—untyped because this word can apply to phrases of (almost) any syntactic type, so that their semantic analysis must apply to phrases of almost any semantic type.

Turner's Property Theory is, then, attractive for our natural language purposes. Unfortunately, (Herzberger 1982) shows that it is impossible to provide a normal form for

Turner's version of the logic, and hence it is very difficult to implement a theorem prover for it. Our solution to this is to take a constructive treatment of first-order logic, allow unrestricted use of both λ-abstraction and β-reduction, and avoid the paradoxes by placing constraints on the assumptions that can be used in a well-founded proof (Ramsay 2001).

Modelling human reasoning

The way humans do every-day reasoning is, we consider, quite different from the way reasoning is handled under classical logic. In classical logic, for example, and using our general knowledge, we judge the following formulae to be true:

(1) Earth has one moon ⇒ Elvis is dead
(2) Earth has two moons ⇒ Elvis is alive
(3) Earth has two moons ⇒ Elvis is dead

(1) is true simply because antecedent and consequent are both true formulae. We find this truth odd, however, because of the absence of any discernible relationship between antecedent and consequent. (2) and (3) are true simply because the antecedent is false, which seems very counter-intuitive. Even more peculiarly, the following formula is provable in classical logic in all circumstances:

(4) (Earth has one moon ⇒ Elvis is dead) or
(Elvis is dead ⇒ Earth has one moon)

but it feels very uncomfortable to say that it must be the case that one of these implies the other.

In order to avoid having to admit proofs like this, and to be able to do reasoning in a slightly more human-like way, we choose constructive logic and natural deduction. In order to prove $P \Rightarrow Q$ by natural deduction, one must show that Q is true *when* P is true; if P is not true, constructive logic does not infer $P \Rightarrow Q$. This treatment of implication hints at a relationship between P and Q which is absent from material implication.

Taking a constructive view also allows us to simplify our reasoning about when the hearer believes something of the form $P \Rightarrow Q$, and hence (because of the constructive interpretation of $\neg P$ as $P \Rightarrow \bot$) about whether she believes $\neg P$. We will assume that $believes(H, P)$ means that H *could* infer P on the basis of her belief set, not that she already does believe P, and we will examine the relationship between $believes(H, P \Rightarrow Q)$ and $believes(H, P) \Rightarrow believes(H, Q)$.

Consider first $believes(H, P) \Rightarrow believes(H, Q)$. Under what circumstances could you convince yourself that this held?

For a constructive proof, you would have to assume that $believes(H, P)$ held, and try to prove $believes(H, Q)$. So you would say to yourself 'Suppose I were H, and I believed P. Would I believe Q?' The obvious way to answer this would be to try to prove Q, using your own rules of inference. Suppose you came up with such a proof. Assuming that H's rules of inference were the same as yours, you would then be able to assume that she could also carry out this proof. But if that were the case, then she would have a proof of $P \Rightarrow Q$ available to her, and hence it would be reasonable to conclude $believes(H, P \Rightarrow Q)$.

Suppose, on the other hand, that you believed $believes(H, P \Rightarrow Q)$, and that you also believed $believes(H, P)$. This would mean that you thought that H had both $P \Rightarrow Q$ and P available to her. But if you had these two available to you, you would be able to infer Q, so since H is very similar to you, she should also be able to infer Q. So from $believes(H, P \Rightarrow Q)$ and $believes(H, P)$ we can infer $believes(H, Q)$, or in other words, $(believes(H, P \Rightarrow Q)) \Rightarrow (believes(H, P) \Rightarrow believes(H, Q))$.

We thus see that if we take $believes(H, P)$ to mean 'If I were H I would be able to prove P', then $(believes(H, P \Rightarrow Q))$ and $(believes(H, P) \Rightarrow believes(H, Q))$ are equivalent. This has considerable advantages in terms of theorem proving, since it means that much of the time we can do our reasoning by switching to the believer's point of view and doing perfectly ordinary first-order reasoning. If, in addition, we treat $\neg P$ as a shorthand for $P \Rightarrow \bot$, we see that $believes(H, \neg P)$ is equivalent to $believes(H, P) \Rightarrow believes(H, \bot)$. If we take the further step of assuming that nobody believes $\bot$, we can see that $believes(H, \neg P) \Rightarrow \neg believes(H, P)$ (though not $\neg believes(H, P) \Rightarrow believes(H, \neg P)$). We cannot, however, always assume that everyone's beliefs are consistent, so we may not always want to take this further step (note that in possible worlds treatments, we are *forced* to assume that everyone's beliefs are consistent), but it is useful to be able to use it as a default rule, particularly once we understand the assumptions that lie behind it.

Constructive Satchmo

We have said that the original presentation of Satchmo is unsuitable for our purposes, since it assumes a classical version of predicate logic. This means that you can prove P by showing that $\neg P$ is unsatisfiable, and you can also use equivalences such as $((P \Rightarrow Q) \Rightarrow R) \iff ((Q \Rightarrow R) \& (P \ or \ R))$, which are not available in constructive logic. We therefore need to adapt Satchmo so that it works properly for constructive logic, and so that it can handle epistemic information.

We do this in two stages: first we have to convert our problem into an appropriate normal form, and then we have to adapt the basic Satchmo engine to work constructively with this normal form. Epistemic information is represented by using Wallen's (1987) notion of CONTEXT, which in our case is epistemic context.

The construction of a normal form proceeds in three stages.

(i) We start by making very straightforward textual changes, to make standard logical form look a bit more like Prolog, and to get rid of existential quantifiers.

- NF-1 Replace `(A & B)` by `(A', B')` and `(A or B)` by `(A'; B')`, where `A'` and `B'` are the normal forms of `A` and `B`.
- NF-2 Replace `not(A)` by `(A' ⇒ absurd)`.
- NF-3 Replace `P ⇒ (Q ⇒ R)` by `((P & Q) ⇒ R)'`.
- NF-4 Replace `believes(J,A)` by `A@@[believes(J)]`. (See below for further discussion of epistemic contexts)
- NF-5 Skolemise away existential quantifiers, and remove all universal quantifiers.

(ii) Separate the result of (i) into Horn and non-Horn clauses, and convert the Horn clauses to ordinary Prolog.

- PL-1 If the normal form of `P` is atomic then assert it as a Prolog fact.
- PL-2 If the normal form of `P` is `(Q , R)` then deal with `Q` and `R` individually.
- PL-3 If the normal form of `P` is `(Q ; R)` then assert `split(Q ; R)` as a Prolog fact.
- PL-4 If the normal form of `P` is `(K ⇒ Q)` where `Q` is atomic then assert `Q :- K` as a Prolog rule.
- PL-5 If the normal form of `P` is `(K ⇒ (Q , R))` then deal with `(K ⇒ Q)` and `(K ⇒ R)` individually.
- PL-6 If the normal form of `P` is `(K ⇒ (Q ; R))` then assert `split(Q ; R) :- K` as a Prolog rule.
- PL-7 If the normal form of `P` is `(A ⇒ B) ⇒ C` then assert `C :- (A => B)` as a Prolog rule. See (MG-3) for further discussion of this rule.

(iii) Perform any optimisations that you can on these.

- OP-1 Remove all 'pure literals' (Kowalski 1975) from the clause set, and store for easy restoration if they later become 'impure'.
- OP-2 Satchmo can be made to perform very poorly if you include disjunctive clauses where one or both disjuncts is actually irrelevant, so optimise relevance by banning the use of a `split` clause until it has been shown that its consequents *will* contribute to a proof of `G`. Do this by asserting a rule which will itself add the `split` clause when its consequents have been shown to be relevant.
- OP-3 Include a 'loop checking' procedure which ensures that any loops which might arise in the automatically generated model are checked (see later for how this is done).
- OP-4 Cut out unnecessary multiple proofs of the same goal. If the head of a rule is ground at the point when called, we know there is no point in finding multiple proofs of it, so place a Prolog cut, to be invoked iff the head was ground at the point when the clause was called.

Once we have the problem converted to normal form, we can use the following adaptation of the basic model generation algorithm.

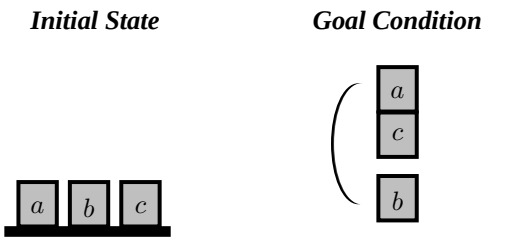
- MG-1 If you can prove `A` by using Prolog facts and rules then you can prove it.
- MG-2 You can prove `A` if you can prove `split(P , Q)` and any of (i) `P ⇒ A` and `Q ⇒ A`, or (ii) `P ⇒ A` and `not(Q)`, or (iii) `not(P)` and `Q ⇒ A`.
- MG-3 To prove `A ⇒ B`, assert `A` and try to prove `B`. If asserting `A` allows you to prove `B` then you have a proof of `A ⇒ B`. Whether or not you succeed in proving `B`, you must retract `A` afterwards.

(MG-1) and (MG-2) are exactly as in the original presentation of Satchmo, except that since Satchmo works by trying to show that the hypotheses + the negation of the goal are unsatisfiable, it always tries to prove `absurd`, whereas a constructive version has to show that the goal itself is provable from the hypotheses (though the second pair of routes through (MG-2) allow you to do this slightly indirectly). (MG-3)[2] is introduced because Satchmo relies on the classical equivalence between $((P \Rightarrow Q) \Rightarrow R)$ and $((Q \Rightarrow R) \& (P \; or \; R))$ when constructing normal forms. This equivalence is no longer available: if we want to prove $P \Rightarrow Q$ we have to use $\Rightarrow$-introduction. In particular, if you want to prove `not(P)`, as you might as a consequence of using (MG-2(ii)) or (MG-2(iii)), then you will have to do this by using (MG-3) to prove `P ⇒ absurd`. There follows a skeletal implementation of this.

```
% You can prove A either directly
prove(A):-
    A.
% or by proving (P or Q),(P => A)and(Q => A)
prove(A):-
    split(P;Q),
    % Check you haven't tried this already
    \+ (P;Q),
    prove(P => (A or absurd)),
    prove(Q => (A or absurd)).
% To prove (P => A), assert P
% and try to prove A (with some funny
% bookkeeping to tidy up after yourself)
(P => A):-
    assert(P),
    (prove(A) -> retract(P);
    (retract(P), fail)).
```

[2] MG-3 is equivalent to $\Rightarrow$-introduction from standard constructive logic. If you start by checking that `B` is not provable without `A`, you get RELEVANT IMPLICATION (Anderson & Belnap jr. 1975).

Figure 1:



The key non-cosmetic differences between this and Satchmo are that (i) this version implements a constructive version of first-order logic rather than a classical one, and (ii) it is slightly more direct when faced with clauses of the form $((P \Rightarrow Q) \Rightarrow R)$. Most of the work in Satchmo is performed in the backward chaining phase where the Prolog facts and rules are being used to prove specific goals. By converting $((P \Rightarrow Q) \Rightarrow R)$ to `R :- (P ⇒ Q)`, we ensure that this rule is activated when it is required, at the cost of having to prove `P ⇒ Q` by asserting `P` and showing that `Q` follows from it. If we convert $((P \Rightarrow Q) \Rightarrow R)$ to `R :- (P ⇒ Q)` and `split(R ; P)`, we end up having to explore the consequences of asserting `P` anyway.

Planning by hypothesising actions

Now that the theorem prover has been introduced, let us move on to how the theorem prover is used to hypothesise actions in order to achieve goals that do not match action effects, but that are entailed by them. First, a 'blocks world' example will be given to illustrate what we mean by 'planning ramifications'. Following this will be an explanation of how information concerning which actions have been hypothesised is carried around in the model. Finally, a description of the model's planning procedure will be given.

An example

Figure 2 shows a list of facts describing a simple knowledge state, which asserts: that j knows (strongly believes) there is a certain configuration of blocks (three, each on the table); that everyone knows about two actions: '*stack*' and '*unstack*'; and that everyone knows that '*above*' is the transitive closure of '*on*'.

The *Initial State* in Figure 1 is a pictorial representation of the knowledge state shown in Figure 2, while the *Goal Condition* in Figure 1 pictures the following proposition:

```
believes(j, above(a,b) & on(a,c))
```

This is the plan solution returned when we call the planner to achieve this proposition from the initial state described by `state_1`:[3]

[3]For clarity's sake, a number of details are missing from Figure 2 which are present in the model, including constraints on the recursive `above(_,_)` rule and on the effects of actions.

Figure 2:

```
state_1(
 believes(j, isblock(a) & isblock(b)
  & isblock(c) & on(a,table) & on(b,table)
  & on(c,table) & clear(a) & clear(b)
  & clear(c))
 & forall(Y,
    believes(Y,
     action(doer(slave),stack(slave,F,G),
      precons(isblock(G) & isblock(F)
       & on(F,table)
       & clear(G) & clear(F)),
      add(on(F,G)),
      delete(on(D,table) & clear(G)))))
 & forall(Y,
    believe(Y,
     action(doer(slave), unstack(slave,F,G),
      precons(on(F,G) & clear(F)
       & isblock(F) & isblock(G)),
      add(clear(G) & on(F,table)),
      delete(on(F,G)))))
 & forall(Y,
    believes(Y,
     (on(P,Q) => above(P,Q))))
 & forall(Y,
    believes(Y,
     ((on(R,Q) & above(P,R))
      => above(P,Q)))).
```

```
[stack(slave,c,b),stack(slave,a,c)]
```

Disregarding epistemic matters for the moment, in order to achieve *above(a,b) & on(a,c)*, something different from an action with an *above(_,_)* expression in its add list is needed (note that although the goal contains a predicate of the form *above(_,_)*, the add list of action *stack* does not contain any *above* predicates). Placing *a* onto *b*, for example, will make *above(a,b)* proveable, but it will also make the achievement of *on(a,c)* impossible. By reasoning with the rules that describe the meaning of *above* as the transitive closure of *on*, the theorem prover hypothesises that the proposition *on(a,X) & on(X,b)* might enable the proof of *above(a,b)* to be completed. It also knows that *on(X,Y)* is an effect of action *stack(X,Y)*. A proof of the preconditions of action *stack(a,X)* is invoked by the planning search, and the process continues (with backtracking), until a full plan solution is found. Thus the planner is able to find *stack* actions to achieve *above* goals, because it knows that the effects of *stack(X,Y)* include *on(X,Y)*, while also knowing that the ramifications of *stack(X,Y)* include *above(X,***Z***)*.

Labels

Now we come to explain how information concerning which actions have been hypothesised is carried around in the model. As a starting point for this, we refer again to the theorem prover's loop-checking mechanism. The programs we are concerned with are generated automatically from sets

of statements, and it is not appropriate to restrict what can be said in this language, just because we are worried about the theorem prover getting into a loop. In order to prevent infinite loops of reasoning occurring, we include an abbreviated copy of the proof stack in a 'label' (after (Gabbay 1996)). The label carries non-logical, arbitrary information about the progress of a proof, and is used for a variety of purposes. Labels are threaded through the clause, so that information can be passed from one subgoal to the next. For loop checking we add the current goal to the goal stack at the points where we are about to call something which might lead us into a loop, and we start this call by explicitly looking to see whether we are in a loop.[4] Another use for the labels is to carry around the equivalence classes that result from using rules relating to equality, which we use for dynamically rewriting clauses (rewriting the entire clause set, as proposed by (Gallier *et al.* 1993), is very expensive if you have large clause sets: we prefer to rewrite clauses as we use them, using information encoded in the label).

The two most important entries in the label (as far as this paper is concerned) are (i) the hypothetical actions that the theorem prover collects, and (ii) a note of the epistemic context of the proposition to which the label belongs. Referring again to the blocks-world example, Figure 3 shows `above(a,b)`, with its label, at the point in the search where the theorem prover has identified some actions that would achieve it.

The label is carried around as the first argument of the goal predicate `above`. The arguments `a` and `b` of the goal are the last arguments of the label (lines (32) and (33)). The normal form `above(a,b)@@[believe(j)]` of the goal is carried around in the label for goal protection purposes, and can be seen as an argument of `top` in line (2). The stack for the loop checker is `trail(C)`, seen on line (3). The actions `stack(slave,E,b)` and `stack(slave,a,E)` that have been found can be seen in lines (7 ff) and (20 ff) as arguments of `hypotheses(...)`. The epistemic context `[believe(j)]` of the goal is seen in line (31). Let us say a brief word here about epistemic entailment.

Epistemic entailment During a proof, a call is made to procedure `checkContexts` to see whether the epistemic context of a proposition is entailed by the epistemic context of the proposition from which it is being currently proved. The KNOWLEDGE AXIOM, TRANSMISSIBILITY AXIOM, and an INTROSPECTION AXIOM (a two-way reading of the so-called POSITIVE INTROSPECTION AXIOM (Hintikka 1962)) hold for knowledge as

[4] Clearly, the definition of `checkloop` is critical here—if you define it too tightly you will lose proofs which are actually available; if you define it too loosely, you will still get into loops. For practical purposes, we choose to define it quite tightly, so that we have some confidence that the algorithm will terminate (it is therefore, of course, bound to be incomplete. That's a choice you have to make—the more loops you cut out, the more potentially provable theorems you miss).

Figure 3:

```
1. above(label(shared(refs(B),
2.   top(above(a, b) @@ [believe(j)])),
3.     nonshared(trail(C),
4.       D,
5.       hypotheses([{,(above(a, b)
6.         @@ [believe(j)],
7.         (action(doer(slave),
8.           stack(slave, E, b),
9.           precons(isblock(b)
10.                  & isblock(E)
11.                  & on(E,table)
12.                  & clear(b)
13.                  & clear(E)),
14.          add(on(E, b)),
15.          delete(on(E, table)
16.                  & clear(b))),
17.        [believe(j)]))},
18.      {,(above(a, b)
19.        @@ [believe(j)],
20.        (action(doer(slave),
21.          stack(slave, a, E),
22.          precons(isblock(E)
23.                  & isblock(a)
24.                  & on(a,table)
25.                  & clear(E)
26.                  & clear(a)),
27.          add(on(a, E)),
28.          delete(on(a, table)
29.                  & clear(E))),
30.        [believe(j)]))}]),
31.      context([believe(j)]))),
32.  a,
33.  b)
```

modelled by the program. The axioms that hold for belief are the INTROSPECTION AXIOM and a mutual belief axiom.

Mutual beliefs are beliefs that conversants believe that they share. For example, we represent '*John believes that he and Sally mutually believe that pigs can't fly*', in standard logic as:

```
believes(john,
  believes(sally,
    mutuallybelieve([john,sally]),
     not(fly(pigs))))
```

A mutual belief that p by j and s entails an infinite number of beliefs ($B_j p$, $B_s p$, $B_j B_s p$, $B_s B_j p$, $B_j B_s B_j p \ldots$), however, this does not present a difficulty for the theorem prover, because, as mentioned above, the theorem prover embodies a deduction model of belief.

Now we will discuss in more detail how the actions got into the label, and how they are used to derive a solution to a planning problem.

Procedure

It is difficult to decide which is the more helpful way to describe our model. It is both a planner that employs a theorem

prover to hypothesise actions, and it is a theorem prover that employs a planning search. It is perhaps more intuitive to depict the model as the former, because the user calls the planning search, and the planning search calls the theorem prover; however, since the theorem prover hypothesises actions (a single action, or a series of actions), there is clearly a fuzzy boundary between theorem proving and planning in the model.

What we can say is that whereas the theorem prover posits desirable series of actions, it does not of itself invoke a search to find out whether the actions are doable (and to make them doable if they are not); this testing of and planning for preconditions is carried out by the planning search (which again invokes the theorem prover to hypothesise desirable actions). Here is a more formal and more detailed explanation.

The user calls `plan` to return a plan that would achieve some goal G from some initial state $W0$ (which has already been asserted into the Prolog database), and `plan` calls the theorem prover.

The theorem prover first reasons to see whether $W0 \models G$. It ensures no plan search is carried out by insisting that the argument of `hypotheses` (carried in the label) be the empty list. If $W0 \models G$, the empty plan is returned to the user as the solution. Otherwise, the theorem prover is called a second time, this time with a variable as the argument of `hypotheses`.

Now the theorem prover distributes the epistemic context of G over the conjuncts of G. We will illustrate using our blocks-world example, so goal G

```
(above(a,b) & on(a,c))@@[believes(j)]
```

becomes G'

```
above(a,b)@@[believes(j)]
& on(a,c)@@[believes(j)]
```

Next, the first conjunct `above(a,b)@@[believes(j)]` of G' is addressed, and an action is identified whose effects entail it. How is this done? Addressing conjunct `above(a,b)@@[believes(j)]` of G', the theorem prover finds that `above(a,b)` is the consequent of a rule whose antecedent is `on(a,b)`, and whose epistemic context is `[believes(Y)]` (see Figure 2). The epistemic context is distributed over the rule,[5] and a proof is sought of antecedent `on(a,b)@@[believes(Y)]`.

Now the theorem prover is looking for an action whose effects would render `on(a,b)@@[believes(Y)]` true. Individual action effects are stored as 'hypothetical facts' in the Prolog database. Figure 4 shows a skeletal version of the effect `on(F,G)` of action `stack(slave,F,G)` as it is stored in the database:[6]

[5] See earlier ('Modelling human reasoning') for our justification of this move.

[6] In order to make our presentation as clear as possible for the reader, we have missed out much of the body of the rule presented in Figure 4. Present in the actual body are procedures for checking whether the head of a rule is ground, for loop checking, and for managing equivalence classes (all mentioned earlier), as well as some additional constraints on `stack`. The label is also missing a lot of arguments, most of which have been mentioned already. The preconditions list has also been simplified.

Figure 4:

```
0. on(label(..., hypotheses(D),
1.          context(E)),
2.    F, G) :-
3.   checkContexts([believes(H)],E),
4.   hypothesis(
5.         label(hypotheses(D),
6.           context([believes(H)])),
7.         action(doer(slave),
8.             stack(slave,F,G),
9.             precons(isblock(G)
10.                      & isblock(F)
11.                      & on(F,table)
12.                      & clear(G)
13.                      & clear(F)),
14.            add(on(F,G)),
15.            delete(on(F,K)&clear(G))).
```

This hypothetical fact says "`on(F,G)` *would* be true, if you allowed me to introduce these hypotheses". The hypothesis the theorem prover wants to introduce here is `stack(J,F,G)` (lines (4)–(15)).

Continuing with the example, the hypothetical fact `on(F,G)` is unified with our current subgoal `on(a,b)`, and (among other things) a check is made to see whether the epistemic context `[believes(j)]` of the subgoal is entailed by the epistemic context `[believes(H)]` of the action, which it is (see lines (1), (3), and (6)). Now the label of `on(a,b)` is threaded through into the label of `above(a,b)`, carrying with it the list of hypothetical facts (which contains one fact at the moment in our example), and the list is returned to `plan`. Now the epistemic context of the action `stack(slave,a,b)` is distributed over the action's preconditions, and `plan` is called to prove the (now epistemic) preconditions of `stack(slave,a,b)`. Thus the whole planning procedure starts again from the beginning, this time with the preconditions of a desirable action as the goal. In our example, all the preconditions of `stack(slave,a,b)` are true, and so the action `stack(slave,a,b)` is applied (the `add` list facts are added to the Prolog database, and the `delete` list facts are deleted). A goal protection check is made, which succeeds, and the next conjunct `on(a,c)@@[believes(j)]` of the goal G' is addressed.

The procedure continues for the second goal conjunct much as just described for the first, except this time there is no need to invoke an `above(_,_)` rule to find a desirable action. However, the goal protection check fails, because to achieve `on(a,c)@@[believes(j)]`,

the planner would have to 'unachieve' the earlier achieved goal `above(a,b)@@[believes(j)]`. Consequently, Prolog fails and backtracks to try an alternative proof of `above(a,b)`. This time the theorem prover collects two hypothetical facts in its list of hypotheses, and returns them to `plan`. Figure 3 above shows the model's representation of the goal `above(a,b)@@[believes(j)]` at this stage, when the goal has just been returned to `plan` along with the hypothetical facts that would enable its proof to be completed. Notice that `stack(slave,E,b)` and `stack(slave,a,E)` are included as arguments of `hypotheses(...)`, being the two actions that need performing before the proof of `above(a,b)@@[believes(j)]` can be completed.

Having more than one action to deal with this time, `plan` makes a plan to achieve the preconditions of the first action `stack(slave,E,b)`, applies that plan, does a goal protection check, and then repeats the procedure for the second action `stack(slave,a,E)` (which by now is fully instantiated), with backtracking, until a plan is found that will achieve the first goal conjunct `above(a,b)@@[believes(j)]` from $W0$. The growing plan by this stage is `[stack(c,b),stack(a,c)]`. Now the second conjunct `on(a,c)@@[believes(j)]` of the goal G' is addressed, and the theorem prover finds that $W1$ models `on(a,c)@@[believes(j)]`, so a plan solution has been found.

Comments on procedure

The example that has been given is a simple task, chosen to keep the necessary explanation to a minimum. One factor that makes it simple is that there are only two conjuncts in the goal condition, and only one of these is an `above(_,_)` goal. The planner can, however, achieve goals having many conjuncts, and including many `above(_,_)` goals, including those which require multiple recursive calls of the `above(_,_)` rules in order to be achieved.

Another factor that makes the example simple is that, due to the fact that all the blocks are on the table and clear in the initial state, no planning is required to achieve the preconditions of the hypothesised actions. The example is not typical, however, and there are many tasks the planner can achieve which require preconditions to be achieved, not simply proved true.

The simplicity of the example has made it unnecessary to describe aspects of the plan search that have been specially designed to overcome unhelpful goal interactions (as first discussed by (Sussman 1974; Sacerdoti 1975)). These include a cross-plan-splicing procedure we call 'Think Ahead' (Field & Ramsay 2004). It works by incorporating thinking about the preconditions of chronologically later actions into the planning of earlier actions, which it does by exploiting the chronological information in the antecedent of a recursive domain-specific inference rule.

The reader may have noticed that our actions have an explicit 'doer' who is always 'slave':

```
forall(Y,
 believes(Y,
  action(doer(slave),stack(slave,F,G)...)))
```

This is done because we want to maintain a clear distinction between three agent types: (i) the agent who *knows about* actions and does all the thinking (`Y` in the `stack` operator); (ii) the agent who will supposedly *execute* the action (`slave` in `stack`); and (iii) the agent who *observes* the action. No explicit observer is mentioned in the `stack` operator, because we have represented stacking as a non-communicative action, which is generally the case.[7] Our operators for communicative actions, however, require both speaker (doer) and hearer (observer) to be made explicit, so that preconditions lists can refer to both the speaker's beliefs and the hearer's beliefs.

General comments and further work

The reader may have observed that there are perhaps alarmingly large amounts of quoted Prolog in this paper. Although this may appear unorthodox, we feel it serves our purpose well to include the code, and so we hope that the reader will forgive us. We are aiming to exploit the efficiency of Prolog as an engine for backward chaining through Horn clauses. The inclusion of a substantial amount of Prolog in the discussion is intended to show how we can obtain directly executable Prolog from problems stated in epistemic logic.

The authors do recognise that most of the AI community views ramifications as a problem (a part of the FRAME PROBLEM (McCarthy & Hayes 1969)), and not the norm. We concede that, whereas our presented model makes plans to achieve what we might call 'positive ramifications' (new propositions which become provable in the new state, but which are not listed in an operator's add list), there has been no mention thus far of what we might call 'negative ramifications'—facts in the knowledge base which are rendered false by the application of an action, but which do not appear in the action operator's delete list, and which therefore remain in the knowledge base, leading to potential inconsistencies.

Currently no procedure is implemented in the model to deal with negative ramifications, because there is no need for it. This is because the inferences made by the entertainment of hypotheses are only held at run-time, they are not collected into a cache to be asserted in the knowledge base for later reference. We recognise that there may be an argument for cacheing inferences, annotated by the hypotheses that were being entertained at that point. However, in our

[7] It is, however, easy to think of situations in which stacking a block is done to convey a message, for example, a mother stacking a block to communicate to her baby how stacking is done. In fact, we consider most 'physical' actions as potential communicative actions.

natural language domain, the costs are likely to outweigh the benefits, because we very seldom repeat inferences under the same sets of assumptions. If we *were* to cache inferences, we would probably appeal to the EXTENDED STRIPS ASSUMPTION (Reiter 1978, p. 407) (formulae not in the delete list of an operator will remain true after the action's performance, unless it can be shown otherwise), and employ a reasoning maintenance system (after (Doyle 1987)). At the point of the addition of a cache of inferences to the knowledge base, the RMS would use necessary supporting conditions to identify and delete any propositions in the database which the new propositions contradicted.

References

Allen, J.; Hendler, J.; and Tate, A. 1990. Editors. *Readings in planning*. San Mateo, California: Morgan Kaufmann.

Anderson, A. R., and Belnap jr., N. 1975. *Entailment: the Logic of Relevance and Necessity*, vol. 1. New Jersey: Princeton University Press.

Doyle, J. 1987. A truth maintenance system. In (Ginsberg 1987), pp. 259–79.

Feigenbaum, E. A., and Feldman, J. 1995. Editors. *Computers and thought*. Cambridge, Massachusetts: MIT Press. First published in 1963 by McGraw-Hill Book Company.

Field, D., and Ramsay, A. 2004. How to build towers of arbitrary heights, and other hard problems. In *Proc. 23rd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, 20–21 December 2004, Cork, Ireland.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2: 189–208.

Gabbay, D. M. 1996. *Labelled deductive systems*. Oxford University Press: Oxford.

Gallier, P.; Narendran, P.; Plaisted, D.; Raatz, S.; and Snyder, W. 1993. An algorithm for finding canonical sets of ground rewrite rules in polynomial time. *Journal of the Association for Computing Machinery* 40(1): 1–16.

Ginsberg, M. L. 1987. *Readings in nonmonotonic reasoning*. Los Angeles, California: Morgan Kauffmann.

Green, C. 1969. Application of theorem proving to problem solving. In *Proc. 1st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 219–39.

Herzberger, H. 1982. Notes on naive semantics. *Journal of Philosophical Logic* 11: 61–102.

Hintikka, J. 1962. *Knowledge and belief: An introduction to the two notions*. New York: Cornell University Press.

Konolige, K. 1986. *A deduction model of belief*. London: Pitman.

Kowalski, R. 1975. A proof procedure using connection graphs. In *Journal of the Association for Computing Machinery* 22(4): 572–595.

Kripke, S. 1963. Semantical considerations on modal logic. In *Acta Philosophica Fennica* 16: 83–94.

Manthey, R., and Bry, F. 1988. Satchmo: a theorem prover in Prolog. *Proc. 9th International Conference on Automated Deduction (CADE-9)*, volume 310 of *Lecture Notes in Artifcial Intelligence*, pp. 415–434, Berlin: Springer-Verlag.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4: 463–502.

Newell, A., and Simon, H. A. 1963. GPS, a program that simulates human thought. In (Feigenbaum & Feldman 1995), pp. 279–93.

Newell, A.; Shaw, J. C.; and Simon, H. A. 1957. Empirical explorations with the logic theory machine. In *Proc. Western Joint Computer Conference* 15: 218–239.

Ramsay, A. M. 1994. Focus on "Only" and "Not". In Y. Wilks. Editor. *Proc. 15th International Conference on Computational Linguistics (COLING-94)*, Kyoto, pp. 881–885.

Ramsay, A. 2001. Theorem proving for untyped constructive λ-calculus: implementation and application. In the *Logic Journal of the Interest Group in Pure and Applied Logics*, Vol. 9(1): 83–100.

Reiter, R. 1978. On closed world data bases. In H. Gallaire and J. Minker. Editors. *Logic and Data Bases*, pp. 55–76. New York: Plenum Press, 1978. Reprinted in (Ginsberg 1987), pp. 300–333.

Sacerdoti, E. D. 1975. The nonlinear nature of plans. In *Proc. 4th International Joint Conference on Artificial Intelligence (IJCAI)*, Tbilisi, Georgia, USSR, pp. 206–14. Reprinted in (Allen, Hendler, & Tate 1990), pp. 162–70.

Sussman, G. J. 1974. The virtuous nature of bugs. In *Proc. Artificial Intelligence and the Simulation of Behaviour (AISB) Summer Conference*. Reprinted in (Allen, Hendler, & Tate 1990), pp. 111–17.

Thomason, R. H. 1974. Editor. *Formal Philosophy: Selected papers of Richard Montague*. New Haven: Yale University Press.

Turner, R. 1987. A theory of properties. In *Journal of Symbolic Logic 52(2):455–472.*

Wallen, L. 1987. Matrix proofs for modal logics. *Proc. 10th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 917–23.

Whitehead, A. N., and Russell, B. 1910. *Principia mathematica*. Cambridge: Cambridge University Press.

4.5 Resolving Conflicts in Action Descriptions

Resolving Conflicts in Action Descriptions

Thomas Eiter and **Esra Erdem** and **Michael Fink** and **Ján Senko**
Institut für Informationssysteme 184/3,
Technische Universität Wien
Favoritenstraße 9-11, 1040 Wien, Austria
email: {eiter, esra, michael, jan}@kr.tuwien.ac.at

Abstract

We study the problem of resolving conflicts between an action description and a set of conditions (possibly obtained from observations), in the context of action languages. In this formal framework, the meaning of an action description can be represented by a transition diagram—a directed graph whose nodes correspond to states and whose edges correspond to transitions describing action occurrences. This allows us to characterize conflicts by means of states and transitions of the given action description that violate some given conditions. We introduce a basic method to resolve such conflicts by modifying the action description, and discuss how the user can be supported in obtaining more preferred solutions. For that, we identify helpful questions the user may ask (e.g., which specific parts of the action description cause a conflict with some given condition), and we provide answers to them using properties of action descriptions and transition diagrams. Finally, we discuss the computational complexity of these questions in terms of related decision problems.

Introduction

Action languages (Gelfond & Lifschitz 1998) are a formal tool for reasoning about actions, where an agent's knowledge about a domain in question is represented by a declarative action description that consists of logical formulas. Consider for instance a light bulb with a switch. When the light is off, then toggling the switch turns the light on; this can be expressed in the action description language $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) by the formula

$$\textbf{caused } Light \textbf{ after } Toggle \wedge \neg Light. \qquad (1)$$

On the other hand, at every state, if the light bulb is broken then the light is off. This can be expressed by the formula

$$\textbf{caused } \neg Light \textbf{ if } Broken. \qquad (2)$$

Other pieces of knowledge, like laws of inertia, may be also included. The meaning of such an action description can be represented by a transition diagram—a directed graph whose nodes correspond to the states of the world and the edges to the transitions describing action occurrences. For instance, the transition diagram of the action description above (consisting of (1), (2), and the inertia laws) is presented in Figure 1.

Note that the action description above is "buggy", since the effects of toggling the switch are not completely specified. Our goal is to "repair" such descriptions taking into account some additional information, such as observations or axioms about the action domain, which can be represented in an action query language (Gelfond & Lifschitz 1998). For example, when the light bulb is broken, toggling the switch may lead to a state where the light is off; this information is possibly obtained from some observations of the agent, and can be expressed in an action query language, e.g., by the statement

$$\textbf{possibly } \neg Light \textbf{ after } Toggle \textbf{ if } Broken. \qquad (3)$$

Some of the additional information may conflict with the action description. For instance, condition (3) does not hold relative to the action description above, since at the state where the light bulb is broken and the light is off, toggling the light switch is not possible. Thus, there is a conflict between the action description and this condition.

In this paper, we consider such conflicts, and how the agent's action description can be modified to resolve them. This may be accomplished in many different ways, and there is no canonical method which works satisfactorily in all cases. According to (Eiter *et al.* 2005), one might aim at dropping a smallest set of candidate formulas to resolve the conflict. In our example, dropping (1) would work. However, under further conditions, like

$$\textbf{necessarily } \neg Light \textbf{ after } Toggle \textbf{ if } Light, \qquad (4)$$

the conflict cannot be resolved just by dropping formulas: removing any of (1), (2) and inertia laws will not lead to an edge from a state where the light is on to a state where the light is off. A refined approach is needed which, semantically, modifies the transition diagram by suitable changes of the formulas to "repair" the action description such that the given queries (i.e., conditions) hold.

This paper makes two main contributions in this direction:

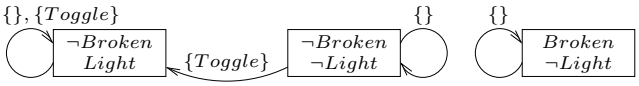


Figure 1: Transition diagram of the action description { (1), (2), (7) }.

1. It provides a precise *notion of conflict* between an action description and a set of queries, and presents a basic algorithm to resolve such conflicts. The idea is to modify the transition diagram of the action description by adding or deleting transitions so that all given conditions are satisfied; such a modification of the transition diagram is possible by adding, deleting or modifying some formulas in the action description. Based on this idea, our algorithm calculates a repair whenever it is possible.
2. Intuitive repair preferences might be difficult to formalize (e.g., both syntactic and semantic aspects might play a role) and thus to achieve with the basic algorithm above. In such cases, the designer might want to *ask questions* about the action description, the transition diagram, and the extra information, whose answers could guide her to come up with an appealing repair in an iterative repair process. For that, we explore several kinds of such questions and determine properties of action descriptions, transition diagrams, and extra information which are helpful in answering them. We also analyze the computational complexity of related problems.

Preliminaries

Transition diagrams. We start with a *propositional action signature* $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$ that consists of a set $\mathbf{F}$ of fluent names, and a set $\mathbf{A}$ of action names. Satisfaction of a propositional formula G over atoms $At \subseteq \mathbf{F} \cup \mathbf{A}$ by an interpretation $P \mapsto I(P) \in \{t, f\}$ for all $P \in At$ as usual, is denoted by $I \models G$. An *action* is an interpretation of $\mathbf{A}$, denoted by the set of action names that are mapped to t.

A *transition diagram* of $\mathcal{L}$ consists of a set S of *states*, a function $V : \mathbf{F} \times S \to \{f, t\}$ such that each state s in S is uniquely identified by the interpretation $P \mapsto V(P, s)$, for all $P \in \mathbf{F}$, and a subset $R \subseteq S \times 2^{\mathbf{A}} \times S$ of *transitions*. We say that $V(P, s)$ is the *value* of P in s. The states s' such that $\langle s, A, s' \rangle \in R$ are the possible *results of the execution* of the action A in the state s. We can think of a transition diagram as a labeled directed graph. Every state s is represented by a vertex labeled with the function $P \mapsto V(P, s)$ from fluent names to truth values; we denote by s the set of fluent literals satisfied by this function. Each triple $\langle s, A, s' \rangle \in R$ is represented by an edge from s to s' and labeled A. See Figure 1 for an example.

Action descriptions. We consider a subset of the action description language $\mathcal{C}$ (Giunchiglia & Lifschitz 1998) that consists of two kinds of expressions (called *causal laws*): *static laws*

$$\textbf{caused } L \textbf{ if } G, \tag{5}$$

where L is a fluent literal or *False*, and G is a propositional combination of fluent names; and *dynamic laws* of the form

$$\textbf{caused } L \textbf{ if } G \textbf{ after } U, \tag{6}$$

where L and G are as above, and U is a propositional combination of fluent names and action names. In (5) and (6) the part **if** G can be dropped if G is $True$.[1] An *action description* is a set of causal laws. For instance, one formalization of the light domain described in the introduction can be expressed in this language by the causal laws (1), (2), and the inertia laws

$$\begin{array}{l} \textbf{inertial } Light, \neg Light \\ \textbf{inertial } Broken, \neg Broken. \end{array} \tag{7}$$

Here an expression of the form **inertial** $L_1, ..., L_k$ stands for the causal laws **caused** L_i **if** L_i **after** L_i for $i \in \{1, ..., k\}$.

The meaning of an action description can be represented by a transition diagram as follows. We say that a causal law l is *applicable* to a transition $\langle s, A, s' \rangle$ in a transition diagram, if

- l is a static law (5), such that $s' \models G$; or
- l is a dynamic law (6), such that $s' \models G$ and $s \cup A \models U$.[2]

We denote by $D(tr)$ the set of all causal laws in an action description D that are applicable to a transition tr; by $H_D(tr)$ the set of the heads of all causal laws in $D(tr)$; and by $sat(H_D(tr))$ the set of interpretations of $\mathbf{F}$ that satisfy $H_D(tr)$.

Let D be an action description with an action signature $\mathcal{L} = \langle \mathbf{F}, \mathbf{A} \rangle$. Then the transition diagram $\langle S, V, R \rangle$ *described* by D, denoted $T(D)$, is defined as follows:

- S is the set of all interpretations s of $\mathbf{F}$ such that, for every static law (5) in D, $s \models G \supset L$,
- $V(P, s) = s(P)$,
- R is the set consisting of all transitions $\langle s, A, s' \rangle$ such that $sat(H_D(\langle s, A, s' \rangle)) = \{s'\}$.

We denote by $S(D)$ (resp. $R(D)$) the set of states (resp. transitions) of $T(D)$. For instance the transition diagram described by the action description consisting of (1), (2), (7) is shown in Figure 1.

Conditions. For expressing extra conditions, we consider here a language with two kinds of statements ("queries") about an action description: *possibility queries* and *necessity queries* of the respective forms

$$\textbf{possibly } \psi \textbf{ after } A \textbf{ if } \phi \tag{8}$$

[1] $True$ (resp. $False$) is the empty conjunction (resp. disjunction).

[2] We identify states s with the interpretations $P \mapsto V(P, s)$.

necessarily ψ **after** A **if** ϕ (9)

where ϕ and ψ are propositional combinations of fluent names, and A is an action. These queries are syntactically different from the ones presented in (Gelfond & Lifschitz 1998) and (Eiter *et al.* 2005); on the other hand, semantically they constitute a fragment of an extension of the action query language $\mathcal{P}$ (Gelfond & Lifschitz 1998) (from which we draw the term "query") and the condition language in (Eiter *et al.* 2005) (see Related and Further Work for a discussion).

A query q of form (8) (resp., (9)) *is satisfied at state s in a transition diagram T*, denoted $T, s \models q$, if either $s \not\models \phi$, or for some (resp., every) transition $\langle s, A, s' \rangle$ of T $s' \models \psi$ holds. We say that T *entails* a set Q of queries (denoted $T \models Q$), if $T, s \models q$ for every $q \in Q$ and for every state s in T. Accordingly, an action description D *entails* Q (denoted $D \models Q$) if $T(D) \models Q$.

Example 1 Let us consider the light domain described in the introduction as our running example. Let D be the action description consisting of (2), (1), and (7); and Q be the set of two queries: possibility query (3) and the necessity query (4), denoted by q_p and q_n respectively. Figure 1 shows $T(D)$ (i.e., the transition diagram of D). Then it can be easily verified that, at state $\{\neg Light,\ Broken\}$, since there is no transition from this state with action $Toggle$, the query q_n is trivially satisfied while q_p is not satisfied.

What a query describes is different from what a causal law does: action descriptions allow us to describe a transition diagram, based on causal explanations (what "is caused"), whereas queries allow us to state assertions (what "holds") about transition diagrams. These assertions may, e.g., be observations or axioms about the action domain. (See (Gelfond & Lifschitz 1998) for a discussion on action query languages.)

Conflicts in Action Descriptions

Given an action description D and a set Q of queries, we say that there is a *conflict* between D and Q, if $D \not\models Q$. Our goal is to resolve these conflicts by modifying the action description.

Conflicts can be characterized, from a semantic point of view, in terms of states and transitions "violating" some queries. We assume that the states of the world are correctly described by the given action description. Thus conflicts are existing transitions (for the violation of a necessity query) and non-existing transitions (for the violation of a possibility query) that cause such conflicts. The idea is then to "repair" an action description by a syntactic modification, such as adding, deleting, or modifying some of its causal laws, so that the detected conflicts are resolved by adding and/or deleting some transitions in the transition diagram.

For an action description D and a set Q of queries, the states and transitions violating possibility and necessity queries in Q, respectively, are as follows.

- A state s of $T(D)$ *violates* a possibility query q of form (8) in Q, if $T(D), s \not\models q$.
- A transition $tr = \langle s, A, s' \rangle$ of $T(D)$ *violates* a necessity query q of form (9) in Q (denoted $tr \not\models q$), if $s \models \phi$ and $s' \not\models \psi$.

Example 2 (cont'd) From $T(D)$ we can identify the following conflicts: the single state violating the possibility query q_p is $\{\neg Light,\ Broken\}$, and the single transition violating the necessity query q_n is $\langle \{Light, \neg Broken\}, \{Toggle\}, \{Light, \neg Broken\} \rangle$.

Since we suppose that states of the world are correctly described by D, we do not need to modify the static laws in D for a repair.

A Method for Resolving Conflicts

Under the assumption above, we can resolve conflicts between an action description D and a set Q of queries by the algorithm presented in Figure 2. Before we explain how this algorithm works, let us describe the notation used in it.

For a set Q of queries, we denote by Q_p (resp. Q_n) the set of possibility (resp. necessity) queries in Q. Then

$$conf_p(D, Q) = \{(q, s) \mid q \in Q_p, s \in S(D), T(D), s \not\models q\}$$
$$conf_n(D, Q) = \{(q, tr) \mid q \in Q_n, tr \in R(D), tr \not\models q\}.$$

For any triple $tr = \langle s, A, s' \rangle$, where s and s' are states and A is an action, and a dynamic causal law of form $l =$ **caused** L **if** U **after** G, $\langle s, A, s' \rangle \models l$ if either l is not applicable to tr, or $s' \models L$.

A *repair item* is an expression of form $(modify, l, l')$, or (add, l), where l and l' are dynamic causal laws. A *repair* is a set of repair items. For an action description D and a repair M, we denote by $M(D)$ the action description obtained from D by applying the modifications specified by the repair items in a repair M: (add, l) modifies D by adding l; $(modify, l, l')$ modifies D by replacing l with l'; all repair items are executed in parallel, i.e., if M comprises several $modify$ items for the same law l, all corresponding modifications l' are generated and eventually replace l. The repairs used by the algorithm RESOLVE(D, Q) are as follows (in causal laws, a state s stands for $\bigwedge_{L \in s} L$, and an action A for $\bigwedge_{X \in A} X \wedge \bigwedge_{X \in \mathbf{A} \setminus A} \neg X$):

$$Delete(\langle s, A, s' \rangle) = \{(add, \textbf{caused } False \textbf{ if } s' \textbf{ after } A \wedge s)\}$$

$$Insert(Tr, D) = \{(add, \textbf{caused } L \textbf{ if } s' \textbf{ after } A \wedge s) \mid \langle s,A,s' \rangle \in Tr, L \in s'\}$$
$$\cup \{(modify, l, \textbf{caused } L \textbf{ if } G \textbf{ after } U \wedge \alpha(Tr, l)),$$
$$(modify, l, \textbf{caused } L \textbf{ if } G \wedge L \textbf{ after } U \wedge A \wedge s) \mid l = \textbf{caused } L \textbf{ if } G \textbf{ after } U, l \in D, \langle s,A,s' \rangle \in Tr, \langle s,A,s' \rangle \not\models l\}$$

Algorithm RESOLVE$(D, Q) : Mod, Incon$

Input: An action description, D, and a set of queries, Q.
Output: A repair, Mod, and a set of queries, $Incon$.

$Mod := \emptyset$; $\quad Incon := \emptyset$;
for all $(q, tr) \in conf_n(D, Q)$ **do**
  $Mod := Mod \cup Delete(tr)$;
$D' := Mod(D)$; $\quad Ins := \emptyset$;
for all $(q, s) \in conf_p(D', Q)$ **do**
  $(q =$ **possibly** ψ **after** A **if** $\phi)$
  $Cands := \{\langle s, A, s'\rangle \mid s' \in S(D), s' \models \psi,$
    $\langle s, A, s'\rangle \models q', \forall q' \in Q_n\}$;
  if $(Cands \neq \emptyset)$ **then**
    select $tr \in Cands$;
    $Ins := Ins \cup \{tr\}$;
  else
    $Incon := Incon \cup \{q\}$;
return $Mod \cup Insert(Ins, D'), Incon$;

Figure 2: An algorithm to resolve conflicts.

where $\alpha(Tr, l) = \bigwedge_{\langle s,A,s'\rangle \in Tr, \langle s,A,s'\rangle \not\models l} \neg A \vee \neg s$.

In the algorithm above, first every transition tr violating the necessity queries in Q is removed, by adding to D the causal laws $Delete(tr)$. The new action description, D', entails Q_n. Then, for each state s violating a possibility query $q =$ **possibly** ψ **after** A **if** ϕ in Q relative to D', a set $Cands$ of transition candidates tr (triples of form $\langle s, A, s'\rangle$ where $s' \in S(D)$) that, when added to $T(D')$, would satisfy q at s (i.e., $s' \models \psi$) but not violate any necessity queries in Q (i.e., $tr \models q', \forall q' \in Q_n$), is computed. If such transition candidates exist (i.e., $Cands \neq \emptyset$), by introducing only one of these candidates into $T(D')$, the violation of q at s is prevented; otherwise no repair of D exists for Q (i.e., $Incon$ is not empty, and it contains the possibility queries that conflict with some necessity query in Q). The set Ins denotes all the transition candidates to be introduced into $T(D')$ so that no possibility query is violated in any state. Adding Ins to $T(D')$ can be achieved by adding to D' the causal laws $Insert(Ins, D')$.

Theorem 1 *For any repair Mod and set $Incon$ of queries output by* RESOLVE*(D, Q), the following hold:*

1. *$D \models Q$ iff $Mod = \emptyset$ and $Incon = \emptyset$;*
2. *$Incon = \emptyset$ iff $\exists D'$ such that $S(D) = S(D')$ and $D' \models Q$;*
3. *if $Incon = \emptyset$, then $Mod(D) \models Q$.*

The selection of a transition candidate $tr \in Cands$ for repairing a possibility query constitutes a choice point of the algorithm, where further heuristics can be employed to prune the set of repairs. We could, e.g., prefer transition candidates that *respect inertia conditions* or compute *minimal modifications*, i.e., repairs such that the modifications to $T(D)$ are minimal w.r.t. addition or deletion of transitions.

Example 3 (cont'd) Stipulating preference of transition candidates that respect inertia, the basic method resolves the conflicts as follows. First, the only transition violating q_n (i.e., $\langle s_1, \{Toggle\}, s_1\rangle$, where $s_1 = \{Light, \neg Broken\}$) is deleted from $T(D)$ by adding the law:

caused $False$ **if** $Light \wedge \neg Broken$ **after** $Toggle \wedge Light \wedge \neg Broken$.

Then, to resolve conflicts with q_p, the only transition candidate respecting inertia (i.e., $\{\langle s_2, \{Toggle\}, s_2\rangle\}$, where $s_2 = \{\neg Light, Broken\}$) is introduced into $T(D')$ by replacing (1) with the laws:

caused $\neg Light$ **if** $\neg Light \wedge Broken$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Broken$ **if** $\neg Light \wedge Broken$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Light$ **if** $Light$ **after** $Toggle \wedge \neg Light \wedge Broken$,
caused $Light$ **after** $Toggle \wedge \neg Light \wedge \neg Broken$.

We remark that algorithm RESOLVE can be implemented to use polynomial work space, producing its output, which is exponential in general, as a stream. After computing RESOLVE(D, Q), to get a more concise description, one may drop redundant causal laws that might have been introduced (e.g., (6) where $U \equiv False$), and apply some equivalence preserving transformations (e.g., replacing two laws **caused** L **after** $A \wedge U$ and **caused** L **after** $A \wedge \neg U$ with **caused** L **after** A.) Note also, that if there exists a repair for D, then there always also exists a repair D' of polynomial size. Informally speaking, D' can be obtained by expressing all necessity queries as dynamic laws and dispensing causality for all actions occurring in queries (**caused** L **if** L **after** A, for every literal L). Such a repair is independent of D apart from static laws and semantically it amounts to a complete transition graph w.r.t. actions occurring in queries modulo transitions violating necessity queries. Thus, it is even less appealing than solutions computed by RESOLVE(D, Q), which aim at making modifications as local as possible on single transitions (in order to retain the original semantics of D as much as possible even in case of further modifications). In most cases, however, neither of these basic repairs will be satisfactory. This motivates the utilization of additional knowledge of certain properties for repair.

Towards User-Assisted Repairs

With the method described above we can automatically repair an action description D with respect to a set Q of queries, under the assumption that the states of the world are described correctly by D. However, we may end up with

an action description with many causal laws, some possibly redundant or implausible. To get a more appealing description most often requires respecting additional knowledge or intuitions of the *designer* about the action description.

Usually, this knowledge cannot be easily formalized, as the following example illustrates:

Example 4 The designer of D might use her knowledge about the domain, i.e., light bulbs and switches, to infer from the conflict with the observation expressed in q_n that the duality of the toggle action has not been modeled correctly, and that the conflict with q_p is due to neglecting the effects of toggling when the bulb is broken. Hence, instead of D, she might consider D' consisting of (2), (7), and:

$$\begin{array}{l}\textbf{caused } Light \textbf{ after } Toggle \wedge \neg Light \wedge \neg Broken \\ \textbf{caused } \neg Light \textbf{ after } Toggle \wedge Light \wedge \neg Broken.\end{array} \quad (10)$$

Note that this description is more concise and plausible than the one generated by the basic method (see Example 3).

For (interactively) providing support to a designer repairing an action description, we present some questions that she may ask about Q, D, and $T(D)$. Answers to these questions are obtained from useful properties of queries, action descriptions, and transition diagrams.

Questions about queries and causal laws. To better understand the reasons for conflicts, the designer may want to check the given queries Q make sense with each other. Then the question is:

D1: If Q is contradictory relative to D, which queries in Q are contradictory?

We understand contradiction in a set Q as follows:

Definition 1 *A set Q of queries is* contradictory *relative to an action description D, if there is no action description D' such that $S(D) = S(D')$ and $D' \models Q$.*

With an answer to **D1**, the designer may drop contradictory queries from Q. Here are some sufficient conditions to find these queries.

Proposition 1 *A set Q of queries is contradictory relative to D, if Q includes some query (8) such that some $s \in S(D)$ satisfies ϕ, but no $s \in S(D)$ satisfies ψ.*

Proposition 2 *A set Q of queries is contradictory relative to D, if Q includes a necessity query* **necessarily** ψ' **after** A **if** ϕ' *and a possibility query (8) such that some state in $S(D)$ satisfies $\phi \wedge \phi'$, but no state in $S(D)$ satisfies $\psi \wedge \psi'$.*

Example 5 In our running example (i.e., Example 1), if Q had contained the query **possibly** $Light \wedge Broken$ **after** $Toggle$ **if** $True$ then, due to Proposition 1, Q would be contradictory relative to D.

If the given set of queries is not contradictory, then she may ask:

D2: If D does not satisfy a particular necessity query q in Q, which dynamic causal laws in D violate q?

We understand violation of a query as follows:

Definition 2 *A dynamic causal law $l \in D$* violates *a given necessity query q, if there is a transition $tc = \langle s, A, s' \rangle$ in $T(D)$ such that tc violates q, l is applicable to tc, and s' satisfies the head of l.*

Once the designer finds out which causal laws violate which queries, she may want to repair the action description in a way that some causal laws (e.g., the inertia laws) are not modified at all:

D3: Can we resolve a conflict between D and Q, without modifying a set D_0 of causal laws in D?

To answer **D3** the following definition and proposition are helpful.

Definition 3 *A transition diagram T* satisfies *a set D of causal laws (denoted $T \models D$), if, for each transition $tc = \langle s, A, s' \rangle$ in T, for each causal law $l \in D$, l is not applicable to tc or s' satisfies the head of l.*

Proposition 3 *Let D be an action description, and Q be a set of queries. If there exists a transition diagram T such that $T \models D$ and $T \models Q$, then there exists an action description D', such that $S(D) = S(D')$, $D \subseteq D'$ and $T = T(D')$.*

With this proposition, we can answer **D3** by checking if any transition diagram, having states $S(D)$, that satisfies D_0 also entails Q.

Example 6 In our running example it is possible to repair D without modifying the inertia laws: there exists an action description containing the inertia laws and satisfying the given queries (cf. Example 4).

In another scenario, the designer may suspect that the definition of a particular fluent causes problems, so she may want to know whether some particular laws have to be modified in order to obtain a repair:

D4: Do we have to modify a set D_0 of dynamic causal laws in D to resolve a conflict between D and Q?

For this, due to the proposition below, we can check whether none of the transition diagrams, with the same states as D (and thus as D_0), that satisfy D_0, entails Q.

Proposition 4 *Let D_0 be an action description, and Q be a set of queries. If there exists an action description D, such that $S(D_0) = S(D)$, $D_0 \subseteq D$ and $D \models Q$, then there exists a transition diagram T such that $T \models D_0$ and $T \models Q$.*

Questions about states and transitions. Alternatively, the designer may want to extract some information from $T(D)$. For instance, an answer to the following question gives information about states violating a query q in Q:

T1: Which states of $T(D)$ that satisfy a given formula ϕ', violate q?

Example 7 In Example 1, if we just consider states where the light is on (i.e., $\phi' = Light$), then the only state at which a query of Q is violated is $\{Light, \neg Broken\}$.

An answer to the following question gives information about transitions violating a necessity query q in Q:

T2: Given formulas ψ' and ϕ', which transitions $\langle s, A, s' \rangle$ of $T(D)$ such that s satisfies ϕ' and s' satisfies ψ', violate q?

With such information extracted from the transition diagram, the designer might decide *how* to modify the action description D.

Suppose that D does not satisfy a possibility query (8) in Q. The designer may want to learn about possible transition candidates that, when added to $T(D)$ by modifying the definition of some literal L in D, might lead to a repair:

T3: Given a literal L, for every state s of $T(D)$ such that s satisfies ϕ, is there some under-specified transition candidate $tc = \langle s, A, s' \rangle$ for D such that s' satisfies $\psi \wedge L$ and L is under-specified relative to tc? If there is, then what are they?

Here under-specification is understood as follows:

Definition 4 *A transition candidate $tc = \langle s, A, s' \rangle$ for D is* under-specified, *if $\{s'\} \subset sat(H_D(tc))$. A literal L is* under-specified *relative to a transition candidate tc, if $\{L, \overline{L}\} \cap H_D(tc) = \emptyset$.*

With a positive answer to **T3**, the designer may try to modify the description D, for instance, by adding the law **caused** L **if** ψ **after** $A \wedge \phi$.

Complexity Results

In this section, we consider computational aspects of the problems in the previous section and report complexity results for associated decision problems, respectively existence problems.

First let us remind the following result from (Eiter *et al.* 2005): Given an action description D and a set Q of queries, deciding $D \models Q$ is Π_2^p-complete in general. Note that, when Q contains the single query **possibly** $True$ **after** A **if** $True$, which expresses the executability of an action A at every state, this result conforms with the ones reported in (Turner 2002; Lang, Lin, & Marquis 2003).

In the following, we formally state two central results and, informally discuss how to obtain further results. The first main result is about the existence of a conflict resolution between an action description D and Q without modifying a subset D_0 of D.

Table 1: Complexity results (completeness) for problems **D1**–**D4**, **T1**–**T3**.

Problem	**D1**	**D2**	**D3**	**D4**	**T1**	**T2**	**T3**
	Σ_2^p	NP	Π_2^p	Σ_2^p	Σ_2^p	NP	Π_2^p
$Q_n = \emptyset$	$\mathrm{P}_{\|\|}^{\mathrm{NP}}$	$O(1)$	Π_2^p	Σ_2^p	Σ_2^p	$O(1)$	Π_2^p
$Q_p = \emptyset$	$O(1)$	NP	$O(1)$	$O(1)$	NP	NP	Π_2^p

Theorem 2 *Given D, Q, and $D_0 \subseteq D$, deciding if there exists some D', such that $S(D)=S(D')$, $D_0 \subseteq D'$, and $D' \models Q$, is Π_2^p-complete.*

We can show Π_2^p-hardness even for $D_0 = \emptyset$; for such D_0, complexity drops only if in addition Q is restricted to queries of form (8) (to $\mathrm{P}_{||}^{\mathrm{NP}}$-completeness, i.e., polynomial time with parallel queries to an NP-oracle, see, e.g., (Johnson 1990)).

The second main result is about the existence of a conflict resolution between an action description D and Q without modifying the transition diagram described by D.

Theorem 3 *Given D and Q, deciding if there exists some D', such that $S(D)=S(D')$, $D' \models Q$, and $R(D) \subseteq R(D')$, is Π_2^p-complete.*

We remark that if some repair of D for Q is known to exist, then deciding the above problem is coNP-complete.

Table 1 shows complexity results for the decision problems resp. existence problems related to the questions above (denoted **D1**–**D4**, resp. **T1**–**T3**) for the general case, and when Q_n=$\emptyset$, or Q_p=$\emptyset$.

Deciding whether Q is contradictory w.r.t. D (**D1**) is Σ_2^p-complete in general. Intuitively, this is because deciding the violation of a possibility query q is Σ_2^p-complete. We have to guess a violating state and verify, by means of an NP-oracle, for corresponding transition candidates that they do not satisfy q. Since we can express necessity queries by dynamic causal laws, this source of complexity carries over to deciding whether a set of (mixed) queries Q is contradictory. From these observations, Σ_2^p-completeness of the existence version of **T1** (i.e., whether such a state exists) is straightforward. However, if Q_n=$\emptyset$, to show that Q is contradictory w.r.t. D, it is sufficient to test whether, for some query (8) in Q_p, some state satisfies ϕ but no state satisfies ψ. This amounts to a Boolean combination of SAT instances, whose evaluation is in $\mathrm{P}_{||}^{\mathrm{NP}}$. For Q_p=$\emptyset$, note that a set Q_n of necessity queries cannot be contradictory.

On the other hand, deciding whether a necessity query q is violated is in NP: Guess and verify in polynomial time a transition violating q. Thus, e.g., deciding whether a causal

law $l \in D$ violates $q \in Q_n$ (i.e., **D2**) is NP-complete, as well as the existence version of **T2**.

D3 is the problem considered in Theorem 2, **D4** is the complementary problem, and corresponding results have been discussed above. Finally, the property of **T3** fails if there exists a state s satisfying ϕ, such that no underspecified transition candidate $tc = \langle s, A, s' \rangle$ for D exists, such that $s' \models \psi \wedge L$. Since for a given s, this can be checked with an NP-oracle, failure of the property is in Σ_2^p.

Related and Further Work

In (Eiter *et al.* 2005), the authors describe a method to minimally modify an action description, when new causal laws are added, by deleting some causal laws, so that given queries are satisfied. In the method above, we obtain an action description by adding or modifying some causal laws, motivated by some reasons for conflicts. For some problems, as discussed in the introduction, just dropping causal laws as in (Eiter *et al.* 2005) does not lead to a solution, whereas our method above does.

Similar to (Eiter *et al.* 2005), (Sakama & Inoue 2003) discusses how to minimally update a logic program syntactically so that given observations are satisfied. A semantical approach to updating a logic program by changes to Kripke structures (which are related to transition diagrams) is given in (Sefránek 2000), but no conditions are considered. In (Zhang, Foo, & Wang 2005) the authors describe how to resolve conflicts between a logic program and a set of constraints by "forgetting" some atoms in the program; in (Zhang & Foo 2005), they describe how logic programs can be updated following this approach.

That an action description can be transformed into a logic program (resp. a propositional theory) (Lifschitz & Turner 1999) might suggest applying update approaches for logic programming mentioned above (resp. for propositional logic (Winslett 1990; Katsuno & Mendelzon 1991)), when applicable (there is no given condition, the action description is inconsistent, etc.), and then obtaining an action description from the respective output. However, such transformations (to and from action descriptions) may lose information about the causal structure of the action domain and yield large and unintuitive action descriptions. In our work, we aim at preserving the causal structure, and keeping the action description intuitive and concise.

In (Balduccini & Gelfond 2003), the authors extend an action description, encoded as a logic program, with "consistency restoring" rules, so that when the action description and given observations are incompatible, these rules can be "applied" to get some consistent answer set. This, however, is more geared towards handling exceptions. Lifschitz describes in (Lifschitz 2000) an action domain in language $\mathcal{C}$ such that every causal law is defeasible (by means of an abnormality predicate). Then, to formulate some other variations of the domain (e.g., to satisfy some observations), the agent can just add new causal laws. Some of these laws are to "disable" some existing causal laws. In (Balduccini & Gelfond 2003) and (Lifschitz 2000), the causal laws of the original domain description are not modified.

Ongoing and future work includes an implementation of the method described above for resolving conflicts, and the investigation of the use of a SAT solver or an answer set solver to answer the questions discussed above (as suggested by the computational complexity results of the corresponding decision problems, presented in Table 1). Furthermore, (Eiter *et al.* 2005) employs a richer language for conditions, in which like in an extension of action query language $\mathcal{P}$ (Gelfond & Lifschitz 1998), e.g., conditions on sequences of action occurrences can be expressed. However, "repair" of such conditions is not immediate (e.g., many possibilities exist to eliminate "bad" trajectories from the transition diagram in general). This remains for future study.

Acknowledgments

We thank anonymous referees for comments and suggestions on a draft of this paper. This work is supported by the Austrian Science Fund (FWF) grant P16536-N04.

References

Balduccini, M., and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *Working notes of AAAI Spring Symposium*, 9–18.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating action domain descriptions. In *Proc. of IJCAI-05*, 418–423.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *ETAI* 3:195–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proc. AAAI*, 623–630.

Johnson, D. S. 1990. A catalog of complexity classes. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science*, volume A. MIT Press, Cambridge, MA. chapter 2, 67–161.

Katsuno, H., and Mendelzon, A. O. 1991. On the difference between updating a knowledge base and revising it. In *Proc. KR*, 387–394.

Lang, J.; Lin, F.; and Marquis, P. 2003. Causal theories of action: A computational core. In *Proc. of IJCAI*, 1073–1078.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proc. LPNMR*, 92–106.

Lifschitz, V. 2000. Missionaries and cannibals in the causal calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, 85–96.

Sakama, C., and Inoue, K. 2003. An abductive framework

for computing knowledge base updates. *TPLP* 3(6):671–713.

Sefránek, J. 2000. A Kripkean semantics for dynamic logic programming. In Parigot, M., and Voronkov, A., eds., *Proc. 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000), Reunion Island, France, November 11-12, 2000*, volume 1955 of *Lecture Notes in Computer Science*, 469–486. Springer.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02)*, 111–124.

Winslett, M. 1990. *Updating Logical Databases*. Cambridge University Press.

Zhang, Y., and Foo, N. Y. 2005. A unified framework for representing logic program updates. In Veloso, M. M., and Kambhampati, S., eds., *Proceedings Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 707–713. AAAI Press AAAI Press / The MIT Press.

Zhang, Y.; Foo, N.; and Wang, K. 2005. Solving logic program conflicts through strong and weak forgettings. In *Proc. IJCAI-05*, 627–632.

4.6 An Extended Query Language for Action Languages

An Extended Query Language for Action Languages (and its Application to Aggregates and Preferences)

James P. Delgrande
School of Computing Science
Simon Fraser University
Burnaby, B.C.
Canada V5A 1S6
jim@cs.sfu.ca

Torsten Schaub*
Institut für Informatik
Universität Potsdam
Postfach 90 03 27
D–14439 Potsdam, Germany
torsten@cs.uni-potsdam.de

Hans Tompits
Institut für Informationssysteme 184/3
Technische Universität Wien
Favoritenstraße 9–11
A–1040 Vienna, Austria
tompits@kr.tuwien.ac.at

Abstract

This paper continues our earlier work in representing arbitrary preferences in causal reasoning and planning systems, albeit in an oblique fashion. Previously, we defined a very general query language relative to histories; from this we specified a second language in which preferences on histories are defined. This in turn allowed us to define the notion of a most preferred history in a set of histories. In this paper, we extend these languages in two directions. First, we add a conditional construct that allows one to select between terms. Second, we add a capability for defining macros. With these two added constructs, one can now define aggregate quantities, such as the total cost of actions in a history, the maximum value of a fluent in a history, or a count of the number of times a fluent goes to zero in a history. Via the preference language, one can then express a preference for histories (or, plans) with minimum action cost, maximum value of a fluent, or in which a fluent is most often zero. We argue that this substantially increases the range of concepts about which one can express preferences.

Introduction

The traditional formulation of planning involves determining how a given goal can be attained, beginning from some initial state and by means of a given set of actions which alter the state of the world. A plan succeeds just when it is executable and attains the goal; otherwise, it fails. However, in realistic situations, things are not quite so simple. Thus, there may be requirements specifying that a plan should be as short as possible or that total cost, where costs are associated with actions, be minimised. As well, there may be *preferred* conditions, that are desirable to attain, but not necessary. Thus, in getting to the airport, the goal is to indeed arrive at the airport in good time; I may prefer to be able to pick up a coffee en route, but this preference is subordinate to the overall goal of getting to the airport. As well, there may be other preferences, such as preferring to take transit to driving, going by a particular route, etc. Each preference partitions the space of successful plans into those that satisfy the preference and those that do not. The goal of a planning problem now shifts to determining a *preferred* plan, in which a maximal set of preferences is satisfied, along with the goal. Such preferences also make sense outside of planning domains, and in fact apply to arbitrary sequences of temporal events. Hence, it is perfectly rational to prefer that one's favourite sport's team wins the championship, even though in the typical course of events one has no control over how the team performs.

In earlier work (Delgrande, Schaub, & Tompits 2004; 2005), we considered the problem of using general preferences over (fluent and action) formulas to determine preferences among temporal histories, or plans. While we focussed on histories as they are used in action description languages (Gelfond & Lifschitz 1998), our approach was, and is, applicable to any planning formalism. A history is defined as an interleaved sequence of states (of the world), and actions that take one state in a sequence to the next. We first specified a query language, $\mathcal{Q}_{\Sigma,n}$, (over some signature Σ and histories of maximum length n) in which one can determine whether an arbitrary expression is true in a given history. Given this language, we defined a *preference-specification language*, $\mathcal{P}_{\Sigma,n}$, that allows the definition of preference relations between histories. Via the language $\mathcal{P}_{\Sigma,n}$, we showed how to specify general preferences in temporal, causal, or planning frameworks. As well, the approach provided a very general language in which other "higher-level" constructs could be encoded, and in which other approaches could be expressed and so compared.

In the present paper, our motivating interest lies with being able to express preferences over aggregate quantities—that is, quantities that in some sense express a collective property of a set of fluent values. If actions come with a measure of their cost and duration, then two corresponding aggregate quantities of these measures would be the total action cost of a plan and the total duration of a plan. Clearly, in many cases these are quantities that one would want to minimise or, in a preference framework, prefer those plans with the minimal action cost or duration.

*Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

Such aggregates can be defined in our query language quite simply, given the addition of two capabilities. First, we extend the query language with a conditional term-forming operator. Syntactically, this operator is of the form $(\phi\ ?\ t_1 : t_2)$. This term denotes t_1 if ϕ is true; otherwise, it denotes t_2. Second, we add a "macro definition" capability, external to the language. That is, assertions in the query language can now contain macros that look exactly like fluents; given an appropriate set of macro definitions, these macros "compile out" so that one obtains assertions in the original language. Given this macro capability along with the conditional construct, one can define complex aggregate notions that can then take part in preference expressions. Thus, in our preference language we can now express a preference for histories with minimum action cost, maximum value of a fluent, or in which a fluent is zero at the most time points.

The next section briefly covers related work. This is followed by a section on the history-specific query language $\mathcal{R}_{\Sigma,n}$. This language extends our earlier query language $\mathcal{Q}_{\Sigma,n}$ with the new conditional operator. The following section describes adding a macro capability to this language and illustrates how aggregates can now be defined. The penultimate section describes the definition and use of our preference language $\mathcal{P}_{\Sigma,n}$ in this extended setting. The last section provides a brief discussion.

Background

Reasoning with preferences is an active area that is receiving increasing attention in AI. Hence, while the main topic of this paper is expressing aggregate quantities in our extended query language, our primary goal is the definition of a general language for specifying preferences on histories (or plans or other temporal sequences). Since histories and plans (implicitly or explicitly) involve sequences of states of the world, expressing aggregates and preferences over aggregates are of particular interest in planning systems.

In AI, Wellman & Doyle (1991) earlier suggested that the notion of *goal* is a relatively crude measure for planners to achieve, and instead that a relative preference over possible plan outcomes constitutes (or should constitute) a fundamental objective for planning. They show how to define goals in terms of preferences and, conversely, how to define (incompletely) preferences in terms of sets of goals. Here, we maintain a strict division between (hard) goals and (soft) preferences, although a framework along the lines of Wellman and Doyle is expressible in our method simply by taking the overarching goal as being $\top$—i.e., all valid histories are the subject of the preferences.

Myers & Lee (1999) assume that there is a set of desiderata, such as affordability or time, whereby successful plans can be ranked. A small number of plans is generated, where the intent is to generate divergent plans. The best plan is then chosen, based on a notion of Euclidean distance between these select attributes. Hence, they deal implicitly with aggregate quantities, with a concrete, quantitative notion of preference, applied to a small set of successful and presumably representative plans. In related work, Haddawy & Hanks (1992) use a utility function to guide a planner.

One approach to preferences in planning has been proposed by Son & Pontelli (2004), where a language for specifying preferences between histories is presented. This language is an extension of action language $\mathcal{B}$ (Gelfond & Lifschitz 1998), and is subsequently compiled into logic programs under the answer-set semantics. The notion of preference explored is based on so-called *desires* (what we call *absolute preferences* in previsous work (Delgrande, Schaub, & Tompits 2004)), expressed via formulas built by means of propositional as well as temporal connectives such as *always*, *until*, etc. From desires, preferences among histories are induced as follows: Given a desire ϕ, a history H is preferred to H' if $H \models \phi$ but $H' \not\models \phi$.

Similarly, Bienvenu & McIlraith (2005) address planning with preferences in the situation calculus. Preferences are founded on the notion of *basic desire formulas*, whose members are somewhat analogous to formulas in our language $\mathcal{Q}_{\Sigma,n}$. These formulas in turn are used in the composition of *atomic preference formulas* (essentially chains of preferences) and *general preference formulas*. As the authors note, this approach extends and modifies that of Son & Pontelli (2004) although expressed in terms of the situation calculus rather than an action language. Based on a concrete means of explicitly combining preferences, a best-first planner is given. Fritz & McIlraith (2005) employ a subset of this language in an approach to compile preferences into DT-Golog.

Eiter *et al.* (2003) describe planning in an answer-set programming framework where action costs are taken into account. The approach allows the specification of desiderata such as computing the shortest plan, the cheapest plan, or some combination of these criteria. This is achieved by employing weak constraints, which filter answer sets, and thus plans, based on *quantitative* criteria.

A General Query Language for Histories

Histories and Queries on Histories

In specifying histories, we extend the notation of Gelfond & Lifschitz (1998) in their discussion of *transition systems*. As described, virtually any general planning system (or indeed causal-reasoning formalism) could be used to provide a setting for our approach; as well, the approach is more broadly applicable than just to planning problems.

Definition 1 *An* action signature Σ *is a quadruple* $\langle D, F, V, A\rangle$, *where* D *is a set of* value names, F *is a set of* fluent names, $V : F \rightarrow 2^D \setminus \emptyset$ *assigns a* domain *to each fluent, and* A *is a set of* action names.

Σ *is* propositional *iff* $D = \{0, 1\}$*, and it is* finite *iff* D*,* F*, and* A *are finite. Moreover, a fluent name* $f \in F$ *is* propositional *iff* $V(f) = \{0, 1\}$.

For simplicity we will assume throughout that action signatures are finite and that D is the set of non-negative integers.

Definition 2 *Let* $\Sigma = \langle D, V, F, A\rangle$ *be an action signature.*

A history*,* H*, over* Σ *is a sequence*

$$(s_0, a_1, s_1, a_2, s_2, \ldots, s_{n-1}, a_n, s_n),$$

where $n \geq 0$*, and*

- *each* s_i*,* $0 \leq i \leq n$*, is a mapping assigning each fluent* $f \in F$ *a value* $v \in V(f)$*, and*
- $a_1, \ldots, a_n \in A$.

The functions $s_0, \ldots, s_n$ *are called* states*, and* n *is the* length *of history* H*, symbolically* $|H|$.

The states of a history may be thought of as possible worlds, and the actions take one possible world into another.

We need to be able to refer to fluent and action names in a history. Since a fluent's value will vary depending on the time point under consideration, we also need to be able to refer to time points and their relations. To this end, we define a query language on histories of maximum length n over an action signature Σ, named $\mathcal{R}_{\Sigma,n}$.

Definition 3 *Let* $\Sigma = \langle D, F, V, A\rangle$ *be an action signature and* $n \geq 0$ *a natural number.*

The alphabet of $\mathcal{R}_{\Sigma,n}$ *consists of the following items:*

1. *a set* $\mathcal{V} = \{i, j, \ldots\}$ *of* time-stamp variables*, or simply* variables*,*
2. *the set of integers,*
3. *the sets* D*,* F*, and* A*,*
4. *the sentential connectives '*$\neg$*' and '*$\supset$*', and the quantifier symbol '*$\exists$*',*
5. *the arithmetic function symbols '*$+$*', '*$-$*', and '*$\cdot$*', the arithmetic relation symbol '*$<$*', and the equality symbol '*$=$*', and*
6. *the parentheses '(' and ')', and the symbols '?' and ':'.*

Terms in $\mathcal{R}_{\Sigma,n}$ are of two types: those denoting time points and those denoting fluent values.

Definition 4 *Let* $\Sigma = \langle D, F, V, A\rangle$ *be an action signature and* $n \geq 0$ *a natural number.*

The terms *of* $\mathcal{R}_{\Sigma,n}$ *are as follows:*

1. *A* time term *is an arithmetic term recursively built from* $\mathcal{V} \cup \{0, \ldots, n\}$*, employing* $+$ *and* $\cdot$ *(as well as parentheses) in the usual manner.*
2. *A* (fluent) value term *is either a member of* D*, an expression of the form* $f(t)$*, where* $f \in F$ *and* t *is a time term, or an arithmetic term recursively built from value terms, employing* $+$*,* $-$*, and* $\cdot$ *in the usual manner. Additionally, if* ϕ *is a formula (cf. Definition 5) and* e_1 *and* e_2 *are value terms, then* $\phi\ ?\ e_1 : e_2$ *is a value term.*

The intent here is that, in $\mathcal{R}_{\Sigma,n}$, time terms range over the numbers $0, \ldots, n$ only, while fluent value terms may denote arbitrarily large integers. So, our definition of the class of formulas of $\mathcal{R}_{\Sigma,n}$, given next, will accommodate that, for instance, $f(0) = 5$ is well formed, where f is a fluent name, with intended interpretation that f has value 5 at time point 0, whereas $\exists i f(0) = i$ is not well formed.

Definition 5 *Let* $\Sigma = \langle D, F, V, A\rangle$ *be an action signature and* $n \geq 0$ *a natural number.*

The formulas *of* $\mathcal{R}_{\Sigma,n}$ *are as follows:*

1. *A* time atom *is an expression of the form* $(t_1 < t_2)$ *or* $(t_1 = t_2)$*, where* t_1, t_2 *are time terms.*
   A value atom *is either an expression of form* $a(t)$*, where* $a \in A$ *and* t *is a time term, or an expression of the form* $(v_1 < v_2)$ *or* $(v_1 = v_2)$*, where* v_1, v_2 *are value terms.*
   An atom containing no variables is ground*.*
2. *A* literal *is an atom possibly preceded by the negation sign* $\neg$*.*
3. *A* formula *is a Boolean combination of atoms, along with quantifier expressions of form* $\exists v$*, for* $v \in \mathcal{V}$*, formed in the usual recursive fashion.*
4. *A* query *is a closed formula, i.e., with no free time-stamp variables.*

For a propositional fluent f and time term e, we write $f(e)$ for $f(e) = 1$ and $\neg f(e)$ for $f(e) = 0$; in such a case, $f(e)$ is said to be *true* or *false* (or equivalently *true at* e or *false at* e), respectively.

We define the operators $\wedge$, $\vee$, and $\leq$, and the universal quantifier $\forall$, in the usual way. Parentheses may be dropped in formulas if no ambiguity arises, and we may write quantified formulas like $\mathsf{Q}v_1\mathsf{Q}v_2\,\alpha$ as $\mathsf{Q}v_1, v_2\,\alpha$, for $\mathsf{Q} \in \{\forall, \exists\}$. For formula α, variables $v_1, \ldots, v_k$, and numbers $i_1, \ldots, i_k$, $\alpha[v_1/i_1, \ldots, v_k/i_k]$ is the result of uniformly substituting v_j by i_j in α, for each $j \in \{1, \ldots, k\}$. Thus, if $v_1, \ldots, v_k$ are the free variables in α, then $\alpha[v_1/i_1, \ldots, v_k/i_k]$ is a closed formula.

Variables range over time points, and so quantification applies to time points only. Atoms consist of actions or fluents indexed by a time point, or of a predicate on arithmetic (time point) expressions. Atoms are used to compose formulas in the standard fashion, and queries consist of closed formulas. This means that we remain within the realm of propositional logic, since quantified expressions $\forall v$ and $\exists v$ can be replaced by the conjunction or disjunction (respectively) of their instances.

Example 1 *Let* $pickup \in A$*,* $red \in F$*, and* $i, j \in \mathcal{V}$*. Then,*

$$pickup(4),\ red(i + j),\ i < j + 2$$

are atoms. As well,

$$red(j) \wedge (\forall k\ (k < j) \supset \neg red(k)),$$
$$ageJohn(t) > ageMary(t + 1)$$

are formulas but not queries, and

$$\exists i, j((i+2<j) \wedge pickup(i) \wedge \neg red(j)),$$
$$\exists i(ageJohn(i) > ageMary(i+1)),$$
$$\exists i(\neg(ageJohn(i) = 35)),$$
$$(((0=1)?1:0)=0)$$

are closed formulas and so queries (*assuming an appropriate action signature*).

The intent of the first formulas above is that it be true in a history in which $pickup$ is true at some time point and three or more time points later red is false. For the last formula above, we have that $0 = 1$ is false, therefore the value of $((0 = 1)?1 : 0)$ is 0, and since $0 = 0$ is true, the formula is true.

The following operators, which basically correspond to similar ones well-known from linear temporal logic (LTL), can be defined:

- $\Box b = \forall i\, b(i)$;
- $\Diamond b = \exists i\, b(i)$; and
- $(b \,\mathsf{U}\, g) = \exists i \left(g(i) \wedge \forall j((j<i) \supset b(j))\right)$.

Here, b and g are propositional fluent names or action names. Informally, $\Box b$ expresses that b *always* holds, $\Diamond b$ that b holds *eventually*, and $b \,\mathsf{U}\, g$ that b holds continually *until* g holds. Other LTL operators are likewise expressible.

Semantics of Queries

The definition of truth of a query with respect to a history is done in two parts. First, we define an interpretation function $\mathcal{I}$ that gives the denotation of terms; from this we define the notion of truth. To ease detail, we use the notation that for a ground term t, $val(t)$ is the value of t according to standard integer arithmetic. Note that although the next two definitions are interdependent, these definitions of denotation and truth are strictly compositional.

Definition 6 *Given query language $\mathcal{R}_{\Sigma,n}$ and a history $H = (s_0, a_1, s_1, \ldots, a_k, s_k)$ over Σ of length $k \leq n$, the* denotation $\mathcal{I}_H$ *is given by:*

1. *If t is a ground time term, then:*

$$\mathcal{I}_H(t) = \begin{cases} 0 & \text{if } val(t) < 0; \\ k & \text{if } val(t) > k; \\ val(t) & \text{otherwise.} \end{cases}$$

2. *If t is a ground fluent value term, then:*
   (a) *if t is $f(t')$, for $f \in F$, then $\mathcal{I}_H(t) = s_i(f)$, where $i = \mathcal{I}_H(t')$;*
   (b) *if t is $\phi\ ?\ e_1 : e_2$, then:*
   if $H \models_{\mathcal{R}_{\Sigma,n}} \phi$, then $\mathcal{I}_H(t) = \mathcal{I}_H(e_1)$, otherwise $\mathcal{I}_H(t) = \mathcal{I}_H(e_2)$;
   (c) *otherwise, $\mathcal{I}_H(t) = val(t)$.*

The rationale behind Part 1 in the definition above is to allow that, while a time term calculation may refer to a time point greater than the length of a history or less than time point 0, its denotation should not. Hence, if $val(t)$ is greater than the length k of history H, then a ground atomic query $\phi(t)$ will be satisfied by H if it is satisfied at the last state of H.

It can be observed for a ground time term t, that $0 \leq \mathcal{I}_H(t) \leq n$, while for a value term we do not necessarily obtain that $\mathcal{I}_H(t) \in D$. This conforms to intuitions: fluents have value only within a history, and time points need to refer to times within a history. On the other hand, value terms are used (among other things) to determine aggregate quantities. Thus, we could have a propositional domain with $D = \{0, 1\}$; however, if we wanted to count the number of times that a light was on, say, we would need other integer values.

Now we can define what it means for a history to satisfy a query expressed in $\mathcal{R}_{\Sigma,n}$.

Definition 7 *Let $H = (s_0, a_1, s_1, \ldots, a_k, s_k)$ be a history over Σ of length $k \leq n$, and let Q be a query of $\mathcal{R}_{\Sigma,n}$.*
We define $H \models_{\mathcal{R}_{\Sigma,n}} Q$ as follows:

1. *If $Q = a(t)$ is a ground action atom, then $H \models_{\mathcal{R}_{\Sigma,n}} Q$ iff $a = a_{\mathcal{I}_H(t)}$.*
2. *If $Q = (v_1 \circ v_2)$, for $\circ \in \{<, =\}$, is a ground (fluent or time) atom, then $H \models_{\mathcal{R}_{\Sigma,n}} Q$ iff $(\mathcal{I}_H(v_1) \circ \mathcal{I}_H(v_2))$.*
3. *If $Q = \neg\alpha$, then $H \models_{\mathcal{R}_{\Sigma,n}} Q$ iff $H \not\models_{\mathcal{R}_{\Sigma,n}} \alpha$.*
4. *If $Q = \alpha \supset \beta$, then $H \models_{\mathcal{R}_{\Sigma,n}} Q$ iff $H \not\models_{\mathcal{R}_{\Sigma,n}} \alpha$ or $H \models_{\mathcal{R}_{\Sigma,n}} \beta$.*
5. *If $Q = \exists v\alpha$, then $H \models_{\mathcal{R}_{\Sigma,n}} Q$ iff, for some $0 \leq i \leq n$, $H \models_{\mathcal{R}_{\Sigma,n}} \alpha[v/i]$.*

If $H \models_{\mathcal{R}_{\Sigma,n}} Q$ holds, then H *satisfies* Q. For simplicity, if $\mathcal{R}_{\Sigma,n}$ is unambiguously fixed, we also write $\models$ instead of $\models_{\mathcal{R}_{\Sigma,n}}$.

We have the following results concerning complexity in $\mathcal{R}_{\Sigma,n}$.

Theorem 1 *Let Σ be an action signature and $n \geq 0$ a natural number.*

1. *Given a history $H = (s_0, a_1, s_1, \ldots, a_k, s_n)$ over Σ of length n and a query*

$$Q = (\mathsf{Q}_1 i_1)(\mathsf{Q}_2 i_2) \ldots (\mathsf{Q}_m i_m)\ C$$

   of $\mathcal{R}_{\Sigma,n}$, where $\mathsf{Q}_i \in \{\forall, \exists\}$, $1 \leq i \leq m$, and C contains no quantifiers, then deciding whether $H \models_{\mathcal{R}_{\Sigma,n}} Q$ holds can be determined in $O(|C|^m)$ time.
2. *Given a query Q of $\mathcal{R}_{\Sigma,n}$, then deciding whether there is a history H over Σ of length n such that $H \models_{\mathcal{R}_{\Sigma,n}} Q$ holds is PSPACE-complete.*

The proof of the first part is straightforward, since for each quantifier expression $(\mathsf{Q}i)\alpha$, one needs to test the n substitution instances of α conjunctively (for universal quantification) or disjunctively (for existential quantification). For the second part, for showing that the

problem is at least in PSPACE, the reduction is from satisfiability of quantified Boolean formulas to formulas of $\mathcal{R}_{\Sigma,1}$; for showing that it is no worse than PSPACE, the reduction is from formulas of $\mathcal{R}_{\Sigma,n}$ to formulas of linear-time temporal logic (LTL).

The language $\mathcal{R}_{\Sigma,n}$ is quite expressive. Indeed, it can be shown that $\mathcal{R}_{\Sigma,n}$ subsumes the languages $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ due to Gelfond & Lifschitz (1998), as well as our earlier language $\mathcal{Q}_{\Sigma,n}$ (Delgrande, Schaub, & Tompits 2005), with respect to expressivity.

Macros

In realistic applications, one is often faced with non-propositional fluents, such as temperature, amount of rain, etc. Moreover, actions often have associated costs and other measures, such as duration. Frequently too, one is interested in aggregates of such quantities—for example, determining the total rainfall, total cost of actions, maximum value attained over an interval, and so on. In the next section, we will see how these types of quantities permit preference statements expressing certain optimisations or desiderata. In this section, we show how, using our conditional construct ($\_ \ ? \ \_ : \_$) along with the addition of a macro capability, we can express aggregates of fluent values without adding to the overall complexity of the language.

Definition of Macros

Macros are defined as follows:

Definition 8 *Let $\Sigma = \langle D, F, V, A\rangle$ be an action signature and $n > 0$. Then, the alphabet of a macro language over Σ and n consists of the alphabet of $\mathcal{R}_{\Sigma,n}$, together with*

1. *a set of fluent names F', where $F \cap F' = \emptyset$, and*
2. *a set of parameter names $\mathbf{P} = \{\$1, \$2, \dots\}$.*

We refer to $\langle \Sigma, F'\rangle$ as the (macro) signature *for such a macro language.*

The fluent names in F' serve as macro names, while the elements of $\mathbf{P}$[1] serve as parameters for the macros, and for which terms will be substituted in the macro expansion.

Definition 9 *Given a macro signature $\langle \Sigma, F'\rangle$ and $n > 0$, a macro M is a sequence*

$$(\langle \nu_1, \mu_1\rangle, \dots, \langle \nu_m, \mu_m\rangle),$$

where

1. *each ν_i is of the form $f(t)$, where $f \in F'$, and either $t \in \{0, \dots, n\}$ or t is the parameter name $\$i$, and*
2. *each μ_i is a value term over $\langle D, F \cup F', V, A\rangle$, but with added primitive terms $\mathbf{P}$, formed in the expected recursive manner.*

[1] In the present paper we use only parameter $\$1$, since we deal just with unary macros. The inclusion of $\mathbf{P}$ anticipates a possible generalization to many-placed macros (which in turn makes conceptual sense only if the language $\mathcal{R}_{\Sigma,n}$ is generalized to allow flunets of arity greater than 1).

In a formula involving macros, $\mu_i[x/t]$ will be the formula μ_i with every occurrence of parameter x replaced by term t.

Definition 10 *Let ϕ be a formula and*

$$M = (\langle \nu_1, \mu_1\rangle, \dots, \langle \nu_m, \mu_m\rangle)$$

a macro.

- ν_i matches *value term t in ϕ, if*
  - $\nu_i = f(a)$,
  - $t \in D$ *or*
    $t = f(b)$ *and if a is a constant, then $a = b$.*

  Matches will be understood to be symmetric.
- $\phi[\nu_i/\mu_i]$ *is defined by:*

  for every $t = f(b)$ in ϕ that matches $\nu_i = f(a)$, replace t in ϕ by $\mu_i[\$1/b]$.
- ϕ^M, *the macro expansion of ϕ by M, is defined by: $((\dots((\phi[\nu_1/\mu_1])[\nu_2/\mu_2])\dots)[\nu_k, \mu_k])^*$, where θ^* denotes iteration of the implicit operators to a fixed point.*
  The macro expansion of a set of macros is defined in the obvious manner.

Note that the definition of a macro expansion ensures, for a macro $(\langle \nu_1, \mu_1\rangle, \dots, \langle \nu_m, \mu_m\rangle)$, that for $i < j$, ν_i will be expanded before ν_j. Of course, the macro expansion is not always defined; for example, the macro $(\langle f(\$1), f(\$1)\rangle)$ can lead to problems. Rather, with appropriately defined macros, one now may express aggregate functions.

Example 2 *The following macro can be used to find the maximum value that fluent f takes on in a history of length n:*

$$\begin{array}{l}(\langle maxf,\ maxfh(n)\rangle,\\ \ \langle maxfh(0),\ f(0)\rangle,\\ \ \langle maxfh(\$1),\ ((f(\$1) > maxfh(\$1-1))\ ?\\ \qquad f(\$1) : maxfh(\$1-1))\rangle).\end{array}$$

So, for example, in a history of length 2, the macro $maxf$ simply stands for the following expression (highlighting the structure by underlining):

$$(f(2)\ >\ \underline{(f(1) > \underline{f(0)}\ ?\ f(1) : \underline{f(0)})}\ ?\ \underline{f(2) : (f(1) > \underline{f(0)}\ ?\ f(1) : \underline{f(0)})}).$$

That is, neither $maxf$ nor its "helper macro" $maxfh$ appear in the actual expression. Last, given the associated mappings $s_0 : f \mapsto 2$, $s_1 : f \mapsto 3$, and $s_2 : f \mapsto 1$, the expression in Example 1 evaluates to 3.

It is important to note that aggregates like $maxfh(i)$ are merely macros representing nested value terms. Thus, in particular, they are not fluents nor are they terms in the language $\mathcal{R}_{\Sigma,n}$.

Example 3 *The following macro sums the values of f in a history:*

$$(\langle sumf,\ sumfh(n)\rangle,\ \langle sumfh(0),\ f(0)\rangle,\ \langle sumfh(\$1),\ f(\$1)+sumfh(\$1-1)\rangle).$$

Example 4 *A similar definition as in the previous example can be given for counting all occurrences of (propositional fluent) f being true:*

$$(\langle cntf,\ cntfh(n)\rangle,\ \langle cntfh(0),\ f(0)\ ?\ 1:0\rangle,\ \langle cntfh(\$1),\ cntfh(\$1-1)+(f(\$1)\ ?\ 1:0)\rangle).$$

Further refinements are easily specified, as illustrated next.

Example 5 *Preferring histories with the globally minimum number of days (states) on which it rained more than t litres can be modelled by an extension of the count macro:*

$$(\langle cntf,\ cntfh(n)\rangle,\ \langle cntfh(0),\ (f(0)\le t)\ ?\ 0:1\rangle,\ \langle cntfh(\$1),\ cntfh(\$1-1)+((f(\$1)\le t)\ ?\ 0:1)\rangle).$$

For modelling action costs, we associate with each action a fluent yielding the cost of the corresponding action and then sum the fluent. This is simple, and moreover allows us to associate with an action other measures, such as duration.

Correctness of Macros

The correctness of a macro (that is to say, the macro does what it is supposed to do) can be determined informally by inspection, or by an inductive argument. Another approach is to define a term corresponding to a macro as an extension to the language $\mathcal{R}_{\Sigma,n}$, and then prove that this new term corresponds to the value computed by the macro. This as well gives a means for specifying the semantics of a macro (after a fashion) and also the semantics of aggregates. We illustrate with two examples.

To begin with, consider where we wish to define a fluent that will correspond to the maximum value of some other fluent obtained so far in a history. Specifically, for fluent f, we want to define a fluent *fmaxfh*, where

$$fmaxfh(i) = \max_{0\le j\le i} f(j).$$

We can do this by extending Definition 6 as follows:

$$\mathcal{I}_H(fmaxfh(i)) = \begin{cases} \mathcal{I}_H(f(0)), & \text{if } i=0;\\ \mathcal{I}_H(f(i) > fmaxfh(i-1)\ ? & \\ \quad f(i) : fmaxfh(i-1)), & \text{otherwise.}\end{cases}$$

We can now define fluent *fmaxf* to correspond to *fmaxfh*(n) – that is, extend the interpretation function so that $\mathcal{I}_H(fmaxf) = \mathcal{I}_H(fmaxfh(n))$. It is now a straightforward, albeit tedious, task to show that any formula ϕ that mentions fluent *fmaxf* has precisely the same truth value in any history as does the macro expansion of macro *maxf* in the formula ϕ with all occurrences of *fmaxf* replaced by *maxf*.

Similarly, we can define a fluent *fsumf* that will correspond to the sum of the values of fluent f obtained so far in a history; i.e., we can define

$$fsumf(i) = \sum_{0\le j\le i} f(j).$$

We do this by extending Definition 6 as follows:

$$\mathcal{I}_H(fsumf(i)) = \begin{cases} \mathcal{I}_H(f(0)), & \text{if } i=0;\\ \mathcal{I}_H(f(i)+fsumf(i-1)), & \text{otherwise.}\end{cases} \quad (1)$$

Again, we can show that the truth value of formulas mentioning *fsumf* will correspond to the macro expansion of corresponding formulas mentioning macro *sumf*.

The overall structure of these correspondences is clear. In the case of macros, we typically have a recursive expansion based on time points, and that terminates at time point 0 or n. In the case of the interpretation function $\mathcal{I}_H$ (relative to history H), this recursive expansion is mirrored in a recursive definition of the value of intermediate fluents. Thus, in this case, the value of a defined fluent (such as *fmaxf*) can be determined from the underlying fluent (viz. f) by a process akin to macro expansion.

The overall scheme can be obviously extended to more than one fluent, and more than a single time point for each step. For example, we can define a fluent ex that counts the number of times the value of fluent f exceeds that of g two time points ago, as follows:

$$\mathcal{I}_H(ex(i)) = \begin{cases} 0, & \text{if } i=0 \text{ or } i=1\\ \mathcal{I}_H(f(i) > g(i-2)\ ? & \\ \quad ex(1-i)+1 : ex(1-i))\ , & \text{otherwise.}\end{cases}$$

From this, a corresponding macro can straightforwardly be defined. This in turn suggests a methodology for constructing macros: First, give a mathematical definition for the desired mathematical concept, for example, the sum of fluent f is given by $\max_{0\le j\le n} f(j)$. This can then be defined within our language $\mathcal{R}_{\Sigma,n}$, as done in (1), and then essentially discharged from the language by the macro definition in Example 3.

Application: Expressing Preferences on Histories

In this section, we sketch a major application of our language, and in particular of the conditional construct and macros, to expressing preferences between histories.

A Preference Language

We briefly summarise our earlier work on preferences and a preference language; for formal details see (Delgrande, Schaub, & Tompits 2005). The central notion of this approach is of a *preference frame*, consisting of a pair $(\boldsymbol{H}, \boldsymbol{P})$, where

- $\boldsymbol{H}$ is a set of *histories* and
- $\boldsymbol{P}$ is a set of *preferences* on histories.

Histories are as in the preceding sections. A preference specifies an individual criterion for distinguishing among histories. A preference defines a binary relation, consisting of pairs of histories where one history is preferred to the other, according to the preference. We define a preference among two histories directly in terms of a formula ϕ; this formula is in a language $\mathcal{P}_{\Sigma,n}$ that extends $\mathcal{Q}_{\Sigma,n}$ so that one can now also refer to histories, in the following sense. We define that H_h is not less preferred than H_l, written $H_l \preceq_\phi H_h$, just if $\langle H_l, H_h\rangle \models \phi$, where ϕ expresses a preference condition between these two histories in $\mathcal{P}_{\Sigma,n}$. $H_l \preceq_\phi H_h$ holds if ϕ is true by evaluating it with respect to H_l and H_h. This requires that we are able to refer to fluent and action names at time points and in histories. Preferences are expressed by means of a formula composed of

- Boolean combinations of fluents and actions indexed by time points and by a history, and
- quantifications over time points.

Indexing with respect to time points and histories is achieved via *labelled atoms* of form $\ell : b(i)$. Here, ℓ is a *label*, either $\boldsymbol{l}$ or $\boldsymbol{h}$, referring to a history which is considered to be lower or higher ranked, respectively, b is an action or fluent name, and i is a time point. Semantically, $\langle H_l, H_h\rangle \models \boldsymbol{l} : b(i)$ holds if b holds at time point i in history H_l; and analogously for $\langle H_l, H_h\rangle \models \boldsymbol{h} : b(i)$. This is extended to labelled formulas in the expected fashion.

For example, we can express that history H_h is preferred to history H_l if fluent f is true at some point in H_h but never true in H_l by the formula

$$\phi = (\boldsymbol{h} : \exists i f(i)) \ \wedge \ (\boldsymbol{l} : \forall i \neg f(i)), \qquad (2)$$

providing $\langle H_l, H_h\rangle \models \phi$ holds.

Each preference $\phi \in \boldsymbol{P}$ induces a binary relation $\preceq_\phi$ on $\boldsymbol{H}$. Depending on the type of preference encoded in $\boldsymbol{P}$, one would supply a strategy from which a maximally preferred history is selected. Thus for preferences only of the form (2), indicating which fluents are desirable, the maximally preferred history might be the one which was ranked as "preferred" by the greatest number of preferences in $\boldsymbol{P}$.

Expressing Preferences among Histories

We define a preference among two histories, H_l and H_h, directly in terms of a formula ϕ:

$$H_l \preceq_\phi H_h \qquad \text{iff} \qquad \langle H_l, H_h\rangle \models \phi. \qquad (3)$$

The intent with $\langle H_l, H_h\rangle \models \phi$ is that ϕ expresses a condition in which H_h is at least as preferred as H_l. This requires that we be able to talk about the truth values of fluents and actions in H_l and H_h. Using our query language on histories, $\mathcal{R}_{\Sigma,n}$, and the notion of truth in a history for a query, we can define a *preference-specification language*, enabling the definition of preference relations between histories, as in (3).

Definition 11 *Let $\Sigma = \langle D, F, V, A\rangle$ be an action signature and $n \geq 0$ a natural number.*

We define the preference-specification language $\mathcal{P}_{\Sigma,n}$ *over $\mathcal{R}_{\Sigma,n}$ as follows:*

1. *The alphabet of $\mathcal{P}_{\Sigma,n}$ consists of the alphabet of the query language $\mathcal{R}_{\Sigma,n}$, together with the symbols $\boldsymbol{l}$ and $\boldsymbol{h}$, called* history labels*, or simply* labels*.*
2. *Atoms of $\mathcal{P}_{\Sigma,n}$ are either time atoms of $\mathcal{R}_{\Sigma,n}$ or expressions of the form $\ell : q$, where $\ell \in \{\boldsymbol{l}, \boldsymbol{h}\}$ is a label and q is an action or value atom of $\mathcal{R}_{\Sigma,n}$.*
   Atoms of the form $\ell : p$ are also called labelled atoms*, with ℓ being the label* of $\ell : p$*. We call $\ell : p$* ground *iff p is ground.*
3. *Formulas of $\mathcal{P}_{\Sigma,n}$ are built from atoms, as introduced above, in a similar fashion as formulas of $\mathcal{R}_{\Sigma,n}$. We call formulas of $\mathcal{P}_{\Sigma,n}$ also* preference formulas*.*
4. *A* preference axiom*, or simply* axiom*, is a closed preference formula, i.e., containing no free time-stamp variables.*

For a formula α of $\mathcal{R}_{\Sigma,n}$ and a history label $\ell \in \{\boldsymbol{l}, \boldsymbol{h}\}$, by $\ell : \alpha$ we understand the formula resulting from α by replacing each action or value atom $b(t)$ of α by the labelled atom $\ell : b(t)$. Informally, a labelled atom $\ell : p$ expresses that p holds in a history associated with label ℓ. This is made precise as follows.

Definition 12 *Let Σ be an action signature and $n \geq 0$. Let ϕ be a preference axiom of $\mathcal{P}_{\Sigma,n}$ and H_l, H_h histories over Σ with $|H_i| \leq n$, for $i = l, h$.*

The relation $\langle H_l, H_h\rangle \models_{\mathcal{P}_{\Sigma,n}} \phi$ is recursively defined as follows:

1. *If $\phi = \ell : p$ is a ground labelled atom, for $\ell \in \{\boldsymbol{l}, \boldsymbol{h}\}$, then $\langle H_l, H_h\rangle \models_{\mathcal{P}_{\Sigma,n}} \phi$ iff*
   (*a*) *$H_l \models_{\mathcal{R}_{\Sigma,n}} p$, for $\ell = \boldsymbol{l}$, and*
   (*b*) *$H_h \models_{\mathcal{R}_{\Sigma,n}} p$, for $\ell = \boldsymbol{h}$.*
2. *Otherwise, $\langle H_l, H_h\rangle \models_{\mathcal{P}_{\Sigma,n}} \phi$ is defined analogously as the truth conditions for $\models_{\mathcal{R}_{\Sigma,n}}$.*

If $\langle H_l, H_h\rangle \models_{\mathcal{P}_{\Sigma,n}} \phi$ holds, then $\langle H_l, H_h\rangle$ is said to *satisfy* ϕ. If Σ and n are clear from the context, we may simply write $\models$ instead of $\models_{\mathcal{P}_{\Sigma,n}}$.

Definition 13 *Let ϕ be a preference axiom of $\mathcal{P}_{\Sigma,n}$. For histories H_l, H_h over Σ of maximum length n, we define*

$$H_l \preceq_\phi H_h \quad \textit{iff} \quad \langle H_l, H_h\rangle \models_{\mathcal{P}_{\Sigma,n}} \phi.$$

Note that the employment of the symbol $\preceq_\phi$ is purely suggestive at this point, since $\preceq_\phi$ may have none of the properties of an ordering.

We give some illustrations next.

Example 6 *The formula*

$$\big(\boldsymbol{h} : (\exists i f_1(i) \wedge \forall i \neg f_2(i))\big) \wedge \big(\boldsymbol{l} : (\exists i f_2(i) \wedge \forall i \neg f_1(i))\big)$$

expresses a preference of f_1 over f_2 in the sense that, for all histories H_l, H_h, we prefer H_h over H_l whenever it holds that H_h satisfies f_1 but not f_2, whilst H_l satisfies f_2 but not f_1.

Given this, we can now state a preference for histories where the fluent f is maximum (as defined in Example 2). In place of macro $maxf$, we employ two macros, $h{:}maxf$ and $l{:}maxf$.

In the former case, we have the macro definition:[2]

$$\begin{array}{l}(\langle h{:}maxf,\ h{:}maxfh(n)\rangle,\\ \ \langle h{:}maxfh(0),\ \boldsymbol{h} : f(0)\rangle,\\ \ \langle h{:}maxfh(\$1),\\ \quad ((\boldsymbol{h} : f(\$1) > h{:}maxfh(\$1-1))\ ?\\ \quad\ (\boldsymbol{h} : f(\$1)) : (h{:}maxfh(\$1-1))\rangle).\end{array}$$

The macro $l{:}maxf$ is defined analogously. This enables the expression of the preference:

$$h{:}maxf \leq l{:}maxf. \tag{4}$$

Similarly, we can state a preference for histories where the values for f are maximum over every subinterval $[0, i]$ where $0 \leq i \leq n$ by:

$$\forall i(h{:}maxfh(i) \leq l{:}maxfh(i)).$$

Since our designated fluent f could in fact be measuring a quantity such as action cost or action duration, the preceding examples show how one can express preferences for plans (assuming that the histories are produced by a planner) that maximize certain quality measures.

As a last example, we can express a preference for the range of values of f being maximal in one of (at least) three ways. This can be directly expressed, given our macro $maxf$ and a suitable definition of $minf$ by:

$$(h{:}maxf - h{:}minf) \ \leq\ (l{:}maxf - l{:}minf).$$

Otherwise, one could directly encode a macro expressing this notion, or, third, one could define a macro $maxdiff$ that expands to $maxf - minf$.

The preference frame with the preference (4) induces a total preorder on the set of histories $\boldsymbol{H}$. It is then a simple matter to select the most preferred history or histories. Of course, things may get much more complicated, particularly in the presence of different forms of preferences or multiple types of preferences. For details on such issues, see (Delgrande, Schaub, & Tompits 2005).

[2] To be sure, this isn't great notation. On the one hand, $h{:}maxf$ is a macro name (composed of 6 characters) while $\boldsymbol{h} : f(\$1)$ is a semantic entity in our language $\mathcal{P}_{\Sigma,n}$ once a term is substituted in for \$1 in the macro expansion.

Conclusion

We have addressed the problem of expressing general preferences that include aggregate quantities over histories. Thus, inter alia, we addressed adding preferences, including preferences on aggregates, in planning systems. We first defined a query language, $\mathcal{R}_{\Sigma,n}$, that extended our earlier query language $\mathcal{Q}_{\Sigma,n}$ (Delgrande, Schaub, & Tompits 2005) by the addition of a conditional term-forming operator. In addition, we defined the notion of macros for this language. Given this, we showed how our second language $\mathcal{P}_{\Sigma,n}$ could be employed for defining general preferences including preferences on aggregate quantities. As before, the framework allows the expression of conditional preferences, or preferences holding in a given context, as well as (trivially) absolute preferences, expressing a general desirability that a formula hold in a history.

We have argued previously that the overall approach is very general and flexible; specifically, we argued that previous approaches to preferences in planning are expressible in our formalism. With the added aggregate capability we can now express aggregate preferences like those discussed by Eiter *et al.* (2003), but notably within our broader preference framework. As well, while the approach is formulated within the framework of action languages, our results are applicable to general planning formalisms.

In a planning context, our approach would amount to generating plans and selecting the most preferred plan based on the preferences. As such, the approach is readily adaptable to an anytime algorithm, in which one may select the currently-best plan(s), but with the hope of a more-preferred plan being generated. An obvious topic for future work is to directly generate a preferred plan (rather than selecting from candidate plans); however, this appears to be a significantly difficult problem. Another topic for future work is to generalize our notion of macro, perhaps allowing for fluent arguments. Thus, if it can be carried out with no additional computational overhead, rather than having a macro $maxf$ for the maximum value of fluent f, it would be more convenient to express this as $max(f)$ where f is now an argument for a general max macro.

Acknowledgements The first author was partially supported by a Canadian NSERC Discovery Grant. The second author was partially supported by DFG under grant SCHA 550/6, TP C. The authors are grateful to Yannis Dimopoulos for getting them interested in preferences on aggregrates.

References

Bienvenu, M., and McIlraith, S. 2005. Specifying and generating preferred plans. In McIlraith, S.; Peppas, P.; and Thielscher, M., eds., *Seventh International Symposium on Logical Formalizations of Commonsense Reasoning*, 25–32.

Delgrande, J.; Schaub, T.; and Tompits, H. 2004. Domain-specific preferences for causal reasoning and planning. In Dubois, D.; Welty, C.; and Williams, M., eds., *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning*, 673–682. Whistler, BC: The AAAI Press/The MIT Press.

Delgrande, J.; Schaub, T.; and Tompits, H. 2005. A general framework for expressing preferences in causal reasoning and planning. In McIlraith, S.; Peppas, P.; and Thielscher, M., eds., *Seventh International Symposium on Logical Formalizations of Commonsense Reasoning*, 46–54.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003. Answer set planning under action costs. *Journal of Artificial Intelligence Research* 19:25–71.

Fritz, C., and McIlraith, S. 2005. Compiling qualitative preferences into decision-theoretic Golog programs. In *IJCAI'05 Workshop on Nonmonotonic Reasoning, Action and Change*, 45–52.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Transactions on AI* 3. Available at `http://www.ep.liu.se/ej/etai/`.

Haddawy, P., and Hanks, S. 1992. Representations for decision-theoretic planning: Utility functions for deadline goals. In *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, 71–82.

Myers, K., and Lee, T. 1999. Generating qualitatively different plans through metatheoretic biases. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 570–576.

Son, T., and Pontelli, E. 2004. Planning with preferences in logic programming. In Lifschitz, V., and Niemelä, I., eds., *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'04*), volume 2923 of *Lecture Notes in Artificial Intelligence*, 247–260. Springer Verlag.

Wellman, M., and Doyle, J. 1991. Preferential semantics for goals. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 698–703.

5 Argumentation, Dialogue, and Decision Making

An Argument is a set of statements related in such a way that it supports a proposition which represents its conclusion. From this informal definition, an exciting and exceedingly fertile area of research has developed. Over the years, interest in argumentation has expanded dramatically, driven in part by theoretical advances but also by successful demonstrations of a wide range of practical applications.

Historically, argumentation has been an effective approach to nonmonotonic reasoning since the work of John Pollock, Ronald P. Loui, and others in the eighties, who showed that argumentation is a very natural way of conceptualising this form of commonsense reasoning. In the early nineties Dung and others showed that argumentation is also very suitable as a general framework for relating nonmonotonic logics of different styles. Finally, in recent years argument-based systems have been introduced in the area of Multi-Agent Systems producing a plethora of work based on the introduction of the rational exchange of information which has opened to research a wide range of application domains.

Argumentation can be studied on its own, but it also has interesting relations with other topics, such as dialogue and decision. For instance, argumentation is an essential component of phenomena such as fact finding investigations, negotiation, legal procedure and online dispute mediation. However, only recently researchers have begun to explore the use of argumentation in these contexts.

Argumentation has brought to the NMR community a perspective that has been consistently shown to be useful in approaching problems that have been studied in the field since its beginnings and, at the same time, has opened new areas that were unexplored until its inception. In the years since the first NMR's Session on Argument, Dialogue, and Decision, much progress has been achieved by the argumentation community which has grown at an increasingly fast pace. In the present Session a number of remarkably interesting works have been included.

The workshop was divided in three sections. On the first one, Lianne Bodenstaff, Henry Prakken, and Gerard Vreeswijk studied the logical formalisation and implementation of dialogue systems for argumentation in their work *On Formalising Dialog Systems for Argumentation in Event Calculus*. Anthony Hunter, in *Approximate Arguments for Efficiency in Logical Argumentation*, proposes to ameliorate the computational problem of generating actual arguments by using approximate arguments. Next, Laura Cecchi, Pablo Fillottrani, and Guillermo Simari in *On Complexity of DeLP through Game Semantics*, consider the complexity of two decision problems: the problem of existence of an argument, and the problem of recognizing an argument as such in the context of the DeLP system.

During the first part of the afternoon, Leila Amgoud and Mathieu Serrurier, in their work *An Argumentation Framework for Concept Learning*, focus on the version space approach to concept learning and provide an argumentation-based framework which translates the problem of classifying examples into a decision one. Carlos Chesñevar and Guillermo Simari, propose *An Abstract Model for Computing Warrant in Skeptical Argumentation Frameworks*, which introduces a novel approach to model the search space of possible arguments necessary for warrant computation in a skeptical abstract argumentation framework. Geoffroy Aubry and Vincent Risch consider the role of deceitful arguments in argumentation frameworks in *Managing Deceitful Arguments with X-logics* by introducing the notion of lie as a new kind of possible answer for an agent.

In the last part of the session four presentations were included. In *Comparing Decisions in an Argumentation-based Setting*, Leila Amgoud and Henri Prade, after remarking that argument-based decision making have until now relied only on one type

of arguments, propose a systematic typology that identifies new types of arguments, and explore decision principles that can be used for comparing decisions. Nicolas Rotstein and Alejandro García's *Defeasible Reasoning about Beliefs and Desires* shows how a deliberative agent can represent beliefs and desires and perform defeasible reasoning in order to support its derived beliefs. Pietro Baroni and Massimiliano Giacomin in *Refining SCC Decomposition in Argumentation Semantics: A First Investigation* argue that a finer decomposition than the proposed SCC-recursive approach to argumentation semantics which considers some suitably defined internal substructures of strongly connected components may be appropriate. Additionally, they support, in some cases, more intuitive results than those in the original approach.

Emil Weydert is investigating the possibility of a "competition" for comparing different argumentation systems. In his talk *How to model arguments in scientific texts: A proposal for comparing nonmonotonic reasoning*, he will motivate the need for such a competition, and discuss the possibility of basing the competition on representing, and reasoning with, scientific knowledge, a domain where there is much conflicting information, and there is the potential for some interesting issues for argumentation systems.

Session chairs

Leila Amgoud
(amgoud@irit.fr)

Guillermo R. Simari
(grs@cs.uns.edu.ar)

Program committee

Philippe Besnard
(besnard@irit.fr)

Carlos I. Chesñevar
(cic@eps.udl.es)

Phan Minh Dung
(dung@cs.ait.ac.th)

Thomas F. Gordon
(thomas.gordon@fokus.fraunhofer.de)

Peter McBurney
(p.j.mcburney@csc.liv.ac.uk)

Simon Parsons
(parsons@sci.brooklyn.cuny.edu)

Henry Prakken
(henry@cs.uu.nl)

Iyad Rahwan
(irahwan@acm.org)

Francesca Toni
(ft@doc.ic.ac.uk)

Bart Verheij
(b.verheij@ai.rug.nl)

Gerard Vreeswijk
(gv@cs.uu.nl)

Schedule Wednesday 31 May 2006 (Rydal-Elterwater Room)
Session Chairs: L. Amgoud and G. Simari

- 10.30 L. Bodenstaff, H. Prakken, and G. Vreeswijk, On formalizing dialog systems for argumentation in event calculus
- 11.00 A. Hunter, Approximate arguments for efficiency in logical argumentation
- 11.30 L. Cecchi, P. Fillottrani, G. Simari, On complexity of DeLP through game semantics
- 12.00 Lunch
- 14.00 L. Amgoud and M. Serruruer, An argumentation framework for concept learning
- 14.30 C. Chesnevar and G. Simari, An abstract model for computing warrant in skeptical argumentation frameworks
- 15.00 G. Aubry abd V. Risch, Managing deceitful arguments with X-logics
- 15.30 Coffee
- 16.00 L. Amgound and H. Prade, Comparing decisions in an argumentation-based setting
- 16.30 N. Rotstein and A. Garcia, Defeasible reasoning about beliefs and desires
- 17.00 P. Baroni and M. Giacomin, Refining SCC decomposition in argumentation semantics: A first investigation
- 17.30 E. Weydert, How to model arguments in scientific texts: A proposal for comparing nonmonotonic reasoning strategies

5.1 On Formalising Dialog Systems for Argumentation in Event Calculus

On Formalising Dialogue Systems for Argumentation in the Event Calculus

Lianne Bodenstaff
Department of Computer Science
University of Twente
The Netherlands
l.bodenstaff@ewi.utwente.nl

Henry Prakken
Department of Information and Computing Sciences
Utrecht University
and
Faculty of Law
University of Groningen
The Netherlands
henry@cs.uu.nl

Gerard Vreeswijk
Department of Information and Computing Sciences
Utrecht University
The Netherlands
gv@cs.uu.nl

Abstract

This paper studies the logical formalisation and implementation of dialogue systems for argumentation, motivated by the claim that this benefits their formal investigation and implementation. A case study is described in which a dialogue system of Prakken is formalised in Shanahan's version of the 'full' Event Calculus and then implemented as a Prolog program. Then a second case study is briefly summarised in which a dialogue system of Parsons, Wooldridge and Amgoud is formalised in the same way. From the case studies some conclusions are drawn on the usefulness of the formalisation method.

Introduction

Logical specification of dialogue systems benefits both the formal investigation of such systems and their implementation in declarative programming languages. Such implementation in turn supports the design of flexible dialogue systems, in which variations in the communication language or protocol can be handled easier than when they are hard-coded in a lower-level programming language. This paper studies the logical specification of a class of dialogue systems that have so far largely been specified in semi-formal ways, viz. systems involving argumentation.

When intelligent agents interact, the need for argumentation can arise in various ways. For instance, collaborating agents who must jointly solve a problem may argue about the pros and cons of the various possible solutions (Atkinson, Bench-Capon, & McBurney 2005), or self-interested negotiating agents may try to persuade each other to accept their offers by arguing about the merits and drawbacks of these offers (Rahwan *et al.* 2003). Various dialogue systems for persuasive argumentation have been proposed, e.g. (Gordon 1994; Amgoud, Maudet, & Parsons 2000; Parsons, Wooldridge, & Amgoud 2003; Atkinson, Bench-Capon, & McBurney 2005; Prakken 2005) but most of them have so far not been fully formally specified in a declarative way. Some are specified in informal mathematical metalanguage, while other systems are functionally specified. Two notable exceptions are Brewka (2001) and Artikis, Sergot, & Pitt (2003), who formalise systems in, respectively, the Situation and Event Calculus. However, these papers study single and particular systems that lack several features of other systems, which therefore still await further investigation.

Locution	Description	P	PWA
claim φ	a player claims that φ is the case	x	x
claim S	a player claims the set of propositions *S*		x
why φ	a player asks why φ is the case	x	x
concede φ	a player concedes proposition φ	x	x
concede S	a player concedes the set of propositions *S*		x
argue A	a player puts forward argument *A*	x	
retract φ	a player retracts proposition φ	x	

Table 1: Some speech acts for argumentation

This paper aims at contributing to such investigation by carrying out a case study in which a dialogue system for argumentation of Prakken (2005) is formalised in a variant of the Event Calculus and then implemented as a Prolog program. After that a second case study will be briefly summarised in which a system of Parsons, Wooldridge, & Amgoud (2003) was formalised in the same way. We end with a discussion of related research and some conclusions.

The system to be formalised

Like most dialogue systems for argumentation, the one of Prakken (2005) is specified as a dialogue game. Such a game firstly has a *topic language* L_t with a logic $\mathcal{L}$, specifying how to represent and reason with domain information. Secondly, a dialogue game has a *communication language* L_c with a *protocol* P: the former specifies the well-formed locutions, or speech acts, while the latter regulates their use. Table 1 displays a possible set of locutions for argumentation dialogues and indicates which of them are part of Prakken (2005) (P) and Parsons, Wooldridge, & Amgoud (2003) (PWA). In this table, the locution $argue\ A$ consists of premises ($prem(A)$) and conclusion ($conc(A)$). The protocol of a dialogue game specifies the 'rules of the game', i.e., it specifies the allowed moves at each point in a dialogue. A dialogue game also has *effect rules*, which specify the effects of utterances on the players' commitments. For instance, an

utterance of *claim* φ initiates the speaker's commitment to φ while the utterance of *retract* φ terminates such commitment. Finally, a dialogue game has *termination rules* and sometimes *outcome rules*.

We now summarise in more detail the P system as it is specified in informal mathematical metalanguage in (Prakken 2005). In fact, the system specified below is an instance of a framework allowing for alternative instantiations on the just-mentioned building blocks. The protocol of P is very liberal in its structural aspects: essentially, both players can speak whenever they like, except that they cannot speak at the same time. Also, they may reply to any earlier move of the other player instead of having to reply to the last such move, and they may move alternative replies to the same move, possibly even in the same turn (a turn is a sequence of moves of one player). Other protocols defined in (Prakken 2005) impose restrictions on these points; one of them, concerning relevance of moves, will be briefly discussed later below.

The **topic language** and its **logic** are assumed to be some argumentation logic fitting the format of Dung's (1995) in which arguments can be formed by chaining deductive and defeasible inference rules into trees and in which arguments can be defeated on their use of defeasible inference rules. Dialogues are between a *proponent* P and *opponent* O of a single *dialogue topic* $t \in L_t$. The protocol is based on the following ideas. Each dialogue move except the initial one replies to one earlier move in the dialogue of the other party (its *target*). Thus a dialogue can be regarded in two ways: as a sequence (reflecting the order in which the moves are made) and as a tree (reflecting the reply relations between the moves). Each replying move is either an *attacker* or a *surrender*. For instance, a *claim* p move can be attacked with a *why* p move and surrendered with a *concede* p move; and a *why* p move can be attacked with an *argue* A move where A is an argument with conclusion p, and surrendered with a *retract* p move. When s is a surrendering and s' is an attacking reply to s'', we say that s' is an *attacking counterpart* of s. The **communication language** L_c is specified in Table 2. In this table, φ is from L_t and arguments A and

Acts	Attacks	Surrenders
claim φ	*why* φ	*concede* φ
why φ	*argue* A ($conc(A) = \varphi$)	*retract* φ
argue A	*why* φ ($\varphi \in prem(A)$), *argue* B (B defeats A)	*concede* φ ($\varphi \in prem(A)$ or $\varphi = conc(A)$)
concede φ		
retract φ		

Table 2: Reply structure

B are well-formed arguments from $\mathcal{L}$, while defeat relations between arguments are determined according to $\mathcal{L}$.

The protocol for L_c is defined in terms of the notion of a **dialogue**, which in turn is defined with the notion of a **move**.

- The set M of *moves* is defined as $\mathbb{N} \times \{P, O\} \times L_c \times \mathbb{N}$, where the four elements of a move m are denoted by, respectively:
  - $id(m)$, the *identifier* of the move,
  - $pl(m)$, the *player* of the move,
  - $s(m)$, the *speech act* performed in the move,
  - $t(m)$, the *target* of the move.
- The set of finite *dialogues*, denoted by $M^{<\infty}$, is the set of all finite sequences $m_1, \ldots, m_i, \ldots$ from M such that
  - each i^{th} element in the sequence has identifier i,
  - $t(m_1) = 0$;
  - for all $i > 1$ it holds that $t(m_i) = j$ for some m_j preceding m_i in the sequence.

  For any dialogue $d = m_1, \ldots, m_i, \ldots$ the sequence $m_1, \ldots, m_i$ is denoted by d_i, where d_0 denotes the empty dialogue.

When $t(m) = id(m')$ we say that m replies to m' in d and that m' is the target of m in d. We sometimes slightly abuse notation and let $t(m)$ denote a move instead of just its identifier. When $s(m)$ is an attacking (surrendering) reply to $s(m')$ we also say that m is an attacking (surrendering) reply to m'.

Protocols are in (Prakken 2005) defined as follows. A *protocol* on M is a set $P \subseteq M^{<\infty}$ satisfying the condition that whenever d is in P, so are all initial sequences that d starts with. A partial function $Pr : M^{<\infty} \longrightarrow \mathcal{P}(M)$ is derived from P as follows:

- $Pr(d) =$ undefined whenever $d \notin P$;
- $Pr(d) = \{m \mid d, m \in P\}$ otherwise.

The elements of *dom*(Pr), the domain of Pr, are called the *legal finite dialogues*. The elements of $Pr(d)$ are called the moves allowed after d. If d is a legal dialogue and $Pr(d) = \emptyset$, then d is said to be a *terminated* dialogue.

Our present **protocol** for L_c is now defined in Table 3.

For all moves m it holds that $m \in Pr(d)$ if and only if m satisfies all of the following rules:

- R$_1$: $pl(m) \in T(d)$;[1]
- R$_2$: If $d \neq d_0$ and $m \neq m_1$, then $s(m)$ is a reply to $s(t(m))$ according to L_c.
- R$_3$: If m replies to m' then $pl(m) \neq pl(m')$.
- R$_4$: If there is an m' in d such that $t(m) = t(m')$ then $s(m) \neq s(m')$.
- R$_5$: For any $m' \in d$ that surrenders to $t(m)$, m is not an attacking counterpart of m'.
- R$_6$: If $d = \emptyset$ then $s(m)$ is of the form *claim* φ or *argue* A.
- R$_7$: If m concedes the conclusion of an argument moved in m' then m' does not reply to a *why* move.

Table 3: The protocol for L_c.

[1] $T(d)$ denotes the player(s) whose turn it is to move in d.

R_1 says that the player of a move must be to move. (T returns for each dialogue the player(s) to move.) R_2-R_4 formalise the idea of a dialogue as a move-reply structure that allows for alternative replies. R_5 says that once a move is

surrendered, it may not be attacked any more. R_6 says that each dialogue begins with a claim or an argument; the initial claim or the conclusion of the initial argument is the *topic* of the dialogue. Finally, R_7 ensures that statements, when conceded, are conceded as claims or premises instead of as conclusions of arguments whenever possible.

The **commitment rules** of the protocol define the effects of a move on the players' commitment sets. As is well known, the agents' commitments should be carefully distinguished from their beliefs: commitments are an agent's publicly declared or accepted points of view, known to the other players, while beliefs are only known to the agent holding them; these beliefs may well be inconsistent with the agent's commitments. The commitment rules assign to each player-dialogue pair a (possibly empty) set of formulas from L_t. Below $C_{pl}(d, m)$ denotes the commitment set of player pl in dialogue d as continued with move m. At the beginning of the dialogue, the commitment sets of both players are empty and then they are updated according to the following rules.

- If $s(m) = \mathit{claim}\ \varphi$ then $C_{pl}(d, m) = C_{pl}(d) \cup \{\varphi\}$
- If $s(m) = \mathit{why}\ \varphi$ then $C_{pl}(d, m) = C_{pl}(d)$
- If $s(m) = \mathit{concede}\ \varphi$ then $C_{pl}(d, m) = C_{pl}(d) \cup \{\varphi\}$
- If $s(m) = \mathit{retract}\ \varphi$ then $C_{pl}(d, m) = C_{pl}(d) \setminus \{\varphi\}$
- If $s(m) = \mathit{argue}\ A$ then $C_{pl}(d, m) = C_{pl}(d) \cup \mathit{prem}(A) \cup \{\mathit{conc}(A)\}$

The **turntaking rule** is very liberal: the proponent starts with making a single move, then the turn switches to the opponent and after her first move it is both player's turn. Thus at any time after the second move both players can make an utterance, except that they cannot speak at the same time. In (Prakken 2005) several more strict turntaking rules are also defined and in (Bodenstaff 2005) one of them is formalised, viz. the turntaking rule for so-called relevant dialogues: in such dialogues the turn switches to the hearer after the speaker has succeeded in making the 'current state' of the dialogue favour his position (see also below).

Finally, **termination** was above implicitly defined as the situation where the player(s) to move cannot make a legal move. This means that to impose some desired termination condition (e.g. that the opponent has become committed to the dialogue topic or the proponent is not committed to the dialogue topic any more) the protocol should be defined such that there are no legal moves after the condition is satisfied.

The Event Calculus

The Event Calculus (EC) is a theory specified in first-order logic about events and their effects on states-of-affairs in the world (called 'fluents' in EC). EC was originally developed by Kowalski & Sergot (1986); in this paper we use a variant of Shanahan (1999) called the 'Full Event Calculus'. Its axioms express principles like 'If an event happens at time T that initiates a some fluent then that fluent starts to hold at T' and 'If a fluent holds at T and nothing terminates it then it also holds at $T + 1$'. The latter is commonly called the 'law of inertia'; it can be overruled by axioms expressing when a fluent is terminated. In applications of EC its general axioms must be supplemented with domain-specific axioms.

Our use of EC for the specification of dialogue protocols is motivated by the fact that a dialogue game can be seen as a dynamic system where dialogue utterances are events that initiate and terminate various aspects of the 'dialogical world', such as a player being the player-to-move or not, a player being committed to a certain proposition or not and a move being legal or not. Such aspects will be modelled as fluents, the value of which can change as an effect of utterances made during the dialogue.

By itself EC is just a first-order theory about actions and their effects. To fully capture the law of inertia it needs to be extended with nonmonotonic features. Shanahan (1999) adds a circumscription policy to the EC in order to reason with EC in circumscription. We instead follow Kowalski & Sergot's original logic-programming approach, since this allows the modelling of temporal persistence of fluents through negation-as-failure: if termination of a fluent cannot be derived, it can be assumed to persist. Thus, for instance, a proposition added to a player's commitments can be assumed to remain a commitment until this is explicitly terminated. Also, it can be elegantly modelled that the present protocol allows replies to any earlier move in the dialogue and not only to the last move. This is modelled by the fact that the legality of a reply persists until it is explicitly terminated.

To be able to reason about fluents, they are in EC *reified*. Reification means that the fluents are treated as first-class objects so that they can be used as arguments of predicates. For example, the sentence 'at time point 2 it is the turn of player *P*' can be represented as follows.

$$\mathit{HoldsAt}(\mathit{Turn}(P), 2)$$

Here the formula *Turn(P)* is reified as a term to allow it to be an argument of the predicate $\mathit{HoldsAt}$.

The axioms of the Full EC make use of a number of special predicates. We now describe their meaning informally and indicate how they can be used in the specification of dialogue games. The first two predicates concern the effects of an action on the value of a fluent.

Initiates(α, β, τ) means that fluent β starts to hold after event α at time τ. This formula will be used in the following ways. Firstly, it will be used to express the addition of a statement to a player's commitment set in effect of an utterance. For instance,

$$\mathit{Initiates}(\mathit{move}(1, P, \mathit{claim}\ q, 0), \mathit{CS}(P, q), 1)$$

says that proponent's claim of q in his first move adds q to his commitments (the move identifier following the speech act is the move's target, in this case the dummy value 0 to express that the claim is the dialogue's first move). In a similar way it can be expressed that in effect of an utterance another move becomes legal. For instance,

$$\begin{aligned}&\mathit{Initiates}(\\ &\quad \mathit{move}(1, P, \mathit{claim}\ q, 0),\\ &\quad \mathit{Legal}(\mathit{move}(id, O, \mathit{why}\ q, id), t))\end{aligned}$$

says that a claiming of q in the proponent's first move initiates the legality of a challenge of q by the opponent. Finally, the fact that a move makes a player the player-to-move can be expressed in a similar way.

Terminates(α, β, τ) means that fluent β ceases to hold after event α at time τ. This predicate is the 'mirror predicate' of *Initiates*: it can be used in an analogous way as that predicate for expressing the deletion of a commitment, the termination of legality of a move and the termination of a player being the one to move.

At the beginning of a dialogue certain fluents will hold and certain fluents will not hold. The following two predicates can be used to express the begin situation of a dialogue.

Initially$_P(\beta)$ means that fluent β holds at the beginning of the dialogue. This formula will be used for defining the commitment set of the players and the legal moves at the beginning of the dialogue.

Initially$_N(\beta)$ means that fluent β does not hold at the beginning of the dialogue. The formula will be used to define which moves are not legal at the beginning of the dialogue, which player is not allowed to make a move and which propositions a player is not committed to.

HoldsAt(β, τ) means that fluent β holds at time point τ and is used to express that a fluent holds at a certain time point. For instance, the formula

$$HoldsAt(Legal(move(4, O, why\ q, 1), 4))$$

expresses that a challenge of q by the opponent is legal at move 4 as a reply to move 1. Such formulas will (often with variables) be used in the conditions of the rules for move legality, turntaking and termination.

Happens(α, τ_1, τ_2) means that event α starts at time point τ_1 and ends at time point τ_2. This predicate will be used to express all moves made during the dialogue. Since a dialogue move is assumed to have no duration, it will always hold that $\tau_1 = \tau_2$.

Besides persistence of a fluent, it must be possible to express *termination* and *initiation* of fluents at certain time points. This can be done with the predicates *Clipped* and *Declipped*; they are used in the general axioms of EC but we will not use them in our domain-specific axioms. *Clipped*$(\tau 1, \beta, \tau_2)$ means that fluent β is terminated between times τ_1 and τ_2. *Declipped*(τ_1, β, τ_2) means that fluent β is initiated between times τ_1 and τ_2.

We next list the general axioms of the full EC. Following convention, variables are assumed to be implicitly universally quantified. The unique-name axioms, which are part of EC, are left implicit, as well as the usual definitions of (in)equality. The first two axioms concern the conditions that should be met in order for a fluent to *persist*. The third axiom concerns the conditions for a fluent to *terminate to hold*. Then three axioms are presented which express exactly the opposite. The fourth axiom expresses when a fluent does *not persist* and the fifth and sixth axiom express what conditions should be met for a fluent to *start to hold*. The last axiom ensures that an event takes a non-negative amount of time. Note that in our Prolog implementation the occurrences of classical negation $\neg$ in the conditions of the axioms will be implemented as negation-as-failure, to capture the law of inertia.

1. $HoldsAt(f, t) \leftarrow Initially_P(f) \wedge \neg Clipped(0, f, t)$
   This axiom states that if a fluent initially holds and is not terminated between time point 0 and time point t then the fluent still holds at time point t.
2. $HoldsAt(f, t_3) \leftarrow Happens(a, t_1, t_2) \wedge Initiates(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg Clipped(t_1, f, t_3)$
   This axiom states that if event a which initiates fluent f occurs then fluent f starts to hold until fluent f is terminated.
3. $Clipped(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3(Happens(a, t_2, t_3) \wedge (t_1 < t_3) \wedge (t_2 < t_4) \wedge Terminates(a, f, t_2))$
   This axiom states that if and only if there exists an event a which occurs and terminates fluent f then fluent f is said to be *clipped*.
4. $\neg HoldsAt(f, t) \leftarrow Initially_N(f) \wedge \neg Declipped(0, f, t)$
   This axiom states that if a fluent did not initially hold and was not initiated between time point 0 and time point t then the fluent does not hold at time point t.
5. $\neg HoldsAt(f, t_3) \leftarrow Happens(a, t_1, t_2) \wedge Terminates(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg Declipped(t_1, f, t_3)$
   This axiom states that if event a which terminates fluent f occurs then fluent f does not hold as long as fluent f is not initiated.
6. $Declipped(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3(Happens(a, t_2, t_3) \wedge (t_1 < t_3) \wedge (t_2 < t_4) \wedge Initiates(a, f, t_2))$
   This axiom states that if and only if there exists an event a which occurs and initiates fluent f then fluent f is said to be declipped.
7. $Happens(a, t_1, t_2) \rightarrow (t_1 \leq t_2)$
   This axiom ensures that the time an event takes can never be negative.

Formalising the P system

This section contains the main contribution of this article: our specification of the P system in EC. The following fluents will be used in addition to those of the general EC axioms.

- $move(id, p, s, tr)$ This fluent states that this is the id^{th} move where participant p states locution s targeted at tr. Initially, id will be equal to tr but they will diverge when a participant makes an illegal move: then no other fluents are changed but time moves with one unit, so the time point is raised while the move identifier is not.
- $Legal(m)$ This fluent expresses that move m is legal, where m is a tuple $move(id, p, s, tr)$. Fluents of this form hold initially for initial *claim* and *argue* moves by the proponent, while they are initiated for moves that are a well-formed reply to a certain locution, when a locution of that type is moved. So every move, when legal, initiates the legality of one or more moves as a reply to that move.
- $CS(p, \varphi)$ This fluent represents that participant p is committed to proposition φ. CS stands for commitment set; fluents of this form are initiated and terminated according to the commitment rules of the dialogue system.
- $Turn(p)$ is a fluent which is initiated when the turn shifts to participant p.
- *P* stands for proponent.

- *O* stands for opponent.
- *p* and $\bar{p}$ are defined as: $\bar{p} = O$ if and only if $p = P$ and $\bar{p} = P$ if and only if $p = O$.

Table 4 indicates to which parts of the dialogue system the various predicates pertain.

Predicates	BB
$Happens(move(id, p, s, tr), t)$	Comm. Language
$Initially_P(Legal(\varphi))$, $Initially_N(Legal(\varphi))$	Protocol
$Initiates(move(id, p, s, tr), Legal(\varphi), t)$, $Terminates(move(id, p, s, tr), Legal(\varphi), t)$	Protocol
$Initially_N(CS(p, \varphi))$	Commit. Rules
$Initiates(move(id, p, s, tr), CS(p, \varphi), t)$, $Terminates(move(id, p, s, tr), CS(p, \varphi), t)$	Commit. Rules
$Initially_P(Turn(p))$, $Initially_N(Turn(p))$	Turntaking
$Initiates(move(id, p, s, tr), Turn(p_2), t)$, $Terminates(move(id, p, s, tr), Turn(p_2), t)$	Turntaking

Table 4: Predicates used

A **dialogue** is specified as a sequence of unconditional *Happens* clauses. Accordingly, the predicate $Happens$ is to be instantiated with a fluent $move(id, p, s, tr)$. For example, $Happens(1, P, claim(q), 0)$ expresses that the proponent claimed q in the first move.

The formalisation of the **protocol** first specifies which initial moves are legal. Initially the only legal moves are a *claim* or an *argue* move by the proponent. After his first move the legality of these moves terminates.

$$Initially_P(Legal(move(1, P, s, 0))) \leftarrow$$
$$s = claim\ \varphi \vee s = argue\ A$$

$$Initially_N(Legal(move(id, p, s, tr))) \leftarrow$$
$$p = O \vee$$
$$(id \neq 1) \vee$$
$$(s \neq claim\ \varphi \wedge s \neq argue\ A) \vee$$
$$(t \neq 0)$$

$$Terminates(move(1, P, s_1, 0), Legal(move(1, P, s_2, 0)), t)$$
$$\leftarrow$$
$$HoldsAt(Legal(move(1, P, s_1, 0)), t)$$

The next formulas specify how the legality of non-initial moves is initiated, capturing rules R_2, R_3, R_4 and R_7 of the protocol. $Defeats(B, A)$ means 'argument B defeats argument A'. Note that the rules for *argue* moves assume that the well-formedness of arguments and their defeat relations are determined by external means.

The general format of the legality-initiating rules is as follows:

m_1 initiates the legality of m_2 if
m_1 was moved legally, and
$pl(m_1)$ is the player-to-move, and
$s(m_2)$ is a well-formed reply to $s(m_1)$, and
the specific conditions for m_2, if any, are satisfied.

Actually, in the rule for replies to initial moves only the third condition needs to be stated:

$$Initiates(move(1, P, claim\ \ \varphi, 0), Legal(move(id, O, s, 1)), t)$$
$$\leftarrow$$
$$s = why\ \varphi \vee s = concede\ \varphi$$

The first rule for replies to non-initial moves also uses the first and second condition, while the second such rule uses all four conditions:

$$Initiates(move(id, p, why\ \varphi, tr), Legal(move(id_2, \bar{p}, s, id)), t)$$
$$\leftarrow$$
$$HoldsAt(Legal(move(id, p, why\ \varphi, tr)), t) \wedge$$
$$HoldsAt(Turn(p), t) \wedge$$
$$((s = argue\ A \wedge conc(A) = \varphi) \vee (s = retract\ \varphi))$$

$$Initiates(move(id, p, argue\ A, tr), Legal(move(id_2, \bar{p}, s, id)), t)$$
$$\leftarrow$$
$$HoldsAt(Legal(move(id, p, argue\ A, tr)), t) \wedge$$
$$HoldsAt(Turn(p), t) \wedge$$
$$((s = why\ \varphi \wedge \varphi \in prem(A))$$
$$\vee$$
$$(s = argue\ B \wedge Defeats(B, A))$$
$$\vee$$
$$(s = concede\ \varphi \wedge \varphi = conc(A) \wedge$$
$$\neg(Happens(move(tr, \bar{p}, why\ \varphi, tr_2), t_2) \wedge t_2 < t)$$
$$\vee$$
$$(s = concede\ \varphi \wedge \varphi \in prem(A)))$$

Next the conditions are specified under which the legality of non-initial moves terminates. The first rule below says that after a move with a specific content is made, it terminates to hold as a legal move with that specific content and the same target.

$$Terminates(move(id, p, s, tr), Legal(move(id_2, p, s, tr)), t)$$
$$\leftarrow$$
$$HoldsAt(Turn(p), t) \wedge$$
$$HoldsAt(Legal(move(id, p, s, tr)), t)$$

The second rule captures protocol rule R_5 and states that when a move is a surrendering move to a target the attacking counterpart of this move at the same target terminates to be legal.

$$Terminates(move(id, p, s_1, tr), Legal(move(id_2, p, s_2, tr)), t)$$
$$\leftarrow$$
$$HoldsAt(Legal(move(id, p, s_1, tr)), t) \wedge$$
$$HoldsAt(Legal(move(id_2, p, s_2, tr)), t) \wedge$$
$$HoldsAt(Turn(p), t) \wedge$$
$$((s_1 = concede\ \varphi \wedge s_2 = why\ \varphi)$$
$$\vee$$
$$(s_1 = concede\ \varphi \wedge s_2 = argue\ A \wedge \neg\varphi = conc(A))$$
$$\vee$$
$$(s_1 = retract\ \varphi \wedge s_2 = argue\ A \wedge \varphi = conc(A)))$$

Summarising, EC's 'law of inertia' is used in this formalisation as follows. Initially, only a *claim* and *argue* move are legal and all other moves are illegal. After a move is made its legality is terminated (but only with that specific content) and the legality of well-formed replies to that move's speech act is initiated. Such legality persists until the move is made. The illegality of moves persists until its legality is initiated as just described.

As for the **commitment rules**, at the start of the dialogue the commitment sets of both players are empty. When a move is made, the speaker's commitments are updated according to the commitment rules of the P system.

$Initially_N(CS(p, \varphi))$

$Initiates(move(id, p, s, tr), CS(p, \varphi), t) \leftarrow$
$HoldsAt(Legal(move(id, p, s, tr)), t) \wedge$
$HoldsAt(Turn(p), t) \wedge$
$(s = claim\ \varphi \vee s = concede\ \varphi)$

$Terminates(move(id, p, retract\ \varphi, tr), CS(p, \varphi), t) \leftarrow$
$HoldsAt(Legal(move(id, p, retract\ \varphi, tr)), t) \wedge$
$HoldsAt(Turn(p), t)$

$Initiates(move(id, p, argue\ A, tr), CS(p, \varphi), t) \leftarrow$
$HoldsAt(Legal(move(id, p, argue\ A, tr)), t) \wedge$
$HoldsAt(Turn(p), t) \wedge$
$(\varphi = conc(A) \vee \varphi = prem(A))$

The **turntaking** rules are that the first move is always made by the proponent after which the turn switches to the opponent. After the second move, the turn of the proponent is initiated but that of the opponent does not terminate.

$Initially_P(Turn(P))$

$Initially_N(Turn(O))$

$Terminates(move(1, P, s, 0), Turn(P), t) \leftarrow$
$HoldsAt(Legal(move(1, P, s, 0)), t)$

$Initiates(move(id, p_1, s, tr), Turn(p_2), t) \leftarrow$
$HoldsAt(Legal(move(id, p_1, s, tr)), t) \wedge$
$((id = 1 \wedge p_1 = P \wedge tr = 0 \wedge t = 1 \wedge p_2 = O)$
$\vee$
$(id = 2 \wedge p_1 = O \wedge tr = 1 \wedge t = 2 \wedge p_2 = P))$

A Prolog implementation

We next briefly describe a Prolog implementation of our formalisation. The source code is available at `www.ewi.utwente.nl/~bodenstaffl`. The implementation computes in any state of a dialogue the players' commitments, whether the moves made were legal, who is to move and what are the legal next moves. It can thus be used as a 'dialogue consultant' by a player, referee or external observer. A Prolog program consists of a database of clauses, which is essentially a set of if-then rules 'head if body' where the head is a first-order atom and the body a disjunction of conjunctions of literals. Facts can be specified with clauses with empty body. Variables in Prolog are written as capitals and anonymous variables are written as underscores. Constants are denoted by lowercase letters or numbers. A typical clause is of the form $\mathtt{A} :- \mathtt{B}, \mathtt{C}$ which means 'if B and C then A'. Logical 'or' is written as a semicolon.

In Prolog, negation is interpreted as 'negation as failure' which means that a negated atom holds if the atom itself cannot be derived. Since our EC formalisation contains classical negations, we first need to convert our formalisation to a general logic program. This can be done as follows. The negations in the conditions of the general EC axioms can safely be translated as negation-as-failure to capture the law of inertia. Furthermore, the negations in the conditions of our specific axioms can be translated as negation-as-failure since in the current application the closed-world assumption can safely be made: a dialogue state can be completely specified, so what is not specified can be assumed false. However, for classical negations in the consequents of axioms a special predicate not_holds_at has to be introduced and the program has to be designed such that for no fluent both $holds\_at(F, T)$ and $not\_holds\_at(F, T)$ can be derived (in our program this was straightforward). The resulting general logic program can then easily be implemented in a Prolog program.

The implementation of, for example, Axiom 4 now is as follows.

$\mathtt{not\_holds\_at(F,T)} :-$
$\mathtt{initially\_n(F)},$
$\backslash + \mathtt{declipped(0,F,T)}.$

(where $\backslash +$ denotes negation as failure). When consulting a program through the Prolog interpreter, queries can be entered. A query is, for example, $? - \mathtt{initiates(move(1,p,claimq,0),turn(p),2)}$. Prolog will try to match facts in the database with the query. If this attempt fails Prolog will try to find rules where the conclusion matches the query. Prolog will assign the values of the query to the variables in the rule . Next Prolog will try to satisfy all variables in the premises. Besides queries where Prolog is only able to answer 'yes' or 'no' it is also possible to leave anonymous variables in the query. Now Prolog will return a proper assignment to the variables if there is one. After returning the first set of possible values, the user of the interpreter can enter a semicolon after which Prolog will give another assignment if there is one, else Prolog will return 'no'.

An argument in the implementation is represented as $\mathtt{since(p,s)}$ and should be read as '$\mathtt{p}$ since $\mathtt{s}$'. The support for proposition $\mathtt{p}$, namely $\mathtt{s}$, should be entered as a list. In Prolog a list is of the form: $[\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots]$. In our test runs no defeat relations where used but they could be added as `Initially_p` clauses. Note that, just as our above formalisation, our implementation assumes that arguments are created and defeat relations are established by some reasoner external to and combined with the Prolog program.

The following queries are useful for planning the first move.

$\mathtt{holds\_at(turn(P),0)}.$
$\mathtt{holds\_at(cs(P,A),0)}.$
$\mathtt{holds\_at(legal(move(1,P,S,0)),0)}.$

In return to these queries Prolog will show whose turn it is, what the commitments of the participants are and which moves are legal at time point 0. When the dialogue proceeds these queries can be used with different time points to plan the following move and to check the effects of moves on the players' commitments. These queries can

also be entered with constants and in negated form, as in `not_holds_at(legal(move(3,o,claim q,2)),4)`. To enter a move the `assert` command is used. `assert` is a built-in predicate which adds its argument as a fact to the program. Asserting a clause always succeeds in Prolog.

We now demonstrate the program by simulating the following dialogue according to the P system. The knowledge base of the proponent is $\{z; q; z, q \rightarrow a\}$ and the one of the opponent is $\{d; d \rightarrow c\}$.

Time point	Move	CS_P	CS_O
T_1	$move(1, P, claim\ a, 0)$	$\{a\}$	
T_2	$move(2, O, why\ a, 1)$	$\{a\}$	
T_3	$move(3, P, argue\ A, 2)$ $conc(A) = a$, $prem(A) = z, q$	$\{a, z, q\}$	
T_4	$move(4, O, argue\ B, 3)$ $conc(B) = c$, $prem(B) = d$	$\{a, z, q\}$	$\{c, d\}$
T_5	$move(5, P, why\ d, 4)$	$\{a, z, q\}$	$\{c, d\}$
T_6	$move(6, O, retract\ d, 5)$	$\{a, z, q\}$	$\{c, d\}$

After loading the program the fist move by the proponent is entered.

```
? − assert(happens(move(1,p,claim(a),0),1)).
Yes
?−
```

To plan the next move, a query is entered to find out which participant is allowed to utter which locution.

```
? − holds_at(legal(move(2,P,S,1)),2).
P = o
S = why(a);
P = o
S = concede(a);
No
?−
```

Only the opponent is allowed to make a move and its only legal locutions are `why a` and `concede a`. We proceed by asserting the next move. After that also the remaining part of the dialogue is given as it is simulated in Prolog.

```
? − holds_at(legal(move(3,P,A,2)),3).
P = p
A = argue(since(a,_G511));
P = p
A = retract_(a);
No
? − assert(happens(move(3,p,argue(since(a,[z,q])),2),3)).
Yes
? − holds_at(legal(move(4,P,A,3)),4).
P = o
A = why(z);
P = o
A = why(q);
P = o
A = argue(_G481);
No
? − assert(happens(move(4,o,argue(since(c,[d])),3),4)).
Yes
? − holds_at(legal(move(5,P,S,4)),5).
P = p
S = concede(d);
P = p
S = why(d);
P = p
S = argue(_G481);
P = p
S = concede(c);
No
? − assert(happens(move(5,p,why(d),4),5)).
Yes
? − holds_at(legal(move(6,P,S,5)),6).
P = o
S = argue(since(d,_G660));
P = o
S = retract_(d);
No
? − assert(happens(move(6,o,retract_(d),5),6)).
Yes
? − holds_at(legal(move(7,P,A,N)),7).
P = o
A = concede(a)
N = 1;
P = p
A = retract_(a)
N = 2;
P = o
A = why(z)
N = 3;
P = o
A = why(q)
N = 3;
P = o
A = concede(z)
N = 3;
P = o
A = concede(q)
N = 3;
P = p
A = argue(_G500)
N = 4;
P = p
A = concede(c)
N = 4;
P = p
A = concede(d)
N = 4;
No
?−
```

It should be noted that two other moves are legal at time point 7 but are not returned by Prolog in the last question `holds_at(legal(move(7,P,A,N)),7)`. These moves are `move(7,p,argue(since(a,Phi)),2)` where `Phi` cannot be `[z,q]` and `move(7,o,argue(Psi),3)` where `Psi` cannot be

[d] because the moves with these propositions already happened at time point 3 and 4 respectively. Prolog fails to return these two clauses because not *all* available propositions are legal to use in these *argue* moves. However, when a specific query is entered, for example, $\texttt{holds\_at}(\texttt{legal}(\texttt{move}(7, \texttt{p}, \texttt{argue}(\texttt{since}(\texttt{a}, \texttt{b})), 2)), 7)$, Prolog will answer affirmatively.

Other case studies

To test the generality of the above approach we applied it in (Bodenstaff 2005) to two other protocols. Firstly, we extended the above formalisation of the P system to a stricter instantiation of the framework of (Prakken 2005), in which all moves have to satisfy a condition of relevance. This notion is defined in terms of a notion of dialogical status of a move. Briefly, a move is *in* if it is surrendered or else all its attacking replies are *out*, and a move is *out* if it has an attacking reply that is *in*. A move is *relevant* if changing the dialogical status of its target also changes the dialogical status of the initial move; the turn shifts as soon as the dialogical status of the initial move has changed. In consequence, each turn now consists of zero or more surrenders followed by zero or one attacker (if zero, then the dialogue terminates automatically). We straightforwardly added this to our formalisation of P by adding definitions of dialogical status and relevance, changing the turntaking rule and giving all rules that initiate move legality a single extra condition that the move is relevant.

Secondly, we applied our approach to the system for persuasion dialogue of Parsons, Wooldridge, & Amgoud (2003). The communication language of the PWA system was displayed above in Table 1 (which renames some terms of PWA for ease of comparison). The main differences with the P system are the absence of a locution for retractions and the replacement of the *argue* locution with a *claim* S locution for a set S of propositions; the latter is to be moved in reply to a *why* φ move. The explicit reply structure of the P system is implicitly built into PWA's protocol, where in addition a claim of a proposition can also be answered with a claim of its negation. PWA's protocol rules refer to the players' internal states, in requiring that their claims and concessions must respect their "assertion and acceptance attitudes". For instance, an agent with a "sceptical" assertion attitude may claim a proposition only if he can construct a justified argument for it on the basis of his own knowledge. As for the structural aspects of a dialogue, the PWA protocol is much stricter than that of P: essentially, in PWA the turn shifts after every move and no alternative replies to a move are allowed, except to a *claim* S move, of which each member of S can be challenged or conceded in turn. PWA's commitment rules are essentially the same as those of P. A dialogue terminates when a player is to move but can make no legal moves; in that case he has to "concede the game".

In formalising PWA in EC, most of our domain-specific EC predicates turned out to be useful but some new predicates had to be added to deal with assertion and concession attitudes and with the stricter structural nature of the protocol. These features also required several new axioms and modification of existing axioms. Nevertheless, our second case study provides support for the generality of our approach.

Related work

As for related work, three publications are particularly relevant to our investigations. Firstly, Yolum & Singh (2004) apply the event calculus to reasoning about commitments to action in trading scenarios. However, they do not address argumentation and do not model reasoning about legality of dialogue moves.

Brewka (2001) reconstructs and then formalises an argumentation protocol of Rescher (1977) in a version of the situation calculus (Reiter 2001). The Rescher/Brewka system is in some respects simpler and in other respects more complex than the P system. Its underlying logic is default logic and its communication language has no explicit reply structure. Arguments are implicitly moved as claim replies to challenges. Players may make any new claim and may retract their own commitments and challenge or concede the other player's commitments at any time. The system has no turntaking rules. Dialogues terminate by convention, after which a determiner is allowed to declare one of the players the winner in a way constrained by the players' final commitments. Since Brewka distinguishes 'possible' from 'legal' moves, players can move illegal moves but then the other player can reply with an *object* locution, after which the effects of the illegal move are undone. Brewka's work was a source of inspiration for our investigations but because of the differences in dialogue systems and formalisation languages it is difficult to give a precise comparison, especially since EC and situation calculus are rather different in style.

Perhaps the closest to the present investigations is the work of Artikis, Sergot, & Pitt (2003), who formalise a modified variant of the Rescher/Brewka system in the $C+$ language of Giunchiglia *et al.* (2004) and then implement it in Giunchiglia *et al.*'s "causal calculator". The $C+$ language is closer to EC than the situation calculus (in fact, Artikis, Pitt, & Sergot (2002) used EC to formalise the contract net protocol). Artikis, Sergot, & Pitt refine Brewka's notion of legal dialogue moves into a distinction between 'permitted' and 'valid' moves. A move is valid if its player has the "power" to move it, i.e., if uttering certain words 'counts as' a certain speech act. As is well-known from legal theory, powers and permissions are logically unrelated; our 'legality' of moves corresponds to Artikis, Sergot, & Pitt's validity of moves but our approach could incorporate their refinements if required by the formalised dialogue system. The same holds for their formalisations of *object* moves and the determiner's role.

Conclusion

In this paper we investigated the suitability of the Event Calculus for logical formalisation of dialogue systems for argumentation and subsequent implementation in declarative programming languages. We have contributed to previous work in this direction as follows. Generally speaking, by applying EC to systems with some new features, our case studies have confirmed and reinforced the earlier findings that

EC (and similar formalisms) are suitable for logical formalisation of dialogue systems for argumentation. More specifically, we have made the following contributions. Firstly, while in the dialogue systems studied by Brewka (2001) and Artikis, Sergot, & Pitt (2003) the protocol is mainly defined in terms of the players' commitments, we have focussed on systems in which it is largely defined in terms of an explicit reply structure of the communication language, with a distinction between attacking and surrendering replies. Secondly, we have focussed on systems with a more liberal dialogue structure and with varying turntaking rules. Finally, we have (in more detail in (Bodenstaff 2005)) shown how the constraining of dialogues by a notion of relevance can be formalised.

Our Prolog implementation of the P system illustrated that the formalisation of dialogue systems in a logical calculus supports their declarative implementation. Such implementation in turn arguably supports the design of flexible dialogue systems, in which variations in the communication language or protocol can be handled easier than when they are hard-coded in a lower-level programming language. Of course, whether these advantages over, for instance, functional specifications indeed hold must still be investigated but the present paper has helped in making such investigations possible by further developing the logic-based approach. Another research topic is the design of intelligent agents who interact within a persuasion protocol. Agents could use a declarative implementation of a dialogue system to reason about the moves they are allowed to make and their effects, and use some internal decision-making or planning mechanism to choose from the available moves.

Acknowledgements

The research reported in this paper was partially supported by the EU under IST-FP6-002307 (ASPIC).

References

Amgoud, L.; Maudet, N.; and Parsons, S. 2000. Modelling dialogues using argumentation. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, 31–38.

Artikis, A.; Pitt, J.; and Sergot, M. 2002. Animated specifications of computational societies. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, 1053–1062.

Artikis, A.; Sergot, M.; and Pitt, J. 2003. An executable specification of an argumentation protocol. In *ICAIL '03: Proceedings of the Ninth International Conference on Artificial Intelligence and Law*, 1–11. New York, NY, USA: ACM Press.

Atkinson, K.; Bench-Capon, T.; and McBurney, P. 2005. A dialogue game protocol for multi-agent argument over proposals for action. *Journal of Autonomous Agents and Multi-Agent Systems* 11:153–171.

Bodenstaff, L. 2005. Formalisation of argumentation protocols in event calculus. Master's thesis, Utrecht University. http://www.ewi.utwente.nl/˜bodenstaffl.

Brewka, G. 2001. Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *Journal of Logic and Computation* 11(2):257–282.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–357.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153:49–104.

Gordon, T. 1994. The Pleadings Game: an exercise in computational dialectics. *Artificial Intelligence and Law* 2:239–292.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4(1):67–95.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2003. Properties and complexity of some formal inter-agent dialogues. *Journal of Logic and Computation* 13. 347-376.

Prakken, H. 2005. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation* 15:1009–1040.

Rahwan, I.; Ramchurn, S.; Jennings, N.; McBurney, P.; Parsons, S.; and Sonenberg, L. 2003. Argumentation-based negotiation. *The Knowledge Engineering Review* 18:343–375.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. Cambridge, MA, USA: MIT Press.

Rescher, N. 1977. *Dialectics: A Controversy-Oriented Approach to the Theory of Knowledge*. Albany, NY, USA: State University of New York Press.

Shanahan, M. 1999. The event calculus explained. *Lecture Notes in Computer Science* 1600:409–430.

Yolum, P., and Singh, M. 2004. Reasoning about commitments in the Event Calculus: an approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence* 42:227–253.

5.2 Approximate Arguments for Efficiency in Logical Argumentation

Approximate Arguments for Efficiency in Logical Argumentation

Anthony Hunter
Department of Computer Science
University College London
Gower Stree, London WC1E 6BT, UK

Abstract

There are a number of frameworks for modelling argumentation in logic. They incorporate a formal representation of individual arguments and techniques for comparing conflicting arguments. A common assumption for logic-based argumentation is that an argument is a pair $\langle \Phi, \alpha \rangle$ where Φ is minimal subset of the knowledgebase such that Φ is consistent and Φ entails the claim α. Different logics are based on different definitions for entailment and consistency, and give us different options for argumentation. For a variety of logics, in particular for classical logic, the computational viability of generating arguments is an issue. Here, we propose ameliorating this problem by using approximate arguments.

Introduction

Argumentation is a vital aspect of intelligent behaviour by humans. Consider diverse professionals such as politicians, journalists, clinicians, scientists, and administrators, who all need to collate and analyse information looking for pros and cons for consequences of importance when attempting to understand problems and make decisions.

There are a number of proposals for logic-based formalisations of argumentation (for reviews see (Prakken & Vreeswijk 2000; Chesnevar, Maguitman, & Loui 2001)). These proposals allow for the representation of arguments for and against some claim, and for attack relationships between arguments. In a number of key examples of argumentation systems, an argument is a pair where the first item in the pair is a minimal consistent set of formulae that proves the second item which is a formula. Furthermore, in these approaches, the notion of attack is a form of undercut, where one argument undercuts another argument when the claim of the first argument negates the premises of the second argument.

In this paper, we consider how we can undertake argumentation more efficiently. Let us start by considering the construction of individual arguments. If Δ is a knowledgebase, and we are interested in a claim α, we look for an argument $\langle \Phi, \alpha \rangle$ where $\Phi \subseteq \Delta$. Deciding whether a set of propositional classical formulae is classically consistent is an NP-complete decision problem and deciding whether a set of propositional formulae classically entails a given formula is a co-NP-complete decision problem. However, if we consider the problem as an abduction problem, where we seek the existence of a minimal subset of a set of formulae that implies the consequent, then the problem is in the second level of the polynomial hierarchy (Eiter & Gottlob 1995). Even worse deciding whether a set of first-order classical formulae is consistent is an undecidable decision problem. So even finding the basic units of argumentation is computationally challenging.

Proof procedures and algorithms have been developed for finding preferred arguments from a knowledgebase following for example Dung's preferred semantics (see for example (Prakken & Sartor 1997; Kakas & Toni 1999; Cayrol, Doutre, & Mengin 2001; Dimopoulos, Nebel, & Toni 2002; Dung, Kowalski, & Toni 2006)). However, these techniques and analyses do not offer any ways of ameliorating the computational complexity inherent in finding arguments and counterarguments even though it is a significant source of computational inefficiency.

A possible approach to ameliorate the cost of entailment is to use approximate entailment: Proposed in (Levesque 1984), and developed in (Schaerf & Cadoli 1995), classical entailment is approximated by two sequences of entailment relations. Approximate entailment has been developed for anytime coherence reasoning (Koriche 2002). However, the approach still needs to be further developed and evaluated for finding arguments and counterarguments in argumentation. This would need to start with a conceptualization of the notions of argument and counteragument derived using approximate entailment and approximate coherence.

In this paper, we take a different approach by presenting a new solution that uses approximate arguments. First we review an existing version of logic-based argumentation in order to illustrate our ideas, and then we present our framework for approximate arguments.

Logical argumentation

In this section we review an existing proposal for logic-based argumentation (Besnard & Hunter 2001). We consider a classical propositional language with classical deduction denoted by the symbol $\vdash$. We use $\alpha, \beta, \gamma, \ldots$ to denote formulae and $\Delta, \Phi, \Psi, \ldots$ to denote sets of formulae.

For the following definitions, we first assume a knowledgebase Δ (a finite set of formulae) and use this Δ throughout. We further assume that every subset of Δ is given an enumeration $\langle \alpha_1, \ldots, \alpha_n \rangle$ of its elements, which we call its

canonical enumeration. This really is not a demanding constraint: In particular, the constraint is satisfied whenever we impose an arbitrary total ordering over Δ. Importantly, the order has no meaning and is not meant to represent any respective importance of formulae in Δ. It is only a convenient way to indicate the order in which we assume the formulae in any subset of Δ are conjoined to make a formula logically equivalent to that subset.

The paradigm for the approach is a large repository of information, represented by Δ, from which arguments can be constructed for and against arbitrary claims. Apart from information being understood as declarative statements, there is no a priori restriction on the contents, and the pieces of information in the repository can be as complex as possible. Therefore, Δ is not expected to be consistent. It need even not be the case that every single formula in Δ is consistent.

The framework adopts a very common intuitive notion of an argument. Essentially, an argument is a set of relevant formulae that can be used to classically prove some claim, together with that claim. Each claim is represented by a formula.

Definition 1. *An* **argument** *is a pair $\langle \Phi, \alpha \rangle$ such that: (1) $\Phi \subseteq \Delta$; (2) $\Phi \nvdash \bot$; (3) $\Phi \vdash \alpha$; and (4) there is no $\Phi' \subset \Phi$ such that $\Phi' \vdash \alpha$. We say that $\langle \Phi, \alpha \rangle$ is an argument for α. We call α the* **claim** *of the argument and Φ the* **support** *of the argument (we also say that Φ is a support for α).*

Example 1. *Let $\Delta = \{\alpha, \alpha \rightarrow \beta, \gamma \rightarrow \neg\beta, \gamma, \delta, \delta \rightarrow \beta, \neg\alpha, \neg\gamma\}$. Some arguments are:*

$$\langle \{\alpha, \alpha \rightarrow \beta\}, \beta \rangle$$
$$\langle \{\neg\alpha\}, \neg\alpha \rangle$$
$$\langle \{\alpha \rightarrow \beta\}, \neg\alpha \vee \beta \rangle$$
$$\langle \{\neg\gamma\}, \delta \rightarrow \neg\gamma \rangle$$

Arguments are not independent. In a sense, some encompass others (possibly up to some form of equivalence). To clarify this requires a few definitions as follows.

Definition 2. *An argument $\langle \Phi, \alpha \rangle$ is* **more conservative** *than an argument $\langle \Psi, \beta \rangle$ iff $\Phi \subseteq \Psi$ and $\beta \vdash \alpha$.*

Example 2. *$\langle \{\alpha\}, \alpha \vee \beta \rangle$ is more conservative than $\langle \{\alpha, \alpha \rightarrow \beta\}, \beta \rangle$.*

Some arguments directly oppose the support of others, which amounts to the notion of an undercut.

Definition 3. *An* **undercut** *for an argument $\langle \Phi, \alpha \rangle$ is an argument $\langle \Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n) \rangle$ where $\{\phi_1, \ldots, \phi_n\} \subseteq \Phi$.*

Example 3. *Let $\Delta = \{\alpha, \alpha \rightarrow \beta, \gamma, \gamma \rightarrow \neg\alpha\}$. Then, $\langle \{\gamma, \gamma \rightarrow \neg\alpha\}, \neg(\alpha \wedge (\alpha \rightarrow \beta)) \rangle$ is an undercut for $\langle \{\alpha, \alpha \rightarrow \beta\}, \beta \rangle$. A less conservative undercut for $\langle \{\alpha, \alpha \rightarrow \beta\}, \beta \rangle$ is $\langle \{\gamma, \gamma \rightarrow \neg\alpha\}, \neg\alpha \rangle$.*

Definition 4. *$\langle \Psi, \beta \rangle$ is a* **maximally conservative undercut** *of $\langle \Phi, \alpha \rangle$ iff $\langle \Psi, \beta \rangle$ is an undercut of $\langle \Phi, \alpha \rangle$ such that no undercuts of $\langle \Phi, \alpha \rangle$ are strictly more conservative than $\langle \Psi, \beta \rangle$ (that is, for all undercuts $\langle \Psi', \beta' \rangle$ of $\langle \Phi, \alpha \rangle$, if $\Psi' \subseteq \Psi$ and $\beta \vdash \beta'$ then $\Psi \subseteq \Psi'$ and $\beta' \vdash \beta$).*

The value of the following definition of canonical undercut is that we only need to take the canonical undercuts into account. This means we can justifiably ignore the potentially very large number of non-canonical undercuts.

Definition 5. *An argument $\langle \Psi, \neg(\phi_1 \wedge \ldots \wedge \phi_n) \rangle$ is a* **canonical undercut** *for $\langle \Phi, \alpha \rangle$ iff it is a maximally conservative undercut for $\langle \Phi, \alpha \rangle$ and $\langle \phi_1, \ldots, \phi_n \rangle$ is the canonical enumeration of Φ.*

An argument tree describes the various ways an argument can be challenged, as well as how the counter-arguments to the initial argument can themselves be challenged, and so on recursively.

Definition 6. *A* **complete argument tree** *for α is a tree where the nodes are arguments such that*

1. *The root is an argument for α.*
2. *For no node $\langle \Phi, \beta \rangle$ with ancestor nodes $\langle \Phi_1, \beta_1 \rangle, \ldots, \langle \Phi_n, \beta_n \rangle$ is Φ a subset of $\Phi_1 \cup \cdots \cup \Phi_n$.*
3. *The children nodes of a node N consist of all canonical undercuts for N that obey 2.*

The second condition in Definition 6 ensures that each argument on a branch has to introduce at least one formula in its support that has not already been used by ancestor arguments. This is meant to avoid making explicit undercuts that simply repeat over and over the same reasoning pattern except for switching the role of some formulae (e.g. in mutual exclusion, stating that α together with $\neg\alpha \vee \neg\beta$ entails $\neg\beta$ is exactly the same reasoning as expressing that β together with $\neg\alpha \vee \neg\beta$ entail $\neg\alpha$, because in both cases, what is meant is that α and β exclude each other). As a notational convenience, in examples of argument trees the $\diamond$ symbol is used to denote the claim of an argument when that argument is a canonical undercut (no ambiguity arises as proven in (Besnard & Hunter 2001)).

Example 4. *Let $\Delta = \{\alpha \vee \beta, \alpha \rightarrow \gamma, \neg\gamma, \neg\beta, \delta \leftrightarrow \beta\}$. For this, two argument trees for the consequent $\alpha \vee \neg\delta$ are given.*

$$\begin{array}{ccc} \langle \{\alpha \vee \beta, \neg\beta\}, \alpha \vee \neg\delta \rangle & & \langle \{\delta \leftrightarrow \beta, \neg\beta\}, \alpha \vee \neg\delta \rangle \\ \uparrow & & \uparrow \\ \langle \{\alpha \rightarrow \gamma, \neg\gamma\}, \diamond \rangle & & \langle \{\alpha \vee \beta, \alpha \rightarrow \gamma, \neg\gamma\}, \diamond \rangle \end{array}$$

A complete argument tree is an efficient representation of the counterarguments, counter-counterarguments, ... Furthermore, if Δ is finite, there is a finite number of argument trees with the root being an argument with consequent α that can be formed from Δ, and each of these trees has finite branching and a finite depth (the finite tree property). Note, also the definitions presented in this section can be used directly with first-order classical logic, so Δ and α are from the first-order classical language. Interestingly, the finite tree property also holds for the first-order case (Besnard & Hunter 2005).

Motivation for approximation

We now turn to the potential for approximation when building argument trees. When building arguments, and hence argument trees, it is tempting to think that automated reasoning technology can do more for us than it is guaranteed to. For each argument, we need a minimal set of formulae that proves the claim. An automated theorem prover (an ATP) may use a "goal-directed" approach, bringing in extra premises when required, but they are not guaranteed

to be minimal. For example, supposing we have a knowledgebase $\{\alpha, \alpha \wedge \beta\}$, for proving $\alpha \wedge \beta$, the ATP may start with the premise α, then to prove β, a second premise is required, which would be $\alpha \wedge \beta$, and so the net result is $\{\alpha, \alpha \wedge \beta\} \vdash \alpha \wedge \beta$, which does not involve a minimal set of premises. In addition, an ATP is not guaranteed to use a consistent set of premises since by classical logic it is valid to prove anything from an inconsistency.

So if we seek arguments for a particular claim δ, we need to post queries to an ATP to ensure that a set of premises entails δ, that the set of premises is minimal for this, and that it is consistent. So finding arguments for a claim α involves considering subsets Φ of Δ and testing them with the ATP to ascertain whether $\Phi \vdash \alpha$ and $\Phi \nvdash \bot$ hold. For $\Phi \subseteq \Delta$, and a formula α, let $\Phi?\alpha$ denote a call (a query) to an ATP. If Φ classically entails α, then we get the answer $\Phi \vdash \alpha$, otherwise we get the answer $\Phi \nvdash \alpha$, In this way, we do not give the whole of Δ to the ATP. Rather we call it with particular subsets of Δ. So for example, if we want to know if $\langle \Phi, \alpha \rangle$ is an argument, then we have a series of calls $\Phi?\alpha$, $\Phi?\bot$, $\Phi \setminus \{\phi_1\}?\alpha$,...,$\Phi \setminus \{\phi_k\}?\alpha$, where $\Phi = \{\phi_1, .., \phi_k\}$. So the first call is to ensure that $\Phi \vdash \alpha$ holds, the second call is to ensure that $\Phi \nvdash \bot$ holds, the remaining calls are to ensure that there is no subset Φ' of Φ such that $\Phi' \vdash \alpha$ holds. Now if $\Phi \vdash \alpha$ holds, but some of the further calls fail (i.e. Φ is not minimal or it is inconsistent) we still have an "approximate argument". So rather than throwing this away, we can treat it as an intermediate finding, and use it as part of an "approximate argument tree" which we can build with fewer calls to the ATP than building a complete argument tree, and this approximate argument tree can then be refined, as required, with the aim of getting closer to a complete argument tree. We formalize this next.

Approximate argumentation

An approximate argument is a pair $[\Phi, \alpha]$ where $\Phi \subseteq \mathcal{L}$ and $\alpha \in \mathcal{L}$. This is a very general definition. It does not assume that Φ is consistent, or that it entails α, or that it is even a subset of the knowledgebase Δ.

To focus our presentation in this paper, we will restrict consideration to a particular class of approximate arguments, namely entailments. If $\Phi \subseteq \Delta$ and $\Phi \vdash \alpha$, then $[\Phi, \alpha]$ is an **entailment**. Furthermore, we will consider some subclasses of entailments defined as follows: If $[\Phi, \alpha]$ is an entailment, and there is no $\Phi' \subset \Phi$ such that $\Phi' \vdash \alpha$, then $[\Phi, \alpha]$ is a **miniment**; If $[\Phi, \alpha]$ is an entailment, and $\Phi \nvdash \bot$, then $[\Phi, \alpha]$ is an **altoment**; And if $[\Phi, \alpha]$ is a miniment, and $[\Phi, \alpha]$ is an altoment, then $[\Phi, \alpha]$ is a **preargument**. Each of these kinds of entailment is defined as a relaxation of the definition for an argument: The support of an entailment implies the consequent, but neither an entailment nor an altoment has a support that is necessarily a minimal set of assumptions for implying the consequent and neither an entailment nor a miniment is necessarily a consistent set of assumptions for implying the consequent.

Example 5. *Let $\Delta = \{\alpha, \neg\alpha \vee \beta, \gamma, \neg\beta, \neg\gamma\}$. So entailments for β include $\{A_1, A_2, A_3, A_4, A_5\}$ of which $\{A_1, A_3, A_5\}$ are altoments, $\{A_2, A_5\}$ are miniments, and A_5 is a preargument.*

$$A_1 = [\{\alpha, \neg\alpha \vee \beta, \gamma, \beta\}, \beta]$$
$$A_2 = [\{\gamma, \neg\gamma\}, \beta]$$
$$A_3 = [\{\alpha, \neg\alpha \vee \beta, \gamma\}, \beta]$$
$$A_4 = [\{\alpha, \neg\alpha \vee \beta, \gamma, \neg\gamma\}, \beta]$$
$$A_5 = [\{\alpha, \neg\alpha \vee \beta\}, \beta]$$

An altoment is a "potentially overweight proof" in the sense that there are more assumptions in the support than required for the consequent, and a miniment is a "minimal proof" in the sense that if any formula is removed from the support there will be insufficient assumptions in the support for the consequent to be entailed.

Some simple observations that we can make concerning entailments include: (1) If $[\Gamma, \alpha]$ is an altoment, then there is a $\Phi \subseteq \Gamma$ such that $\langle \Phi, \alpha \rangle$ is an argument; (2) If $[\Gamma, \alpha]$ is an entailment, then there is a $\Phi \subseteq \Gamma$ such that $[\Phi, \alpha]$ is a miniment; and (3) If $[\Phi, \alpha]$ is a preargument, then $\langle \Phi, \alpha \rangle$ is an argument.

An **approximate undercut** for an approximate argument $[\Phi, \alpha]$ is an approximate argument $[\Psi, \beta]$ such that $\beta \vdash \neg(\wedge\Phi)$ (where if $\Phi = \{\phi_1, .., \phi_n\}$, and $\langle \phi_1, \ldots, \phi_n \rangle$ is the canonical enumeration of Φ, then $\wedge\Phi$ is $\phi_1 \wedge .. \wedge \phi_n$).

An **approximate tree** is a tree T where each node is an approximate argument from Δ. There are various kinds of approximate tree. Here we define three particular kinds: (1) An **entailment tree** for α is an approximate tree where each node is an entailment and the root is for α; (2) An **altoment tree** for α is an approximate tree where each node is an altoment and the root is for α; and (3) A **preargument tree** for α is an approximate tree where each node is a preargument and the root is for α. For these trees, we do not impose that the children of a node are approximate undercuts, but in the way we construct them, we will aim for this.

Example 6. *In the following, T_1 is an entailment tree, T_2 is an altoment tree, and T_3 is a preargument tree.*

$$T_1 \quad [\{\beta, \neg\beta, \delta, \delta \vee \beta, (\beta \vee \delta) \rightarrow \alpha\}, \alpha]$$
$$[\{\phi, \neg\delta, \delta \vee \neg\beta\}, \neg\beta] \qquad [\{\phi, \neg\phi\}, \neg\delta]$$

$$T_2 \quad [\{\beta, \delta, \delta \vee \beta, (\beta \vee \delta) \rightarrow \alpha\}, \alpha]$$
$$[\{\phi, \neg\delta, \delta \vee \neg\beta\}, \neg\beta] \qquad [\{\phi, \neg\beta, \neg\delta \vee \beta\}, \neg\delta]$$

$$T_3 \quad [\{\beta, \beta \rightarrow \alpha\}, \alpha]$$
$$[\{\gamma, \gamma \rightarrow \neg\beta\}, \neg(\beta \wedge (\beta \rightarrow \alpha))]$$

Obviously, all preargument trees are altoment trees, and all altoment trees are entailment trees.

Definition 7. *A* **complete preargument tree** *is a preargument tree where (1) for no node $[\Phi, \beta]$ with ancestor nodes $[\Phi_1, \beta_1], \ldots, [\Phi_n, \beta_n]$ is Φ a subset of $\Phi_1 \cup \cdots \cup \Phi_n$ and (2) for each node with a preargument $[\Phi, \beta]$, the children nodes consist of all the prearguments of the form $[\Psi, \neg(\wedge\Phi)]$.*

The first condition above ensures that the support of each preargument on a branch has to include at least one premise that has not been used by its ancestors. The second condition above ensures that the children nodes of a node consist

of all preargumentsthat are approximate undercuts negating the conjunction of the support of the parent. This mirrors the definition for an argument tree (Definition 6) and the following result shows they are isomorphic. As an illustration, T_3 in Example 6 is a complete preargument tree.

Proposition 1. *If T is a complete preargument tree, then there is an argument tree T' and there is a bijection f from the nodes in T to the nodes in T' such that for all the preargumentsts $[\Phi, \alpha]$ in T, $f([\Phi, \alpha]) = \langle \Phi, \alpha \rangle$.*

From the above result, we see that not only is each preargument effectively equivalent to an argument (i.e. each preargument $[\Phi, \alpha]$ is corresponds to an argument $\langle \Phi, \alpha \rangle$), but that also the constraints on the definition are sufficient for each approximate undercut to behave equivalently to a canonical undercut.

The following definition of refinement is a relationship that holds between some altoment trees. It holds when each altoment in T_2 uses the same or fewer assumptions than its corresponding altoment in T_1 and its claim is weaker or the same as its corresponding altoment in T_1. For this, let $\mathsf{Support}([\Phi, \alpha])$ be Φ, and let $\mathsf{Claim}([\Phi, \alpha])$ be α, where $[\Phi, \alpha]$ is an entailment.

Definition 8. *Let T_1 and T_2 be altoment trees. T_2 is a* **refinement** *of T_1 iff there is a bijection f from the nodes of T_1 to the nodes of T_2 such that for all nodes A in T_1, $\mathsf{Support}(A) \subseteq \mathsf{Support}(f(A))$ and $\mathsf{Claim}(f(A)) \vdash \mathsf{Claim}(A)$. We call f the* **refinement function***.*

Proposition 2. *If T_2 is a refinement of T_1 with refinement function f, then for all $[\Phi_1, \alpha_1] \in T_1$, if $f([\Phi_1, \alpha_1]) = [\Phi_2, \alpha_2]$, then $\langle \Phi_2, \alpha_2 \rangle$ is more conservative than $\langle \Phi_1, \alpha_1 \rangle$.*

Refinement is useful because we can build a tree using altoments, and then refine those altoments as part of a process of obtaining a better approximate tree, and if required, a complete preargument tree. We consider this process more generally for entailments in the next section.

Constructing approximate trees

To render the construction, and improvement, of approximate trees implementable, we define the **revision steps** that can be undertaken on a tree T as follows where T' is the result of the revision step, and all entailments come from the knowledgebase Δ.

1. T' is obtained by **resupport** from T by taking an entailment $[\Phi, \alpha]$ in T and removing one formula, say ϕ, such that $[\Phi \setminus \{\phi\}, \alpha]$ is an entailment.
2. T' is obtained by **reconsequent** from T by replacing an entailment $[\Phi, \alpha]$ in T with entailment $[\Phi, \alpha']$ where $[\Psi, \beta]$ is the parent of $[\Phi, \alpha]$ in T and $[\Psi, \beta]$ is the parent of $[\Phi, \alpha']$ in T' and $\alpha \not\equiv \alpha'$ and $\alpha' \equiv \neg(\wedge\Psi)$.
3. T' is obtained by **expansion** from T by taking an entailment $[\Phi, \alpha]$ in T and adding an entailment $[\Psi, \beta]$ such that $[\Psi, \beta]$ is an approximate undercut of $[\Phi, \alpha]$ and it has not been shown that $\Psi \vdash \bot$.
4. T' is obtained by **contraction** from T by removing an entailment $[\Psi, \beta]$ (and all its offspring) such that $[\Psi, \beta]$ is a miniment and $\Psi \vdash \bot$.
5. T' is obtained by **deflation** from T by removing an entailment $[\Psi, \beta]$ (and all its offspring) such that $[\Psi, \beta]$ is a child of $[\Phi, \alpha]$ and $\Psi \cup \Phi \nvdash \bot$.

We explain the revision steps as follows: Resupport weakens the support of an entailment to remove an unnecessary premise; Reconsequent strengthens the claim of an entailment so that it negates the conjunction of its parents support; Expansion adds a new approximate undercut to the tree (assuming that it has not been shown to have an inconsistent support); Contraction removes a node which has become an inconsistent miniment (after previously being subject to resupport); and Deflation removes a node with a support that is consistent with the support of its parent (after previously one or other being subject to resupport). Illustrations of using revision steps are given in Example 7 and Example 8.

Example 7. *Each of the following three trees is an altoment tree. Furthermore, T_2 is a resupport of T_1 and T_3 is an expansion of T_2.*

$$T_1 \quad \begin{array}{c}[\{\alpha, \alpha \to \beta, \neg\gamma, \delta\}, \beta] \\ \uparrow \\ [\{\gamma, \gamma \to \neg\alpha, \sigma\}, \neg\alpha]\end{array} \qquad T_2 \quad \begin{array}{c}[\{\alpha, \alpha \to \beta, \delta\}, \beta] \\ \uparrow \\ [\{\gamma, \gamma \to \neg\alpha, \sigma\}, \neg\alpha]\end{array}$$

$$T_3 \quad \begin{array}{c}[\{\alpha, \alpha \to \beta, \delta\}, \beta] \\ \uparrow \\ [\{\gamma, \gamma \to \neg\alpha, \sigma\}, \neg\alpha] \\ \uparrow \\ [\{\neg\gamma\}, \neg\gamma]\end{array}$$

The next result shows that we can obtain a complete preargument tree by some finite sequence of revision steps.

Theorem 1. *If T_n is a complete preargument tree for α, then there a sequence $\langle T_1, ..., T_n \rangle$ of approximate trees for α, s.t. T_1 is an altoment tree with just a root node, and for each i, where $i < n$, T_{i+1} is obtained by a revision step from T_i.*

Starting with an altoment tree for α that contains one node, and then using a sequence of revision steps to obtain a complete preargument tree, does not necessarily offer any computational advantages over constructing a complete argument tree directly (by finding an argument for the root, and then finding canonical undercuts to the root, and then by recursion, finding canonical undercuts to the canonical undercuts). To revise an approximate tree involves calls to the ATP, and so the more we revise an approximate tree, the less there is an efficiency advantage over constructing a complete argument tree. The real benefit of approximate argumentation is that an intermediate tree (in the sequence above) is more informative (than a partially constructed argument tree) since it tends to have more nodes, and thereby better indicate the range of potential conflicts arising in Δ. So, in comparison with an argument tree, an approximate tree is less cautious (it compromises on the correctness of the arguments used), is less expensive (it uses fewer calls to an ATP and each call made would also be made to construct each argument in the corresponding argument tree), and is more informative (in reporting on potential conflicts in Δ).

To obtain an entailment tree, we use the following algorithm which has an upper limit on the number of revision steps used.

Definition 9. *The algorithm* $\texttt{Generate}(\Delta, \Phi, \alpha, \lambda)$ *returns an entailment tree that is constructed in a fixed number of cycles (delineated by the number* λ*), for a knowledgebase* Δ*, and an altoment* $[\Phi, \alpha]$ *for the root of the entailment tree that is returned.*

$\texttt{Generate}(\Delta, \Phi, \alpha, \lambda)$
  Let T be the node $[\Phi, \alpha]$
  Let counter $= 0$
  While counter $\leq \lambda$ and there is
    a revision step T' of T
    Let $T = T'$
    Let counter $=$ counter $+ 1$
  Return T

The aim of using the above algorithm is to start with an entailment for α as the root, and then incrementally revise this entailment tree to get as close as possible to a complete preargument tree within an acceptable number of iterations.

Example 8. *Let* $\Delta = \{\alpha, \alpha \rightarrow \beta, \delta, \gamma, \neg\alpha \vee \neg\beta, \beta\}$ *For trees* $T_1, .., T_7$ *below,* T_{i+1} *is obtained from* T_i *by the revision step given in brackets. Let* $\diamond$ *denote the claim that is the negation of the conjunction of the support of its parent.*

$$[\{\alpha, \alpha \rightarrow \beta, \delta\}, \alpha \wedge \beta]$$

(3)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta, \delta\}, \alpha \wedge \beta] \\ \uparrow \\ [\{\alpha, \neg\alpha \vee \neg\beta\}, \neg\beta]\end{array}$$

(3)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta, \delta\}, \alpha \wedge \beta] \\ \nearrow \quad \nwarrow \\ [\{\alpha, \neg\alpha \vee \neg\beta\}, \neg\beta] \quad [\{\neg\alpha \vee \neg\delta, \delta\}, \neg\alpha]\end{array}$$

(1)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta\}, \alpha \wedge \beta] \\ \nearrow \quad \nwarrow \\ [\{\alpha, \neg\alpha \vee \neg\beta\}, \neg\beta] \quad [\{\neg\alpha \vee \neg\delta, \delta\}, \neg\alpha]\end{array}$$

(2)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta\}, \alpha \wedge \beta] \\ \nearrow \quad \nwarrow \\ [\{\alpha, \neg\alpha \vee \neg\beta\}, \diamond] \quad [\{\neg\alpha \vee \neg\delta, \delta\}, \neg\alpha]\end{array}$$

(1)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta\}, \alpha \wedge \beta] \\ \nearrow \quad \nwarrow \\ [\{\neg\alpha \vee \neg\beta\}, \diamond] \quad [\{\neg\alpha \vee \neg\delta, \delta\}, \neg\alpha]\end{array}$$

(2)
$$\begin{array}{c}[\{\alpha, \alpha \rightarrow \beta\}, \alpha \wedge \beta] \\ \nearrow \quad \nwarrow \\ [\{\neg\alpha \vee \neg\beta\}, \diamond] \quad [\{\neg\alpha \vee \neg\delta, \delta\}, \diamond]\end{array}$$

The following shows that if we run the $\texttt{Generate}(\Delta, \Phi, \alpha, \lambda)$ algorithm for sufficient time, it will eventually result in a complete preargument tree for α.

Theorem 2. *For all propositional knowledgebases* Δ *and altoment* $[\Phi, \alpha]$*, there is a* $\lambda \in \mathbb{N}$ *such that* $\texttt{Generate}(\Delta, \Phi, \alpha, \lambda)$ *returns an altoment tree* T *for* α*, and* T *is a complete preargument tree for* α*.*

Corollary 1. *Let* $\langle T_1, ..., T_n \rangle$ *be a sequence of entailment trees for* α *such that* T_1 *is an altoment tree with just a root node, and for each* i*, where* $1 \leq i < n$*,* T_{i+1} *is obtained by a revision step from* T_i*. If* T_n *is such that no further revision steps are possible on it, then* T_n *is a complete preargument tree for* α*.*

We can improve the algorithm by incorporating selection criteria to prefer some revision steps over others. For example, we may prefer: (1) an expansion that adds a node higher up the tree rather than lower down the tree; (2) re-support steps on nodes with large supports rather than those with small supports; (3) as many expansion steps as possible in order to get an impression of as many as possible of the conflicts that potentially exist. By adopting selection criteria, we aim to have more meaningful (from a user's perspective) approximate trees returned when the threshold for the number of revision steps undertaken by the algorithm.

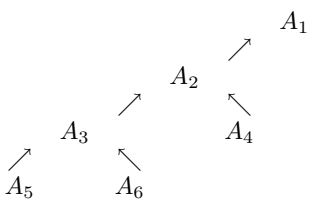
Bias in approximate trees

In this section, we consider another way to bias the construction of approximate trees by amending the revision steps available. To explain and motivate this, we first require some further definitions. For an approximate tree T, each node in T is either an **attacking node** or a **defending node**. If a node A_r is the root, then A_r is a defending node. If a node A_i is a defending node, then any child A_j of A_i is an attacking node. If a node A_j is an attacking node, then any child A_k of A_j is a defending node. The intuition of this nomenclature is apparent in the following definition for judging.

Definition 10. *The* **win judge** *is a function, denoted* Win*, from the set of approximate trees to* $\{yes, no\}$ *such that* $\mathsf{Win}(T) = yes$ *iff* $mark(A_r) = U$ *where* A_r *is the root node of* T*. For a node* A_i*,* $mark(A_i) = U$ *when it is undefeated, and* $mark(A_i) = D$ *it is defeated. Deciding whether a node is defeated or undefeated depends on whether or not all its children are defeated: For all non-leaf nodes* A_i*,* $mark(A_i) = D$ *iff there is a child* A_j *of* A_i *s.t.* $mark(A_j) = U$*. For all leaves* A_l*,* $mark(A_l) = U$*.*

So $\mathsf{Win}(T)$ is yes when the approximate argument at the root is defended from all attacks. This is a well-studied criterion for argumentation (e.g. (García & Simari 2004)) though there are some interesting alternatives. However we stress at this point that we believe the primary purpose of logical argumentation is to highlight the key arguments and counterarguments in the knowledgebase Δ rather than applying a judge function. Nonetheless, the win judge is a useful way of assessing the conflicts in a knowledgebase.

Example 9. *Consider the following preargument tree* T *where* $A_1 = [\{\delta, \delta \rightarrow \sigma\}, \sigma]$*,* $A_2 = [\{\neg\gamma, \neg\gamma \rightarrow \neg\delta\}, \diamond]$*,* $A_3 = [\{\alpha, \beta, \alpha \wedge \beta \rightarrow \gamma\}, \diamond]$*,* $A_4 = [\{\gamma \vee \delta\}, \diamond]$*,* $A_5 = [\{\neg\alpha\}, \diamond]$*, and* $A_6 = [\{\neg\beta\}, \diamond]$*.*



Hence, $\mathsf{Win}(T) = yes$ *holds. The intuition in this example is that* A_4 *is sufficient to defeat* A_2 *irrespective of* A_3*. And*

so the existence of A_5 and/or A_6 does not affect the ability of A_4 to defeat A_2 and hence allow A_1 to be undefeated.

We now consider how we can incorporate bias into the construction of altoment trees. For this, we adapt the revision step of resupport as follows: T' is obtained by **attack-resupport** from T by taking an entailment $[\Phi, \alpha]$ for an attacking node in T and removing one formula, say ϕ, such that $[\Phi \setminus \{\phi\}, \alpha]$ is an entailment. Using this alternative to resupport, we can define the following type of biased tree.

Definition 11. *Let $\langle T_1, ..., T_n \rangle$ be a sequence of altoment trees for α such that T_1 is a node with an altoment $[\Phi, \alpha]$, and for each i, where $1 \leq i < n$, T_{i+1} is obtained by an attack-resupport, reconsequent, contraction, or deflation step from T_i, and when no such step is possible for T_i, then T_{i+1} is obtained by an expansion step of T_i. If T_n is such that no further step is possible, then T_n is an* **attack-bias tree** *for α started from $[\Phi, \alpha]$.*

For purposes of comparison, suppose we have a λ s.t. $\texttt{Generate}(\Delta, \Phi, \alpha, \lambda)$ returns a complete preargument tree T for α. In this case, we say that T is a complete preargument tree for α started from $[\Phi, \alpha]$.

Proposition 3. *If T is an attack-bias tree, then each attacking node is a preargument, whereas each defending node is an altoment but not necessarily a preargument.*

The above result means that in an attack-bias tree it is easier to undercut a defending node than an attacking node. This leads us to the following result.

Proposition 4. *Let T_a be a attack-bias tree started from $[\Phi, \alpha]$, and let T_p be a complete preargument tree started from $[\Phi, \alpha]$. If $\mathsf{Win}(T_a) = yes$, then $\mathsf{Win}(T_p) = yes$.*

Constructing an attack-bias tree may involve fewer revision steps than constructing a complete preargument tree, and from the above, it provides a bound on the outcome of the corresponding complete preargument tree. We can define further biased constructions such as for a defence-bias tree by restricting resupport steps to the defending nodes.

Discussion

Much progress has been made on developing formalisms for argumentation. Some algorithms for argumentation have been developed (for example (Kakas & Toni 1999; Cayrol, Doutre, & Mengin 2001; Baroni & Giacomin 2002; García & Simari 2004)). However, relatively little progress has been made in developing techniques for overcoming the computational challenges of constructing arguments. Even though there is a proposal for storing information about previous calls to an ATP in a form of lemma generation called contouring (Hunter 2006), and there are proposals for making defeasible logic programming more efficient by storing non-instantiated arguments in a database, again for reducing the number of called to an ATP (Capobianco, Chesnevar, & Simari 2005), and by the use of prunning strategies (Chesnevar, Simari, & Godo 2005), there does appear to be a pressing need to look more widely for approaches for improving the efficiency of argumentation.

In this paper, we have introduced a framework for approximate arguments that can be used as useful intermediate results when undertaking argumentation using an ATP. Finding approximate arguments requires fewer calls to an ATP, but it involves compromising the correctness of arguments. Our next step is to undertake empirical studies with the algorithm to investigate this efficiency-correctness trade-off.

Approximate arguments may also be a useful vehicle for studying approximate arguments as arising in human cognition, and may be important for characterising some kinds of discussions and negotiations in multi-agent dialogue systems where agents may put forward approximate arguments that are not altoments, or even not entailments (perhaps for competitive reasons or through ignorance).

Whilst our presentation is based on a particular approach to logic-based argumentation, we believe the proposal could be adapted for a range of other logic-based approaches to argumentation (for example (García & Simari 2004; Amgoud & Cayrol 2002)) by adapting our definitions for approximate arguments and for revision steps.

Acknowledgements

The author is very grateful to Philippe Besnard for feedback on an earlier draft of this paper.

References

Amgoud, L., and Cayrol, C. 2002. A model of reasoning based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence* 34:197–216.

Baroni, P., and Giacomin, M. 2002. Argumentation through a distributed self-stabilizing approach. *J. Experimental & Theoretical AI* 14(4):273–301.

Besnard, P., and Hunter, A. 2001. A logic-based theory of deductive arguments. *Artificial Intelligence* 128:203–235.

Besnard, P., and Hunter, A. 2005. Practical first-order argumentation. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI'2005)*, 590–595. MIT Press.

Capobianco, M.; Chesnevar, C.; and Simari, G. 2005. Argumentation and the dynamics of warranted beliefs in changing environments. *International Journal on Autonomous Agents and Mutliagent Systems* 11:127–151.

Cayrol, C.; Doutre, S.; and Mengin, J. 2001. Dialectical proof theories for the credulous preferred semantics of argumentation frameworks. In *Proc. ECSQARU'01*, volume 2143 of *LNCS*, 668–679. Springer.

Chesnevar, C.; Maguitman, A.; and Loui, R. 2001. Logical models of argument. *ACM Comp. Surveys* 32:337–383.

Chesnevar, C.; Simari, G.; and Godo, L. 2005. Computing dialectical trees efficiently in possibilistic defeasible logic programming. In *Proceedings of the 8th Interntational Logic Programming and Non-monotonic Reasoning Conference*, Lecture Notes in Computer Science. Springer.

Dimopoulos, Y.; Nebel, B.; and Toni, F. 2002. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence* 141:57–78.

Dung, P.; Kowalski, R.; and Toni, F. 2006. Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence* 170(2):114–159.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM* 42:3–42.

García, A., and Simari, G. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Hunter, A. 2006. Contouring of knowledge for intelligent searching for arguments. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'06)*. (in press).

Kakas, A., and Toni, F. 1999. Computing argumentation in logic programming. *J. Logic and Computation* 9:515–562.

Koriche, F. 2002. Approximate coherence-based reasoning. *J. of Applied Non-classical Logics* 12(2):239–258.

Levesque, H. 1984. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'84)*, 198–202.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7:25–75.

Prakken, H., and Vreeswijk, G. 2000. Logical systems for defeasible argumentation. In Gabbay, D., ed., *Handbook of Philosophical Logic*. Kluwer.

Schaerf, M., and Cadoli, M. 1995. Tractable reasoning via approximation. *Artificial Intelligence* 74:249–310.

5.3 On Complexity of DeLP through Game Semantics

On the Complexity of DeLP through Game Semantics*

Laura A. Cecchi
Depto. Ciencias de la Computación - Fa.E.A.
Universidad Nacional del Comahue
Buenos Aires 1400
(8300) Neuquén - ARGENTINA
lcecchi@uncoma.edu.ar

Pablo R. Fillottrani and **Guillermo R. Simari**
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av. Alem 1253
(8000) Bahía Blanca - ARGENTINA
{prf,grs}@cs.uns.edu.ar

Abstract

Defeasible Logic Programming (DeLP) is a general argumentation based system for knowledge representation and reasoning. Its proof theory is based on a dialectical analysis where arguments for and against a literal interact in order to determine whether this literal is believed by a reasoning agent. The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that is sound and complete for DeLP proof theory.

Complexity theory is an important tool for comparing different formalism and for helping to improve implementations whenever it is possible. In this work we address the problem of studying the complexity of some important decision problems in DeLP. Thus, we characterize the relevant decision problems in the context of DeLP and $\mathcal{GS}$, and we define data and combined complexity for DeLP. Since DeLP computes every argument from a set of defeasible rules, it is of central importance to analyze the complexity of two decision problems. The first one can be defined as "Is a set of defeasible rules an argument for a literal under a defeasible logic program?". We prove that this problem is **P**-complete. The second decision problem is "Does there exist an argument for a literal under a defeasible logic program?". We prove that this problem is in **NP**. Furthermore, we study data complexity of query answering in the context of DeLP. As far as we know, data complexity has not been introduced in the context of argumentation systems.

KEYWORDS: Argumentation Systems, Defeasible Reasoning, Logic Programming, Game-based Semantics, Complexity

Introduction

Defeasible Logic Programming (DeLP) is a general argumentation based tool for knowledge representation and reasoning (García & Simari 2004)[1]. Its proof theory is based on a dialectical analysis where arguments for and against a literal interact in order to determine whether this literal is believed by a reasoning agent. The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that links game-semantics (Abramsky & McCusker 1997) and model-theory. Soundness and completeness of $\mathcal{GS}$ with respect to DeLP proof theory have been proved (Cecchi & Simari 2004).

Complexity theory is an important tool for comparing different formalism, and for helping to improve implementations whenever it is possible. For this reason, it is important to analyze the computational complexity and the expressive power of DeLP. The former tells us how difficult it is to answer a query, while the latter gives a precise characterization of the concepts that are definable as queries.

Even thought complexity for nonmonotonic reasoning systems has been studied in depth for several formalisms such us default logic, autoepistemic logic, circumscription, abduction and logic programming (Cadoli & Schaerf 1993; Dantsin *et al.* 2001) until recently not many complexity results for argumentation systems have been reported.

This situation can be explained in part by the fact that, historically, implementations of argumentation systems have been limited to areas with no real time response restriction (see (Verheij 1998; Gordon & Karacapilidis 1997)). Recently, however, several applications have been developed, and implemented using argumentation systems related, for instance, with multiagent systems and web search (Atkinson, Bench-Capon, & Mc Burney 2004; Chesñevar & Maguitman 2004a; 2004b; Bassiliades, Antoniou, & Vlahavas 2004). Scalability and robustness of such approaches heavily depend on the computational properties of the underlying algorithms. It is hence crucial to study these properties in order to expand the application fields of argumentation systems.

Different computational complexity results (Dimopoulos, Nebel, & Toni 2002; Bench-Capon 2003; Amgoud & Cayrol 2002; Dunne & Bench-Capon 2002) have been presented on argumentation abstract framework (Bondarenko *et al.* 1997; Dung 1995), based on admissibility and preferability semantics. However, those results do not apply directly to DeLP, because its semantics are quite different. Another notable study of the computational complexity of defeasible systems

*This research was partially supported by the Secretaría General de Ciencia y Tecnología of the Universidad Nacional del Sur, by the Universidad Nacional del Comahue (Proyecto de Investigación 04/E062), by the Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13096, PICT 2003 15043, PAV 076) and by the National Research Council (CONICET), ARGENTINA.

[1]The interested reader can find an on-line interpreter for DeLP in `http://lidia.cs.uns.edu.ar/DeLP`

has been done in (Maher 2001). But, defeasible theory analyzed in this work greatly differs from DeLP in several points, such as knowledge representation (facts and strict rule, defeasible and defeaters rules) and their proof theories.

When measuring the complexity of evaluating queries in a specific language, we distinguish between several kinds of complexity according to (Vardi 1982; Papadimitriou & Yannakakis 1997; Dantsin *et al.* 2001). *Data complexity* is the complexity of evaluating a specific query in the language, when the query is fixed, and we study the complexity of applying this query to arbitrary databases; the complexity is thus given as a function of the size of the database. *Program or Expression complexity* appears when a specific database is fixed, and we study the complexity of applying queries represented by arbitrary expressions in the language; the complexity is given as a function of the length of the expression. *Combined complexity* considers both query and database instance as input variables.

In this work we are concerned with the study of complexity of some important decision problems of DeLP. The system and its asociated game semantics $\mathcal{GS}$ are analyzed introducing relevant decision problems in relation to the possible query answers.

Since DeLP builds the arguments from a defeasible logic program results of central importance to consider and evaluate two questions: "is a set of defeasible rules an argument for a literal under a defeasible logic program?" which has been proved to be **P**-complete, and does there exit an argument for a literal under a defeasible logic program? which has been proved to be in **NP**.

We define data, expression and combined complexity in the context of DeLP, in order to evaluate the efficiency of DeLP implementations. In particular, we study data complexity of query answering to assess DeLP applications over database technologies. As far as we know data complexity has not been introduced in the context of argumentation systems.

The paper is structured as follows. In the following section we briefly outline the fundamentals of DeLP, and describe the declarative game-based semantics $\mathcal{GS}$. Then, we discuss DeLP through $\mathcal{GS}$ semantics pointing out the decision problems that are of central importance, and we define data, expression and combined complexity in the context of DeLP. Afterwards, we give complexity results on the existence of an argument for a literal L under a defeasible logic program $\mathcal{P}$, and on the decision problem of whether a subset of defeasible rules is an argument for a literal L under $\mathcal{P}$. Next, we analyze data complexity for DeLP, and we present complexity results for two decision problems on entailment. In the last section, we summarize the main contributions of this work, and we present our conclusions and future research lines.

DeLP and Game Semantics $\mathcal{GS}$

We will start by introducing some of the basic concepts in DeLP (see (García & Simari 2004)). In the language of DeLP a literal L is a atom A or a negated atom $\sim A$, where $\sim$ represents the strong negation in the logic programming sense. The complement of a literal L, denoted as $\overline{L}$, is defined as follows: $\overline{L} = \sim A$, if L is an atom, otherwise if L is a negated atom, $\overline{L} = A$. Let X be a set of literals, $\overline{X}$ is the set of the complement of every member in X.

Definition 1 A *strict rule* is an ordered pair, denoted "$Head \leftarrow Body$", where "$Head$" is a ground literal, and "$Body$" is a finite set of ground literals. A strict rule with head L_0 and body $\{L_1, \ldots L_n, n > 0\}$ is written as $L_0 \leftarrow L_1, \ldots L_n$. If body is the empty set, then we write $L_0.$, and the rule is called a *Fact*. A *defeasible rule* is an ordered pair, denoted "$Head \prec Body$", where "$Head$" is a ground literal, and "$Body$" is a finite, non-empty set of ground literals. A defeasible rule with head L_0 and body $\{L_1, \ldots L_n, n > 0\}$ is written as $L_0 \prec L_1, \ldots L_n$.
A *defeasible logic program* $\mathcal{P}$, abbreviated *de.l.p.*, is a set of strict rules and defeasible rules. We will distinguish the subsets Π_F of facts, Π_R of strict rules, $\Pi = \Pi_F \cup \Pi_R$ and the subset Δ of defeasible rules.

We denote by Lit the set of all the ground literals that can be generated considering the underlying signature of a *de.l.p.* an we denote by Lit^+ the set of all the atoms in Lit.

Intuitively, whereas Π is a set of certain and exception-free knowledge, Δ is a set of defeasible knowledge, i.e., tentative information that could be used, whenever nothing is posed against it.

By definition a *de.l.p.* may be an infinite set of strict and defeasible rules but for complexity analysis we restrict ourselves to finite defeasible logic programs.

DeLP proof theory is based on developments in non monotonic argumentation systems (Pollock 1987; Simari & Loui 1992). An *argument for a literal* L is a minimal subset of Δ that together with Π consistently entails L. The notion of entailment corresponds to the usual SLD derivation used in logic programming, performed by backward chaining on both strict and defeasible rules, where negated atoms are treated as a new atom in the underlying signature. Thus, an agent can explain a literal L, throughout this argument.

In order to determine whether a literal L is supported from a *de.l.p.* a dialectical tree for L is built. An argument for L represents the root of the dialectical tree, and every other node in the tree is a defeater argument against its parent. At each level, for a given a node we must consider all the arguments against that node. Thus every node has a descendant for every defeater. A comparison criteria is needed for determining whether an argument defeats another. Even though there exist several preference relations considered in the literature, in this first approach we will abstract away from that issue.

We will say that a literal L is warranted if there is an argument for L, and in the dialectical tree each defeater of the root is itself defeated. Recursively, this leads to a marking procedure of the tree that begins by considering the fact that leaves of the dialectical tree are undefeated arguments as a consequence of having no defeaters. Finally, an agent will believe in a literal L, if L is a warranted literal.

There exist four possible answers for a query L: YES if L is warranted, NO if $\overline{L}$ is warranted (i.e., the complement of L is warranted), UNDECIDED if neither L nor $\overline{L}$ are warranted,

and UNKNOWN if L is not in the underlying signature of the program.

We have briefly given an intuitive introduction to the DeLP language and the dialectical procedure for obtaining a warranted conclusion. For complete details on DeLP see (García & Simari 2004).

Games have an analogy with a dispute and, therefore, that analogy extends to argument-based reasoning. A dispute can be seen as a game where in an alternating manner, the player P, the proponent, starts with an argument for a literal. The player O, the opponent, attacks the previous argument with a counterargument strong enough to defeat it. The dispute could continue with a counterargument of the proponent, and so on. When a player runs out of moves, i.e., that player can not find a counterargument for any of his adversary's arguments, the game is over. If the proponent's argument has not been defeated then she has won the game.

The semantics $\mathcal{GS}$ is a declarative trivalued game-based semantics for DeLP that links game-semantics (Abramsky & McCusker 1997) and model theory. Soundness and completeness of $\mathcal{GS}$ with respect to DeLP proof theory have been proved (Cecchi & Simari 2004). In the following we present some notions of $\mathcal{GS}$, for more details see (Cecchi & Simari 2000; 2004).

Let X be a set and $\{x_1, \ldots, x_n\} \subseteq X$, X^* is the set of finite sequences over X and $[x_1 \ldots x_n]$ denotes the sequence of the elements $x_1, \ldots, x_n$. We write $|s|$ for the length of a finite sequence and s_i for the ith element of s, $1 \leq i \leq |s|$. Concatenation of sequences is indicated by juxtaposition. If $t = su$ for some sequences $t,\ s,\ u$, then we say that s is a prefix of t. Let $Pref(S)$ be a set of prefix of S, then S is prefix closed if $S = Pref(S)$.

In order to use a game to capture the dialectical procedure, we need to define in a declarative way the movements of such game: the argument. The followings definitions are based on the notation introduced in (Lifschitz 1996).

Definition 2 Let X be a set of ground literals. The set X is *rigorously closed under a de.l.p.* $\mathcal{P}$, if for every strict rule $Head \leftarrow Body$ of $\mathcal{P}$, $Head \in X$ whenever $Body \subseteq X$, and for every defeasible rule $Head' \prec Body'$ of $\mathcal{P}$, $Head' \in X$ whenever $Body' \subseteq X$.
The set X is *consistent* if there is no literal L such that $\{L, \overline{L}\} \subseteq X$. Otherwise, we will say that X is *inconsistent*.
We say that X is *logically closed* if it is consistent or it is equal to Lit.

Intuitively, if the set of knowledge of an agent is rigorously closed under a *de.l.p.*, the agent will not believe in a literal that she cannot explain.

Definition 3 Let $\mathcal{P}$ be a *de.l.p.*. The *set of rigorous consequences* of $\mathcal{P}$, denoted $Cn_R(\mathcal{P})$, is the least set of literals w.r.t. inclusion, such that it is logically closed and rigorously closed under $\mathcal{P}$.

Even though rigorous consequences do not reflect the underlying ideas of strict and defeasible rules, they are very useful for introducing a declarative definition of argument.

Definition 4 Let $\mathcal{P} = \langle \Pi, \Delta \rangle$ be a *de.l.p.*. We say that $\langle \mathcal{A}, L \rangle$ is an argument structure for a ground literal L, if $\mathcal{A}$ is a set of defeasible rules of Δ, such that:

1. $L \in Cn_R(\Pi \cup \mathcal{A})$
2. $Cn_R(\Pi \cup \mathcal{A}) \neq Lit$
3. $\mathcal{A}$ is minimal w.r.t. inclusion, i.e., there is no $\mathcal{A}' \subseteq \mathcal{A}$ such that satisfies (1) and (2).

For convenience we will simply speak of argument instead of argument structure whenever this does not lead to misunderstandings. Let's introduce game concept and $\mathcal{GS}$ semantics.

Definition 5 Let $\mathcal{P} = (\Pi, \Delta)$ be a *de.l.p.*, L a literal and $\langle \mathcal{A}, L \rangle$ an argument structure for L. A *game* for $\langle \mathcal{A}, L \rangle$ with respect to $\mathcal{P}$, that we denote $G(\langle \mathcal{A}, L \rangle, \mathcal{P})$, is a structure

$$(M_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}, J_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}, P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})})$$

where

- $M_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$ is a set of argument structure.
- $J_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})} : M_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})} \times I \rightarrow \{\mathsf{P}, \mathsf{O}\}$ where I is an enumerable index;
- $P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})} \subseteq M^*_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$, where $P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$ is a non-empty, prefix-closed set.
  Each sequences s of $P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$ satisfy:
  1. $s = [\langle \mathcal{A}, L \rangle] s'$, s' possibly empty.
  2. For all $i,\ 1 < i \leq |s|$

     $$\begin{aligned} J_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}(s_1, 1) &= \mathsf{P} \\ J_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}(s_i, i) &= \overline{J_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}(s_{i-1}, i-1)} \end{aligned}$$

     $\overline{\mathsf{P}} = \mathsf{O}$ and $\overline{\mathsf{O}} = \mathsf{P}$.
  3. If $s \in P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$, then for each argument structure $\langle \mathcal{A}_2, L_2 \rangle$ that is a legal move for $s_{|s|}$, there exists a sequence $t \in P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$, such that $t = s[\langle \mathcal{A}_2, L_2 \rangle]$.
  4. No other sequence belongs to $P_{G(\langle \mathcal{A}, L \rangle, \mathcal{P})}$.

Movements in a game are the introduction of arguments. A legal move in the game over a sequence s is an argument $\mathcal{A}$ such that strictly defeats $s_{|s|}$ or defeats non strictly $s_{|s|}$ and $s_{|s|}$ strictly defeats $s_{|s|-1}$. Furthermore, such legal move $\mathcal{A}$ cannot be part of another argument in s, ie we cannot introduce more than once an argument neither for nor against the first move. Finally, this move must be consistent with every move made by the same player in the sequence s.

For every argument $\mathcal{A}$ for a literal L we can built a game whose first move is $\langle \mathcal{A}, L \rangle$. Thus, a family of games will be obtained considering all the arguments for L.

Definition 6 Let $\mathcal{P}$ be a *de.l.p.*, L a literal under the signature of $\mathcal{P}$, $\langle \mathcal{A}_1, L \rangle, \ldots, \langle \mathcal{A}_n, L \rangle$ all the argument structures of L under $\mathcal{P}$ and $G(\langle \mathcal{A}_1, L \rangle, \mathcal{P}), G(\langle \mathcal{A}_2, L \rangle, \mathcal{P}), \ldots, G(\langle \mathcal{A}_n, L \rangle, \mathcal{P})$ the corresponding games for the arguments of L.

$$\{G(\langle \mathcal{A}_1, L \rangle, \mathcal{P}), G(\langle \mathcal{A}_2, L \rangle, \mathcal{P}), \ldots, G(\langle \mathcal{A}_n, L \rangle, \mathcal{P})\}$$

is the *game family of* L and we denote it as $\mathcal{F}(L, \mathcal{P})$.

Definition 7 Let a be the first proponent movement in the game. A sequence s is complete if $s = [a]s_1$, with s_1 potentially empty, then there is no movement $b \in M_{G(\langle \mathcal{A}, L\rangle, \mathcal{P})}$ such that $[a]s_1[b] \in P_{G(\langle \mathcal{A}, L\rangle, \mathcal{P})}$. A sequence s is preferred if each opponent movement has a proponent answer. In other words, a sequence s is preferred if $|s|$ is odd.

Definition 8 A strategy over a game G is a set of sequences S, such that for all sequence $s \in S$, either:

- s is preferred; or
- there exists other sequence $s' \in S$, such that s' is preferred and s and s' has a prefix t, $|t| = n$, n is even and $s_{n+1} \neq s'_{n+1}$.

Definition 9 Let $\mathcal{P}$ be a *de.l.p.*, $L \in Lit$ and $G(\langle \mathcal{A}, L\rangle, \mathcal{P}) \in \mathcal{F}(L, \mathcal{P})$. We say that *P wins the game* $G(\langle \mathcal{A}, L\rangle, \mathcal{P})$ or that $G(\langle \mathcal{A}, L\rangle, \mathcal{P})$ *is won by P*, if the set of complete sequences of $P_{G(\langle \mathcal{A}, L\rangle, \mathcal{P})}$ is an strategy. Otherwise, we say that *O wins the game* or that $G(\langle \mathcal{A}, L\rangle, \mathcal{P})$ *is won by O*.

A player can win a game even though he does not win every complete sequence in such game. In (Prakken & Sartor 1997) the authors have developed an argument-based extended logic programming system which differs from DeLP in its winning rule: a player wins a dialogue tree if and only if he wins all the branches of the tree.

Definition 10 Let $\mathcal{P}$ be a *de.l.p.*. A *game-based interpretation for* $\mathcal{P}$, or G-Interpretation for $\mathcal{P}$ for short, is a tuple $\langle T, F\rangle$, such that T and F are subsets of atoms of the underlying signature of $\mathcal{P}$ and $T \cap F = \varnothing$.

In the previous definition T stands for true while F stands for false. The set of atoms UNDECIDED is defined as the set $U = Lit^+ - \{T \cup F\}$.

Each game can finish in two possible ways: won by the proponent P or won by the opponent O. There is no possibility for a draw. As the first move is made by the P, we are interested in those games won by this player.

Definition 11 Let $\mathcal{P}$ be a *de.l.p.*, h an atom of the underlying signature of $\mathcal{P}$, $\mathcal{F}(h, \mathcal{P})$ the game family for h and $\mathcal{F}(\overline{h}, \mathcal{P})$ the game family for $\overline{h}$ under a *de.l.p.* $\mathcal{P}$. A *game-based model for* $\mathcal{P}$, that we name G-Model of $\mathcal{P}$, is a G-interpretation $\langle T, F\rangle$ such that:

- If there exists a game $G(\langle \mathcal{A}, h\rangle$ in the family $\mathcal{F}(h, \mathcal{P})$ won by P, then h belongs to T.
- If there exists a game $G(\langle \mathcal{A}, \overline{h}\rangle, \mathcal{P})$ in the family $\mathcal{F}(\overline{h}, \mathcal{P})$ won by P, then h belongs to F.

Since we only consider literals under the signature of *de.l.p.*, the G-model definition does not contemplate the answer UNKNOWN. The minimal G-model defines a sound and complete semantics $\mathcal{GS}$ for DeLP (Cecchi & Simari 2004). We will say that $\mathcal{GS}$ entails a literal L from a *de.l.p.* $\mathcal{P}$, denoted by $\mathcal{P} \models_{\mathcal{GS}} L$, whenever $L \in T$ or $\overline{L} \in F$, being $\langle T, F\rangle$ the minimal G-model of $\mathcal{P}$.

The following theorem relates proof theory and game-based semantics, showing soundness and completeness.

Theorem 1 Let $\mathcal{P}$ be a *de.l.p.* and L a literal. L is warranted under $\mathcal{P}$ if and only if L belongs to the set T or $\overline{L}$ belongs to the set F of the minimal G-models $\langle T, F\rangle$ of $\mathcal{P}$ under $\mathcal{GS}$ semantics.

We have briefly presented the DeLP language, its proof theory and its declarative game-based semantics $\mathcal{GS}$. Now, we will be able to analyze the system and study some complexity properties.

Discussion on $\mathcal{GS}$ Complexity

DeLP is a defeasible reasoning system where every consequence of a *de.l.p.* is analyzed considering all the arguments for and against it. The trivalued game semantics $\mathcal{GS}$ characterizes such reasoning by two sets T and F, since $T \cup \overline{F}$ is the set of all warranted literals. Undecided literals are the remaining literals L for which there is no warrant for it nor for its complement. When considering DeLP in relation to game semantics, there are two relevant computational decision problems to analyze in the context of a *de.l.p.* $\mathcal{P}$:

- GAMESAT: Deciding whether there is a game for a literal α won by the proponent P in the context of a *de.l.p.* $\mathcal{P}$.
- NOWINGAME: Deciding whether there is no game for a literal α neither for the complement of α won by the proponent P in the context of a *de.l.p.* $\mathcal{P}$.

The former problem involves just finding a game that is won by the proponent. In order to capture the latter, it is necessary to find all the games for the literal and for its complement, and to establish that none of them is won by the proponent.

A positive GAMESAT answer for a given *de.l.p.* $\mathcal{P}$ and a literal L implies that $L \in T$, being $\langle T, F\rangle$ the minimal G-model, i.e., $\mathcal{P} \models_{\mathcal{GS}} L$. A positive GAMESAT answer for $\overline{L}$ means that $L \in F$ in the minimal G-model, i.e., $\mathcal{P} \models_{\mathcal{GS}} \overline{L}$.

The NOWINGAME decision problem for a *de.l.p.* $\mathcal{P}$ and a literal L is equivalent to determining if given the minimal G-model $\langle T, F\rangle$ of $\mathcal{P}$, $L \in Lit^+ - \{T \cup F\}$, i.e., $\mathcal{P} \not\models_{\mathcal{GS}} L$ and $\mathcal{P} \not\models_{\mathcal{GS}} \overline{L}$.

In this case, three interesting situation can be contemplated, and establish the followings decision problems:

- *Whether there is no game for a literal L, neither for its complement $\overline{L}$.* The game families for a literal L and for its complement $\overline{L}$, $\mathcal{F}(L, \mathcal{P})$ and $\mathcal{F}(\overline{L}, \mathcal{P})$ respectively, are empty. L has no argument neither for nor against it. Therefore, the agent has no information about such query.
- *Whether there is no game for a literal L, and the non empty set of all games in the family of its complement $\overline{L}$ are won by the opponent.* The game family for a literal L, $\mathcal{F}(L, \mathcal{P})$, is empty and only games won by the opponent are in the non empty family $\mathcal{F}(\overline{L}, \mathcal{P})$. L has no argument for and all the arguments for its complement are defeated. Therefore, the agent has no information for L, and he cannot defend its complement. In a similarly way, we can define the case where the agent cannot defend a literal L, and has no information about its complement.

- *Whether all games in the non empty families for L and for its complement $\overline{L}$ are won by the opponent.* $\mathcal{F}(L, \mathcal{P})$ and $\mathcal{F}(\overline{L}, \mathcal{P})$ are non empty set and all the argument are defeated. The agent cannot defend any argument neither for nor against the literal L.

In order to determine the computational complexity of the decision problems introduced above, we will study DeLP from two approaches: combined and data complexity. Combined complexity of a fragment of logic programming has been defined and used in (Dantsin *et al.* 2001):

Complexity of (some fragment of) logic programming: is the complexity of checking if for variable programs $\mathcal{P}$ and variable ground atoms A, $\mathcal{P} \models A$.

On the other hand, the notion of data complexity is borrowed from relational database theory (Vardi 1982). Databases are nowadays the main tool for storing and retrieving very large sets of data. Data complexity allows us to study DeLP as a query language measuring its complexity focus on the size of the databases, and using defeasible and strict rules for inference purpose. Data complexity is a key measure to determine the efficiency of argumentation system implementations based on database technologies.

For methodological and complexity issues, it is important to distinguish in a *de.l.p.* the input data from the inference rules. Thus, hereafter, we will denote $\mathcal{P} = \langle \Pi_F, \Pi_R \cup \Delta \rangle$, where Π_F is a finite set of ground facts, and $\Pi_R \cup \Delta$ is a finite set of ground strict and defeasible rules. Making an analogy with database concepts, Π_F represents the input databases, also called the *extensional part*, and $\Pi_R \cup \Delta$ are the inference rules, called the *intensional part* of the database. We define a Boolean query as a finite set of strict and defeasible rules together with a ground literal L. The intended intuitive meaning of defining such query is the following: we want to know whether a literal L is entailed by $\mathcal{GS}$ from $\Pi_R \cup \Delta$ together with the database Π_F.

Following the principle and notions above, in the context of DeLP we will define data, program and combined complexity as follows.

Definition 12 Let Ω be any of the decision problems introduced above, $\mathcal{P} = \langle \Pi_F, \Pi_R \cup \Delta \rangle$ and $(\Pi_R \cup \Delta, L)$ a query:

- The *data complexity* of Ω is the complexity of Ω when the query is fixed, and the database varies, i.e., parameters $\Pi_R \cup \Delta$ and L are fixed.
- The *program or expression* complexity of Ω is the complexity of Ω when the database instance is fixed, and the query varies, i.e., the parameter Π_F is fixed.
- The *combined complexity* of Ω is the complexity where every parameter Π_F, $\Pi_R \cup \Delta$ and L vary.

Expression and combined complexity are quite close and they are rarely differentiated. For this reason we will only discuss data and combined complexity.

In order to carry out this complexity analysis we will first focus on the complexity of determining whether there is an argument $\mathcal{A}$ for a literal L. Then we will study if the game played with initial move $\mathcal{A}$ is won by the proponent.

The complexity of computing arguments

Arguments and counterarguments are the movements in a game, and hence the core of DeLP. Dung's formalism (Dung 1995) and some extensions that have been developed (Bench-Capon 2002; 2003; Amgoud & Cayrol 2002), offer a powerful tool for the abstract analysis of defeasible reasoning. However, these approaches operate with arguments and their attack and defeat relation at an abstract level, avoiding to deal with the underlying logical language used to structure the arguments. On the other hand DeLP does construct the arguments and analyzes the defeater relationship. Thus, studying the decision problem: "is a given subset of defeasible rules an argument for a literal under a *de.l.p.*?" is of central importance.

Following the definition of argument this problem has three parts: is L a consequence of $\Pi \cup \mathcal{A}$?, is $\Pi \cup \mathcal{A}$ consistent?, and is there a subset $\mathcal{A}'$ of $\mathcal{A}$ such that it is consistent with Π and that together with Π derives L?

Let $\mathcal{P} = \langle \Pi_F, \Pi_R \cup \Delta \rangle$ be a *de.l.p.*, L be a literal and $\mathcal{A} \subseteq \Delta$. The first condition of definition 4, that involves rigorous consequences concept is $L \in Cn_R(\Pi \cup \mathcal{A})$. In (Cecchi & Simari 2000), we have defined the following transformation Φ from a *de.l.p.* into a propositional definite logic program, i.e., a propositional logic program with just Horn clauses. Let A be an atom. $\Phi(A) = A$, $\Phi(\sim A) = A'$ where A' is a new atom not in the signature of the *de.l.p.* and the transformation of a conjunction is $\Phi(A, B) = \Phi(A), \Phi(B)$. $\Phi(H \prec B) = \Phi(H) \leftarrow \Phi(B)$ and all other rules remain the same $\Phi(H \leftarrow B) = \Phi(H) \leftarrow \Phi(B)$. We will use this transformation, and the following lemma in order to reduce the rigorous consequences of a *de.l.p.* into consequences of propositional Horn clauses.

Lemma 1 Let DP be a definite logic program, and $\mathcal{M}$ be the minimal model of DP, then $\mathcal{M} = Cn_R(DP)$.

We are interested in computing the time complexity of verifying whether $L \in Cn_R(\Pi \cup \mathcal{A})$. We shall construct a logic program with just Horn clauses, denoted $\mathcal{HP}(\Pi, \mathcal{A}, L)$ such that $L \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$.

Suppose that $A_1, \ldots, A_n$ are all the atoms in $\Pi \cup \mathcal{A}$. We define $\mathcal{HP}(\Pi, \mathcal{A}, L)$ as follows:

$$\begin{aligned} \mathcal{HP}(\Pi, \mathcal{A}, L) \quad = \quad & \Phi(\Pi) \cup \Phi(\mathcal{A}) \cup \{yes \leftarrow \Phi(L)\} \cup \\ & \{yes \leftarrow \Phi(A_i), \Phi(\overline{A_i}) : 1 \leq i \leq n\} \end{aligned}$$

Even though the SAT decision problem is **NP**-complete, both checking whether a definite propositional logic program $\mathcal{DP}$ satisfies a ground atom A, i.e., $\mathcal{DP} \models A$, and HORNSAT, i.e., the decision problem whether there is a truth assignment that satisfies a collection of Horn clauses, are **P**-complete (Dantsin *et al.* 2001; Papadimitriou & Yannakakis 1997).

Lemma 2 $\mathcal{HP}(\Pi, \mathcal{A}, L)$ is a transformation from a *de.l.p.* P into propositional Horn clauses such that verifying whether a literal L belongs to $Cn_R(\mathcal{P})$ is equivalent to verifying

Algorithm: Minimal
Input: $\mathcal{A}$ an argument for a literal L, and Π a set of strict rules.
Output: true if A is a minimal argument for L, false otherwise

minimal=true
Aux= $\mathcal{A}$
While minimal and not $Aux = \varnothing$ do
select $H \prec B \in$ Aux
$A' = \mathcal{A} - \{H \prec B\}$
if $L \in Cn_R(\Pi \cup A')$
then minimal=false
else Aux= Aux - $\{H \prec B\}$

Figure 1: Algorithm for verifying if a set of defeasible rules is minimal with respect to set inclusion for deriving a literal L.

whether yes is entailed from the transformed propositional Horn program. Thus, $L \in Cn_R(\mathcal{P})$ reduces to $\mathcal{DP} \models yes$, being $\mathcal{DP}$ a propositional Horn program.

Proof: In order to prove our claim, we have to establish that:

1. $L \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$.

   We will consider two cases:
   - $\Pi \cup \mathcal{A}$ is consistent.
     $L \in Cn_R(\Pi \cup \mathcal{A})$ if and only if $\Phi(L) \in \Phi(Cn_R(\Pi \cup \mathcal{A}))$ if and only if $\Phi(L) \in Cn_R(\Phi(\Pi \cup \mathcal{A}))$(see (Cecchi & Simari 2000)) if and only if, by lemma 1, $\Phi(L)$ is in the minimal model of $\Phi(\Pi \cup \mathcal{A})$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$ by the definition of minimal model, the monotonicity property and the use of the rule $yes \leftarrow \Phi(L)$.
   - $\Pi \cup \mathcal{A}$ is inconsistent.
     $L \in Cn_R(\Pi \cup \mathcal{A}) = Lit$ if and only if there exists $i, 1 \leq i \leq n$, such that $\Phi(L_i)$ and $\Phi(\overline{L_i})$ are in $\Phi(Cn_R(\Pi \cup \mathcal{A}))$ if and only if $\Phi(L_i)$ and $\Phi(\overline{L_i})$ are in the minimal model of $\mathcal{HP}(\Pi, \mathcal{A}, L)$ if and only if $\mathcal{HP}(\Pi, \mathcal{A}, L) \models yes$ by definition of minimal models, monotonicity property and the use of the rule $yes \leftarrow \Phi(L_i), \Phi(\overline{L_i})$.
2. $\mathcal{HP}$ is computed in logarithmic space: the transformation is quite simple, and is feasible in logarithmic space, since rules can be generated independently of each other except those of the form $yes \leftarrow \Phi(L_i), \Phi(\overline{L_i})$ which depends on the literal in the input.

Therefore $\mathcal{HP}(\Pi, \mathcal{A}, L)$ is a reduction from $L \in Cn_R(\Pi \cup \mathcal{A})$ into propositional Horn clauses. ■

Theorem 2 Let $\mathcal{P} = (\Pi_F, \Pi_R \cup \Delta)$ a *de.l.p.*, $\mathcal{A} \subseteq \Delta$, and L a literal. Determining whether $L \in Cn_R(\Pi \cup A)$ is **P**-complete.

Proof:
- *Membership:* Given a definite logic program $\mathcal{P}$ the least fixpoint T_P^∞ of the operator T_P can be computed in polynomial time (Papadimitriou 1994; Dantsin *et al.* 2001) : the number of iterations is bounded by the number of rules plus one. Each iteration step is feasible in polynomial time. Thus finding the minimal model of a logic program with just Horn clauses is in **P** (Dantsin *et al.* 2001).
  By lemma 2, $L \in Cn_R(\Pi \cup A)$ has been reduced to propositional logic programming. Therefore, $L \in Cn_R(\Pi \cup A)$ is in **P**.
- *Hardness:* Horn rules are strict rules in a *de.l.p.*, and the minimal model of a definite logic program $\mathcal{DP}$ is equal to $Cn_R(\mathcal{DP})$. Therefore, by applying reduction by generalization, we have that $\mathcal{DP} \models L$ reduce to $L \in Cn_R(\mathcal{DP})$. Propositional logic programming is **P**-complete (Dantsin *et al.* 2001). This suffices to complete the proof. ■

Until now we have proved that the first condition of argumentation definition is **P**-complete. Now we will analyze the rest of the issues we need for computing an argument. We will denote the cardinality of the language by $|Lit|$ and defeasible rules cardinality by $|\Delta|$.

In Figure 1, we present an algorithm for verifying whether a set of defeasible rules is minimal with respect to set inclusion for entail a literal L. Worst case of the minimality condition is considered assuming that the argument has at most $|\Delta|$ defeasible rules, i.e., Δ is an argument for some literal. Computing the minimality condition involves $|\Delta|$ loops verifying that $L \in Cn_R(\Pi \cup A')$, which is in **P**. Thus, this problem is solvable in polynomial time, and, therefore, it is in **P**.

Finally, to check whether the set of defeasible rules is consistent under a *de.l.p.*, we verify that there is no atom such that the atom and its complement are members of $Cn_R(\Pi \cup A)$. In the worst case, when $Cn_R(\Pi \cup A)$ is consistent, this algorithm must control every atom in the signature of the *de.l.p.*. Thus, to check if it is consistent is proportional to the number of atoms $|Lit|/2$ and therefore it is in **P**.

Theorem 3 The decision problem "is a given subset of defeasible rules an argument for a literal under a *de.l.p.*?" is **P**-complete.

Proof:
Membership: (sketch) From the above development it follows membership to **P**.
Hardness: We employ a reduction from $\mathcal{DP} \models L$, being $\mathcal{DP}$ a propositional Horn program. Consider the following transformation $r(\mathcal{DP}) = \mathcal{DP}' = \langle \Pi, \Delta \rangle$, where $\Pi = \mathcal{DP}$ and Δ is empty. r is a transformation computed in logarithmic space such that whenever a literal L is entailed by a propositional Horn program $\mathcal{DP}$, the decision problem "is a given subset of $\Delta = \varnothing$ an argument for L under $\mathcal{DP}'$" finish in an accepting state.

L is in the minimal model of a propositional Horn program if and only if $L \in Cn_R(\mathcal{DP})$ if and only if $\varnothing$ is an argument for L, since is minimal and consistent with Π, if and only if "is a given subset of $\Delta = \varnothing$ an argument for

L under $\mathcal{DP}'$" finish in an accepting state. Thereby establishing that the decision problem "is a given subset of defeasible rules an argument for a literal under a *de.l.p.*?" is **P**-complete. ■

Our final aim is to determine the complexity of computing the set of all the arguments under a *de.l.p.*. This is motivated in that GAMESAT and NOWINGAME require for playing a game to compute every argument that defeats each argument introduced in a previous move. A subset $A \subseteq \Delta$ may be a potential argument of different literals in the language. Thus, the maximum number of checks for potential arguments that depends on the size of the set of defeasible rules and on the size of Lit, is $|Lit| * 2^{|\Delta|}$.

Lemma 3 Let AP be the polynomial time needed for the decision problem "is a given subset of defeasible rules an argument for a literal l under a *de.l.p.*?". Then, the upper bound time for computing all the arguments is $|Lit| * 2^{|\Delta|} * AP$.

The result above states an exponential upper bound for computing $\mathcal{X}$, the set of all the arguments in Dung's formalism (Dung 1995).

Even though we must verify whether every subset of Δ is an argument for every literal in the language of the *de.l.p.*, because the consistency condition in the definition of argument, $A \subseteq \Delta$ cannot be an argument for a literal and for its complement, so we will consider only $\frac{|Lit|}{2} * 2^{|\Delta|} = |Lit| * 2^{|\Delta|-1}$ potential arguments in order to play a game or equivalently to build the dialectical tree. This upper bound could be improved by considering minimality over the arguments, i.e., no $A_1 \subseteq \Delta$ would be an argument for a literal L if A_2 is an argument of L and $A_2 \subseteq A_1$.

Finally, we consider the argument existence decision problem.

Corollary 1 *(Argument Existence)* The decision problem "whether there is an argument for a literal L under a *de.l.p.*" is **NP**.

Proof: We can guess any subset of Δ, and verify whether this subset is an argument for a literal L under *de.l.p.* in polynomial time. This proves membership in **NP**. ■

These results contrast with those of (Parsons, Wooldridge, & Amgoud 2003), where determining whether there is an argument for a formula h is Σ_2^P-complete. Even thought there are some similarities between argument definitions, they differ in the underlying logic. While in DeLP approach an argument is a subset of defeasible rules, and the inference mechanism to obtain it is logic programming based, an argument in the formalism described in (Parsons, Wooldridge, & Amgoud 2003) is a subset of formulas of a propositional language, and $\vdash$ stands for classical inference.

Data Complexity for DeLP

In order to determine the upper bound for the data complexity of the decision problems GAMESAT and NOWINGAME, we will first analyze the dialectical tree structure over the size of the facts and the strict and defeasible rules.

The dialectical tree is explored in a complete depth first way, as minimax does. If the maximum depth of the tree is m, and there are b legal movements at each point, then the time complexity will be $O(b^m)$(Russell & Norvig 2003). If we implement the technique alpha-beta pruning, and we consider that successors are examined in random order, then the time complexity will be roughly $O(b^{\frac{3m}{4}})$(Russell & Norvig 2003). The maximum depth of a dialectical tree for an argument under a *de.l.p.* with $|\Delta|$ defeasible rules is $2^{|\Delta|}$, i.e., we can consider every potential argument in one branch of the tree. Any argument can appear more than once in the tree but at most once in every branch, because of the acceptable argumentation line definition. What about branch factor: there exists $|Lit|/2$ literals that can be in conflict with the last argument. These literals may have at most $2^{|\Delta|}$ potential arguments. So our branching factor is in the worst case $|Lit|/2 * 2^{|\Delta|}$. Thus, exploring the dialectical tree as minimax does has an upper bound of $O((|Lit|*2^{|\Delta|-1})^{2^{|\Delta|}})$.

Every time we must insert a neighbour node B of a node A in the tree structure or equivalently, when a player makes a move, we must check if it is a legal move in the game, i.e., if B attacks and defeats A, and if B does not introduce inconsistency. In order to determine whether B is a defeater of A, we must take into account the preference criterion between arguments. Any preference criterion defined among arguments could be used in DeLP. For this reason, the complexity class of the following decision problem "whether an argument can be considered in the tree structure of a game" will be left parameterized in the class C.

Theorem 4 Let C be the complexity class for the decision problem:"whether an argument can be considered in the tree structure of a game". The upper bound for data complexity of GAMESAT is $\mathbf{NP}^C$.

Proof: For fixed $\Pi_R \cup \Delta$, the size of the dialectical tree for an argument $\langle \mathcal{A}, L \rangle$ is polynomial in the size of the literals in Π_F. Furthermore, computing each argument is in **P**, and considering each argument in the tree structure is in C. In order to decide whether a literal L belongs to the set T of the minimal G-model, we guess for an argument of L such that the game played from this argument is won by the Proponent. The number of arguments is polynomial when $\Pi_R \cup \Delta$ is fixed, and determining whether the game is won by the Proponent can be done with a C oracle. This proves membership in $\mathbf{NP}^C$. ■

Since NOWINGAME is a conjunction of GAMESAT complements an immediate corollary to the result above follows naturally.

Corollary 2 Let C be the complexity class for the decision problem:"whether an argument can be considered in the tree structure of a game". The upper bound for data complexity of NOWINGAME is co-$\mathbf{NP}^C$.

Decision Problem	Complexity
Is $L \in Cn_R(Rules)$?	**P**-complete
Is $\langle \mathcal{A}, L \rangle$ an argument?	**P**-complete
Argument Existence	**NP**
GAMESAT	Data Complexity $\mathbf{NP}^C$
NOWINGAME	Data Complexity $co - \mathbf{NP}^C$

Table 1: Problems studied, and the main complexity results obtained.

Even though we have not analyzed in depth the complexity for computing the preference criterion, we illustrate this concept with two different cases.

In (Chesñevar & Maguitman 2004a; 2004b), the authors use specificity (Simari & Loui 1992) as a syntax-based criterion among conflicting arguments, preferring those arguments which are more informed or more direct, in order to assess natural language usage based on the web corpus and to evaluate and rank search results, respectively. Computing specificity depends strongly on the set $2^{|Lit|}$.

Other DeLP implementations use a static preference relation (Chesñevar *et al.* 2004). In this case, the preference criterion is computed by comparing arguments values. Such values are obtained through different mathematical formulas applied to the certainty of a formula in the language. Computing such preference criterion involves just a comparison between two certainty values. However, an extra cost is considered in the argument construction procedure, since the certainty value is computed keeping a trace of all uncertain information used to derive a goal.

Conclusion and Future Work

We have analyzed complexity of DeLP through the $\mathcal{GS}$ semantics, pointing out some relevant decision problems. In particular, we have analyzed in depth GAMESAT and NOWINGAME. In order to achieve our aim, we have distinguished database and a query from a *de.l.p.*, and we have defined data, expression and combined complexity in the context of DeLP. As far as we know, argumentation systems have not been studied yet as a query language, and, therefore, there is no previous data complexity analysis for defeasible reasoning. Table 1 summarizes the problems studied and the main complexity results obtained.

As DeLP do not assume as input the argument set, the first results that has been established where related to arguments, the movements of a game. We have focused on the existence of an argument in order to play a game, and on verifying whether a set is an argument. We state an exponential upper bound for the set of all the arguments. Because of the underpinning logic of DeLP our complexity results are a bit better than those based on classical logic.

Data complexity results on GAMESAT and NOWINGAME give a guideline for determining expressive power for DeLP. Since our results are parameterized, we can state a lower bound on **NP**, otherwise known as Σ_1^1, which coincides with the class of properties of finite structures expressible in existential second-order logic (Fagin 1974).

When analyzing Data complexity we have fixed the query and we have parametrized the preference criteria. Thus an interesting topic for future research is to study to what extent this results can be applied to others rule-based argumentation systems whose theory proof is rather similar.

As future work we will analyze combined complexity of the decision problems introduced. We are studying the expressive power of DeLP in order to compare this system with other non monotonic formalisms.

References

Abramsky, S., and McCusker, G. 1997. Game Semantics. In Schwichtenberg, H., and Berger, U., eds., *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School*. Springer-Verlag.

Amgoud, L., and Cayrol, C. 2002. A reasoning model based on the production of acceptable arguments. *Annals of Math and Artificial Intelligence* 34:197–215.

Atkinson, K.; Bench-Capon, T.; and Mc Burney, P. 2004. A dialogue game protocol for multi-agent argument over proposals for action. Technical Report ULCS-04-007, Department of Computer Science, University of Liverpool, Liverpool, U.K.

Bassiliades, N.; Antoniou, G.; and Vlahavas, I. 2004. A defeasible logic reasoner for the semantic web. In *Proc. of the Workshop on Rules and Rule Markup Languages for the Semantic Web*, 49–64.

Bench-Capon, T. J. M. 2002. Value-based argumentation frameworks. In *NMR 2002*, 443–454.

Bench-Capon, T. J. M. 2003. Persuasion in Practical Argument Using Value Based Argumentation Frameworks. *Journal of Logic and Computation* 13(3):429–448.

Bondarenko, A.; Dung, P.; Kowalski, R.; and Toni, F. 1997. An Abstract, Argumentation-Theoretic Approach to Default Reasoning. *Artificial Intelligence* 93(1-2):63–101.

Cadoli, M., and Schaerf, M. 1993. A survey of complexity results for nonmonotonic logics. *Journal of Logic Programming* 17:127–160.

Cecchi, L. A., and Simari, G. R. 2000. Sobre la Relación entre la Definición Declarativa y Procedural de Argumento. In *VI CACiC*, 465–476.

Cecchi, L. A., and Simari, G. R. 2004. Sobre la relación entre la Semántica GS y el Razonamiento Rebatible. In *X CACiC - Universidad Nacional de La Matanza*, 1883–1894.

Chesñevar, C., and Maguitman, A. 2004a. An Argumentative Approach to Assessing Natural Language Usage based on the Web Corpus. In *Proc. of the European Conference on Artificial Intelligence (ECAI) 2004*, 581–585.

Chesñevar, C., and Maguitman, A. 2004b. ARGUENET: An Argument-Based Recommender System for Solving Web Search Queries. In *Proc. of the 2nd IEEE Intl. IS-2004 Conference*, 282–287.

Chesñevar, C.; Simari, G.; Alsinet, T.; and Godo, L. 2004. A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge. In *Proc. of the UAI-2004*, 76–84.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)* 33(3):374 – 425.

Dimopoulos, Y.; Nebel, B.; and Toni, F. 2002. On the Computational Complexity of Assumption-based Argumentation for Default Reasoning. *Artificial Intelligence* 141(1):57–78.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming and n-person games. *Artificial Intelligence* 77:321–357.

Dunne, P. E., and Bench-Capon, T. 2002. Coherence in finite argument systems. *Artificial Intelligence* 141:187–203.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. In Karp, R., ed., *Complexity of Computation. SIAM-AMS Proceedings*, volume 7, 43–73.

García, A. J., and Simari, G. R. 2004. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Gordon, T., and Karacapilidis, N. 1997. The Zeno Argumentation Framework. In ACM., ed., *The Sixth International Conference on Artificial Intelligence and Law*, 10–18.

Lifschitz, V. 1996. Foundations of logic programming. In Brewka, G., ed., *Principles of Knowledge Representation*. CSLI Publications. 1–57.

Maher, M. J. 2001. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* 1(6):691–711.

Papadimitriou, C. H., and Yannakakis, M. 1997. On the complexity of database queries (extended abstract). In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 12–19. New York, NY, USA: ACM Press.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley Publishing Company.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2003. Properties and complexity of some formal inter-agent dialogue. *Journal of Logic and Computation* 13(3):347–376.

Pollock, J. 1987. Defeasible Reasoning. *Cognitive Science* 11:481–518.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorties. *Journal of Applied Non-Classical Logics* 7:25–75.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A modern approach*. New Jersey: Prentice Hall, second edition.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53:125–157.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC82*, 137–146. New York, NY, USA: ACM Press.

Verheij, B. 1998. Argumed - a template-based argument mediation system for laweyers. In Hage, J.; Bench-Capon, T. J.; Koers, A.; de Vey Mestdagh, C.; and Grütters, C., eds., *Legal Knowledge Based Systems. JURIX: The Eleventh Conference*, 113–130.

5.4 An Argumentation Framework for Concept Learning

An Argumentation Framework for Concept Learning

Leila Amgoud and **Mathieu Serrurier**
IRIT - CNRS
118, route de Narbonne, 31062 Toulouse, France
{amgoud, serrurier}@irit.fr

Abstract

Concept learning is an important problem in AI that consists of, given a set of training examples and counter-examples on a particular concept, identifying a model that is coherent with the training examples, i.e that classifies them correctly. The obtained model is intended to be reused for the purpose of classifying new examples. Version space is one of the *formal* frameworks developed for that purpose. It takes as input a consistent set of training examples on the concept to learn, a set of possible models, called *hypothesis* ordered by generality, and returns the hypothesis that are coherent with the training examples. The returned set of hypothesis is called *version space* and is described by its lower and upper bounds.

This paper provides an argumentation-based framework that captures the results of the version space approach. The basic idea is to construct arguments in favor of/against each hypothesis and training example, to evaluate those arguments and to determine among the conflicting arguments the acceptable ones. We will show that the acceptable arguments characterize the version space as well as its lower and upper bounds. Moreover, we will show that an argumentation-based approach for learning offers an additional advantage by allowing the handling of common problems in classical concept learning. Indeed, it is possible to reason directly with sets of hypothesis rather than one, and to deal with inconsistent sets of training examples. Lastly, the framework translates the problem of classifying examples into a decision one.

Introduction

Machine learning aims at building *models* that describe a *concept* from a set of *training examples*. The models are intended to be sufficiently general in order to be reused on new examples. When teexhe concept to learn is binary, i.e. examples of that concept can be either true or false, the problem is called *concept learning*. In (Mitchell 1982), Mitchel has proposed the famous general and abstract framework, called *version space learning*, for concept learning. That framework takes as input a consistent set of *training examples* on the concept to learn, a set of possible models, called *hypothesis* ordered by *generality*, and returns the hypothesis that are coherent with the training examples. The returned set of hypothesis is called *version space* and is described by its lower and upper bounds.
Besides, argumentation has become an Artificial Intelligence keyword for the last fifteen years, especially in sub-fields such as non monotonic reasoning, inconsistency-tolerant reasoning, multiple-source information systems (Amgoud & Cayrol 2002; Prakken & Sartor 1997; Simari & Loui 1992; Gómez & Chesñevar 2003). Argumentartion follows three steps: i) to construct arguments and counter-arguments for a statement, ii) to select the "acceptable" ones and, finally, iii) to determine whether the statement can be accepted or not.

This paper claims that argumentation can also be used as an alternative approach for concept learning, which not only retrieves in an elegant way the results of the version space learning framework, but also offers several other advantages. In this argumentation-based approach, the concept learning problem is reformulated as follows: given a set of examples (the training ones, and/or additional examples) and a set of hypothesis, what should be the class of a given example? To answer this question, arguments are constructed in favor of all the possible classifications of the examples. A classification can come either from an hypothesis or from a training example. The obtained arguments may be conflicting since it may be the case that the same example is affected to different classes. We will show that the acceptability semantics defined in (Dung 1995) allow us to identify and characterize the version space as well as its lower and upper bounds.
The framework presents also the following features that make it original and flexible:

1. it handles i) the case of a consistent set of training examples; ii) the case of an inconsistent set of training examples; and iii) the case of an empty set of training examples;
2. it allows one to reason directly on the set of hypothesis;
3. examples are classified on the basis of the whole set of hypothesis rather than only one hypothesis as it is the case in standard concept learning. Indeed, in the standard approach, a unique hypothesis is chosen, and all the examples are classified on the basis of that hypothesis.
4. it presents different original and intuitive decision criteria for choosing the class of an example.

The paper is organized as follows. We first present the version space learning framework, then we introduce the basic

argumentation framework of Dung (Dung 1995). The third section introduces our argumentation-based framework for learning.

Vesion Space Learning

The aim of this section is to introduce the version space framework developed by Mitchel in (Mitchell 1982). Let $\mathcal{X}$ denote a *feature space* used for describing examples. Elements of $\mathcal{X}$ may then be pairs (attribute, value), first order facts, ... This set $\mathcal{X}$ is supposed to be equipped with an equivalence relation $\equiv$. Let $\mathcal{U} = \{0, 1\}$ be a *concept space*, where 1 means that the example of the concept is positive, and 0 means that the example is negative.
The version space framework takes as input a *hypothesis space* $\mathcal{H}$, and a set $\mathcal{S}$ of m *training examples*.

$$\mathcal{S} = \{(x_i, u_i)_{i=1,...,m} \text{ s.t } x_i \in \mathcal{X} \text{ and } u_i \in \mathcal{U}\}$$

Note that the set $\mathcal{S}$ contains both positive examples (i.e. the value of $x-i$ is equal to 1), and negative ones (i.e. the value of $x - i$ is equal to 0). Otherwise, the learning problem becomes trivial and not genuine. An important notion in concept learning is that of consistency. In fact, a set of examples is said to be consistent if it does not contain two logically equivalent examples with two different values. Formally:

Definition 1 (Consistency) *The set $\mathcal{S}$ of training examples is* consistent *iff $\nexists$ (x_1, u_1), $(x_2, u_2) \in \mathcal{S}$ such that $x_1 \equiv x_2$ and $u_1 \neq u_2$. Otherwise, $\mathcal{S}$ is said* inconsistent.

Regarding the *hypothesis space* $\mathcal{H}$, it may be, for instance, decision trees, propositional sets of rules, neural nets ... An *hypothesis* h is a mapping from $\mathcal{X}$ to $\mathcal{U}$ (i.e. h: $\mathcal{X} \mapsto \mathcal{U}$). The set $\mathcal{H}$ is supposed to be equipped with a *partial preorder* $\succeq$ that reflects the idea that some hypothesis are more general than others in the sense that they classify positively more examples. This preorder defines a lattice on the hypothesis space. Formally:

Definition 2 (Generality order on hypothesis) *Let h_1, h_2 $\in \mathcal{H}$. h_1 is* more general *than h_2, denoted by $h_1 \succeq h_2$, iff $\{x \in \mathcal{X} | h_1(x) = 1\} \supseteq \{x \in \mathcal{X} | h_2(x) = 1\}$.*

Before defining the output of the framework, let us first introduce a key notion, that of *soundness*.

Definition 3 (Soundness) *Let $h \in \mathcal{H}$. An hypothesis h is* sound *with respect to a training example $(x, u) \in \mathcal{S}$ iff $h(x) = u$. If $\forall (x_i, u_i) \in \mathcal{S}$, h is sound w.r.t (x_i, u_i), then h is said to be* sound *with $\mathcal{S}$.*

The framework identifies the *version space*, which is the set $\mathcal{V}$ of all the hypothesis of $\mathcal{H}$ that are sound with $\mathcal{S}$. The idea is that a "good" hypothesis should at least classify the training examples correctly.

Definition 4 (Version space)

$$\mathcal{V} = \{h \in \mathcal{H} | \text{ h is sound with } \mathcal{S}\}$$

Version space learning aims at identifying the *upper* and the *lower* bounds of this version space $\mathcal{V}$. The upper bound will contain the most general hypothesis, i.e the ones that classify more examples, whereas the lower bound will contain the most specific ones, i.e the hypothesis that classify less examples.

Definition 5 (General hypothesis) *The set of* general hypothesis *is $\mathcal{V}_G = \{h \in \mathcal{H} \mid h$ is sound with $\mathcal{S}$ and $\nexists\, h' \in \mathcal{H}$ with h' sound with $\mathcal{S}$, and $h' \succeq h\}$.*

Definition 6 (Specific hypothesis) *The set of* specific hypothesis *is $\mathcal{V}_S = \{h \in \mathcal{H} \mid h$ is sound with $\mathcal{S}$ and $\nexists\, h' \in \mathcal{H}$ with h' sound with $\mathcal{S}$, and $h \succeq h'\}$.*

From the above definition, we have the following simple property characterizing the elements of $\mathcal{V}$.

Property 1 *(Mitchell 1982)*

$$\mathcal{V} = \{h \in \mathcal{H} | \exists h_1 \in \mathcal{V}_S, \exists h_2 \in \mathcal{V}_G, h_2 \succeq h \succeq h_1\}$$

In (Mitchell 1982), an algorithm that computes the version space $\mathcal{V}$ by identifying its upper and lower bounds $\mathcal{V}_S$ and $\mathcal{V}_G$ has been proposed.

The above framework has some limits. First, finding the version space is not sufficient for classifying examples out of the training set. This is due to possible conflicts between hypothesis. Second, it has been shown that the complexity of the algorithm that identifies $\mathcal{V}_S$ and $\mathcal{V}_G$ is very high. In order to palliate that limit, learning algorithms try in general to reach only one hypothesis in the version space by using heuristical exploration of $\mathcal{H}$ (from general to specific exploration, for instance FOIL (Quinlan 1990), or from specific to general exploration, for instance PROGOL (Muggleton 1995)). That hypothesis is then used for classifying new objects. Moreover, it is obvious that this framework does not support inconsistent set of examples:

Property 2 *(Mitchell 1982) If the set $\mathcal{S}$ is inconsistent, then the version space $\mathcal{V} = \emptyset$.*

A consequence of the above result is that no concept can be learned. This problem may appear in the case of noisy training data set.

Let us illustrate the above definitions through the following example in which we try to learn the concept "a sunny day".

Example 1 (Learning the concept sunny day) *In this example, the feature space is a pair (attribute, value). Three attributes are considered: pressure, temperature, and humidity. Four training examples are given, and are summarized in Table below. For instance (pressure, low), (temperature, medium), and (humidity, high) is a negative example for the concept a sunny day, whereas the (pressure, medium), (temperature, medium), and (humidity, low) is a positive one.*

pressure	*temperature*	*humidity*	*sunny*
low	*medium*	*high*	*0*
medium	*medium*	*low*	*1*
low	*medium*	*medium*	*0*
medium	*high*	*medium*	*1*

Let us suppose that the hypothesis space $\mathcal{H}$ is the space of constrains *on the values of each attribute. Indeed, the constraints are conjunctions of accepted values of attributes.*

The special constraint $\emptyset$ (resp. $?$) means that no (resp. all) values of attributes are accepted. If a vector of values of attributes match all the constraints, then it is considered as a positive *example, otherwise it is a* negative *one. The hypothesis $\langle \emptyset, \emptyset, \emptyset \rangle$ and $\langle ?, ?, ? \rangle$ are respectively the lower and the upper bound of the hypothesis space $\mathcal{H}$. Using the version space learning algorithm, we get: $\mathcal{V}_G$ = $\{\langle$ medium $\vee$ high, ?, ?$\rangle\}$ and $\mathcal{V}_S$ = $\{\langle$ medium, medium $\vee$ high, low $\vee$ medium$\rangle\}$. Here $\mathcal{V}_S$ and $\mathcal{V}_G$ contain both only one hypothesis. The hypothesis in V_G, for instance, considers as positive examples of the sunny day concept all features that have $medium$ or $high$ values for the pressure attribute.*

Abstract Argumentation Framework

Argumentation is a reasoning model that follows the following steps:

1. Constructing *arguments* and counter-arguments.
2. Defining the *strengths* of those arguments.
3. Evaluating the *acceptability* of the different arguments.
4. Concluding or defining the *justified conclusions*.

In (Dung 1995), an argumentation system is defined as follows:

Definition 7 (Argumentation system) *An* argumentation system *(AS) is a pair $\langle \mathcal{A}, \mathcal{R} \rangle$. $\mathcal{A}$ is a set arguments and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is a defeasibility relation. We say that an argument A defeats an argument B iff $(A, B) \in \mathcal{R}$ (or $A\ \mathcal{R}\ B$).*

Note that to each argumentation system is associated an oriented graph whose nodes are the different arguments, and the edges represent the defeasibility relationship between them. Among all the conflicting arguments, it is important to know which arguments to keep for inferring conclusions or for making decisions. In (Dung 1995), different semantics for the notion of acceptability have been proposed. Let's recall them here.

Definition 8 (Conflict-free, Defence) *Let $\mathcal{B} \subseteq \mathcal{A}$.*

- *$\mathcal{B}$ is* conflict-free *iff there exist no A_i, $A_j \in \mathcal{B}$ such that $A_i\ \mathcal{R}\ A_j$.*
- *$\mathcal{B}$* defends *an argument A_i iff for each argument $A_j \in \mathcal{A}$, if $A_j\ \mathcal{R}\ A_i$, then there exists $A_k \in \mathcal{B}$ such that $A_k\ \mathcal{R}\ A_j$.*

Definition 9 (Acceptability semantics) *Let $\mathcal{B}$ be a conflict-free set of arguments, and let $\mathcal{F}$: $2^{\mathcal{A}} \mapsto 2^{\mathcal{A}}$ be a function such that $\mathcal{F}(\mathcal{B}) = \{A \mid \mathcal{B}$ defends $A\}$.*

- *$\mathcal{B}$ is a* complete extension *iff $\mathcal{B} = \mathcal{F}(\mathcal{B})$.*
- *$\mathcal{B}$ is a* grounded extension *iff it is the minimal (w.r.t. set-inclusion) complete extension.*
- *$\mathcal{B}$ is a* preferred extension *iff it is a maximal (w.r.t. set-inclusion) complete extension.*
- *$\mathcal{B}$ is a* stable extension *iff it is a preferred extension that defeats all arguments in $\mathcal{A} \backslash \mathcal{B}$.*

Let $\{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ be the set of all possible extensions under a given semantics.

Note that there is only one grounded extension which may be empty. It contains all the arguments which are not defeated, and also the arguments which are defended directly or indirectly by non-defeated arguments.
The last step of an argumentation process consists of determining, among all the conclusions of the different arguments, the "good" ones, called *justified conclusions*.

An Argumentation Framework for Concept Learning

The aim of this section is to propose an instantiation of the general and abstract framework of Dung that allows learning concepts from sets of trainng examples. We will show that this argumentation-based model captures the results of the version space learning presented in a previous section. The sets of *version space*, *specific* and *general* hypothesis are characterized in our model. Since the classical approach of version space learning considers only the case where the set of training examples is consistent, we will present two versions of our model. In the first one, the set $\mathcal{S}$ is supposed to be consistent. This model is then generalized to the case where $\mathcal{S}$ can be inconsistent. We will show that even in this latter case, the version space $\mathcal{V}$ is not always empty, thus it is still possible to learn concepts.

Throughout this section, we will consider a features space $\mathcal{X}$, a concept space $\mathcal{U} = \{0, 1\}$, a hypothesis space $\mathcal{H}$, which is equipped with a partial preordering $\succeq$ (see Definition 2), and a set $\mathcal{S}$ of $m > 0$ training examples.

Consistent Case

In order to instantiate the abstract framework of Dung, one needs to define the set $\mathcal{A}$ or arguments as well as the defeasibility relationship between those arguments. In our particular application, one needs to argue about particular classifications, thus arguments are constructed in favor of assigning particular values from $\mathcal{U}$ to an example in $\mathcal{X}$. Indeed, an argument in favor of the pair (x, u) represents the reason of assigning the value u to the example x. Two reasons can be distinguished:

1. (x, u) is a training example in $\mathcal{S}$,
2. there exists a hypothesis $h \in \mathcal{H}$ that classifies x in u.

Definition 10 (Argument) *An* argument *is a triplet $A = \langle h, x, u \rangle$ such that:*

1. *$h \in \mathcal{H}$, $x \in \mathcal{X}$, $u \in \mathcal{U}$*
2. *If $h \neq \emptyset$, then $u = h(x)$*
3. *If $h = \emptyset$, then $(x, u) \in \mathcal{S}$*

h is called the support *of the argument, and (x, u) its* conclusion. *Let $\texttt{Example}(A) = x$, and $\texttt{Value}(A) = u$.*
Let $\mathcal{A}$ be the set of arguments built from $(\mathcal{H}, \mathcal{X}, \mathcal{U})$.

Note that from the above definition for any trainng example $(x_i, u_i) \in \mathcal{S}$, $\exists \langle \emptyset, x_i, u_i \rangle \in \mathcal{A}$. Let $\mathcal{A}_{\mathcal{S}} = \{\langle \emptyset, x, u \rangle \in \mathcal{A}\}$ (i.e. the set of arguments coming from the training examples).

Property 3 *Let $\mathcal{S}$ be a set of training examples. $|\mathcal{S}| = |\mathcal{A}_{\mathcal{S}}|$*[1].

Proof *This follows from the above definition, and from the fact that a hypothesis h cannot be empty.* ■

Since the set $\mathcal{S}$ of training examples is not empty, then $\mathcal{A}_{\mathcal{S}}$ is not empty as well.

Property 4 $\mathcal{A}_{\mathcal{S}} \neq \emptyset$.

Proof *This follows directly from the above property, i.e. $|\mathcal{S}| = |\mathcal{A}_{\mathcal{S}}|$, and the assumption that $\mathcal{S} \neq \emptyset$.* ■

Let us illustrate the notion of argument through example 1.

Example 2 *In example 1, there are exactly four arguments with an empty support, and they correspond to the training examples: $\mathcal{A}_{\emptyset}$ = $\{a_1 = \langle \emptyset$, (pressure, low) $\wedge$ (temperature, medium) $\wedge$ (humidity,high), 0$\rangle$,*
$a_2 = \langle \emptyset$, (pressure, medium) $\wedge$ (temperature, medium) $\wedge$ (humidity, low), 1$\rangle$,
$a_3 = \langle \emptyset$, (pressure, low) $\wedge$ (temperature, medium) $\wedge$ (humidity, medium), 0$\rangle$,
$a_4 = \langle \emptyset$, (pressure, medium) $\wedge$ (temperature, high) $\wedge$ (humidity, medium), 1$\rangle\}$. There are also arguments with a non-empty support such as:
$\langle a_5 = \langle$? , medium $\vee$ high, ?$\rangle$, (pressure, low) $\wedge$ (temperature, high) $\wedge$ (humidity, high), 1$\rangle\}$,
$a_6 = \langle\langle$ medium$\vee$ high, ?, ?$\rangle$, (pressure, low) $\wedge$ (temperature, high) $\wedge$ (humidity, high), 0$\rangle$,
$a_7 = \langle\langle$ medium, medium$\vee$ high, ?$\rangle$, (pressure, low) $\wedge$ (temperature, high) $\wedge$ (humidity, high), 0$\rangle$.

In (Amgoud & Cayrol 2002; Prakken & Sartor 1997; Simari & Loui 1992), it has been argued that arguments may have different strengths depending on the quality of information used to construct them. In (Simari & Loui 1992), for instance, arguments built from specific information are stronger than arguments built from more general ones. In our particular application, it is clear that arguments with an empty support are stronger than arguments with a non-empty one. This reflects the fact that classifications given by training examples take precedence over ones given by hypothesis in $\mathcal{H}$. It is also natural to consider that arguments using more general hypothesis are stronger than arguments with less general hypothesis.

Definition 11 (Comparing arguments) *Let $\langle h, x, u\rangle$, $\langle h', x', u'\rangle$ be two arguments of $\mathcal{A}$. $\langle h, x, u\rangle$ is* preferred *to $\langle h', x', u'\rangle$, denoted by $\langle h, x, u\rangle$* Pref *$\langle h', x', u'\rangle$, iff:*

- *$h = \emptyset$ and $h' \neq \emptyset$, or*
- *$h \succeq h'$.*

Property 5 *The relation Pref is a partial preorder.*

Proof *This is due to the fact that the relation $\succeq$ is a partial preorder.* ■

Now that the set of arguments is built, it is possible to define the defeasibility relation $\mathcal{R}$ between arguments in $\mathcal{A}$. Here again, there are two ways in which an argument A can attack another argument B:

1. by *rebutting* its *conclusion*. This situation occurs when the two arguments have contradictory conclusions, i.e. the same example is classified in different ways.
2. by *undercutting* its *support*. This occurs when the support of B classifies in a different way the example of the conclusion of A. However, this relation is only restricted to training examples. Indeed, only arguments built from training examples are allowed to undercut other arguments. The idea behind this is that training examples are the only, in some sense, certain information one has, and thus cannot be defeated by hypothesis. However, hypothesis have controversial status.

Definition 12 (Rebutting) *Let $\langle h, x, u\rangle$, $\langle h', x', u'\rangle$ be two arguments of $\mathcal{A}$.* $\langle h, x, u\rangle$ rebuts $\langle h', x', u'\rangle$ *iff $x \equiv x'$, $u \neq u'$.*

Example 3 *In example 2, we have for instance :*
a_5 rebuts a_6, a_5 rebuts a_7, a_6 rebuts a_5, *and* a_7 rebuts a_5.

Definition 13 (Undercutting) *Let $\langle h, x, u\rangle$, $\langle h', x', u'\rangle$ be two arguments of $\mathcal{A}$.* $\langle h, x, u\rangle$ undercuts $\langle h', x', u'\rangle$ *iff $h = \emptyset$ and $h'(x) \neq u$.*

Example 4 *In example 2, we have for instance :*
a_1 undercuts a_5, a_2 undercuts a_5, a_3 undercuts a_5, *and* a_4 undercuts a_5.

The rebutting and undercutting relations are used in most argumentation systems that handle inconsistency in knowledge bases.

Property 6 *If $\mathcal{S}$ is consistent, then $\nexists$ $A, B \in \mathcal{A}_{\mathcal{S}}$ such that A rebuts B, or A undercuts B.*

Proof *Let $A = \langle \emptyset, x, u\rangle$, $B = \langle \emptyset, x', u'\rangle \in \mathcal{S}$ such that A rebuts B. According to Definition 12, $x \equiv x'$ and $u \neq u'$. This contradicts the fact that $\mathcal{S}$ is consistent.* ■

The two above conflict relations are brought together in a unique relation, called "Defeat".

Definition 14 (Defeat) *Let $A = \langle h, x, u\rangle$, $B = \langle h', x', u'\rangle$ be two arguments of $\mathcal{A}$. A* defeats *B iff:*

1. *A rebuts (resp. undercuts) B, and*
2. *(A Pref B), or (not(A Pref B) and not(B Pref A))*

Example 5 *With the argument defined in ex. 2 we have for instance :*
a_1 defeats a_5, a_2 defeats a_5, a_3 defeats a_5, a_4 defeats a_5, a_5 defeats a_6, a_5 defeats a_7 *and* a_6 defeats a_5.

From the above definition, it is easy to check that an argument with a empty support cannot be defeated by an argument with a non-empty support.

Property 7 $\forall A \in \mathcal{A}_{\mathcal{S}}, \nexists B \in \mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$ *s.t B defeats A.*

Proof *Let $A \in \mathcal{A}_{\mathcal{S}}$ and $B \in \mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$ such that B defeats A. This means that B rebuts A (because according to Definition 13, an argument with a non-empty support cannot undercut an argument with an empty one. Moreover, according to Definition 14, we have either B Pref A, or (not(B Pref A) and not(A Pref B)). This is impossible because according to Definition 11, arguments in $\mathcal{A}_{\mathcal{S}}$ are always preferred to arguments with a non-empty support.* ■

[1] $||$ denotes the cardinal of a given set

The argumentation system for concept learning is then the following:

Definition 15 (Argumentation system) *An* argumentation system *for concept learning (ASCL) is a pair $\langle \mathcal{A}, defeat \rangle$, where $\mathcal{A}$ is the set of arguments (see Definition 10) and defeat is the relation defined in Definition 14.*

Let us now identify the acceptable arguments of the above ASCL. It is clear that the arguments that are not defeated at all will be acceptable. Let $\mathcal{C}$ denote that set of non-defeated arguments.

Proposition 1 *If $\mathcal{S}$ is consistent, then $\mathcal{A}_{\mathcal{S}} \subseteq \mathcal{C}$.*

Proof *Let $A \in \mathcal{A}_{\mathcal{S}}$. Let us assume that $\exists B \in \mathcal{A}$ such that B defeats A. According to Property 7, $B \notin \mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$. Thus, $B \in \mathcal{A}_{\mathcal{S}}$. Moreover, B defeats A means that B rebuts A. This means then that A classifies a training example in u, and B classifies an equivalent example in $u' \neq u$. This contradicts the fact that the set $\mathcal{S}$ is consistent.* ■

From the above proposition and Property 4, it is clear that the ASCL has a non-empty grounded extension.

Proposition 2 (Grounded extension) *If $\mathcal{S}$ is consistent, then the argumentation system $\langle \mathcal{A}, defeat \rangle$ has a non empty grounded extension $\mathcal{E}$.*

Proof *This is due to the fact that $\mathcal{A}_{\mathcal{S}} \neq \emptyset$ and $\mathcal{A}_{\mathcal{S}} \subseteq \mathcal{C}$.* ■

Note that the system $\langle \mathcal{A}, defeat \rangle$ is not always finite. By finite we mean that each argument is defeated by a finite number of arguments. This is due to the fact that $\mathcal{H}$ and $\mathcal{X}$ are not always finite.

Proposition 3 *If $\mathcal{H}$ and $\mathcal{X}$ are finite, then the system $\langle \mathcal{A}, defeat \rangle$ is finite.*

When an argumentation system is finite, its characteristic function $\mathcal{F}$ is continuous. Consequently, the least fixed point of this function can be defined by an iterative application of $\mathcal{F}$ to the empty set.

Proposition 4 *If the argumentation system $\langle \mathcal{A}, defeat \rangle$ is finite, then the grounded extension $\mathcal{E}$ is:*

$$\mathcal{E} = \bigcup \mathcal{F}^{i \geq 0}(\emptyset) = \mathcal{C} \cup [\bigcup_{i \geq 1} \mathcal{F}^i(\mathcal{C})].$$

Let us now analyze the other acceptability semantics, namely preferred and stable ones. From Proposition 2, one concludes that the ASCL $\langle \mathcal{A}, defeat \rangle$ has at least one non-empty preferred extensions.

Proposition 5 *If $\mathcal{S}$ is consistent, then the ASCL $\langle \mathcal{A}, defeat \rangle$ has $n \geq 1$ non-empty preferred extensions.*

Proof *In (Dung 1995), it has been shown that the grounded extension is included in very preferred extension. Since the grounded extension is not empty (according to Proposition 2, then there exists at least one non-empty preferred extension).* ■

In general, the preferred extensions of an argumentation system are not stable. However, in our ASCL these extensions coincide. This result is due to the fact that the oriented graph associated to the above ASCL has no odd length circuits. However, it may contain circuits of even length.

Proposition 6

- *The graph associated with the system $\langle \mathcal{A}, defeat \rangle$ has no odd length circuits.*
- *The preferred extensions and stable extensions of the system $\langle \mathcal{A}, defeat \rangle$ coincide.*

Proof (Sketch of the proof) ***Part 1:*** *Let A, B, C be three arguments such that A defeats B, B defeats C, and C defeats A.*

Case 1: *Let us suppose that $A \in \mathcal{A}_{\mathcal{S}}$.*
Accroding to Property 6, $B \in \mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$. According to Property 7, C should be in $\mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$. Contradiction because according to Property 7, C cannot defeat A, which is in $\mathcal{A}_{\mathcal{S}}$.

Case 2: *Let us suppose that $A, B, C \in \mathcal{A} \backslash \mathcal{A}_{\mathcal{S}}$. This means that A rebuts B, B rebuts C, and C rebuts A (according to Definition 13). Consequently, $\texttt{Example}(A) \equiv \texttt{Example}(B) \equiv \texttt{Example}(C)$, and $\texttt{Value}(A) \neq \texttt{Value}(B)$, $\texttt{Value}(B) \neq \texttt{Value}(C)$. Due to the fact that $\mathcal{U} = \{0, 1\}$, we have $\texttt{Value}(A) = \texttt{Value}(C)$. This contradicts the assumption that C rebuts A.*

Part 2: *This is a consequence of the fact that there is no odd circuits in the system.* ■

Note, however, that the intersection of all the preferred (stable) extensions coincides with the grounded extension.

Proposition 7 *Let $\langle \mathcal{A}, defeat \rangle$ be a ASCL. Let $\mathcal{E}$ be its grounded extension, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ its preferred (stable) extensions. $\mathcal{E} = \bigcap_{i=1,\ldots,n} \mathcal{E}_i$.*

Let us now show how the above ASCL can retrieve the results of the version space learning, namely the version space and its lower and upper bounds. Before doing that, we start first by introducing some useful notations.

Let $\texttt{Hyp}$ be a function that returns for a given set of arguments, their non empty supports. In other words, this function returns all the hypothesis used to build arguments:

Definition 16 *Let $T \subseteq \mathcal{A}$.*

$$\texttt{Hyp}(T) = \{h \mid \exists \langle h, x, u \rangle \in T \text{ and } h \neq \emptyset\}$$

Now we will show that the argumentation-based model for concept learning computes in an elegant way the version space $\mathcal{V}$ (see Definition 4).

Proposition 8 *Let $\langle \mathcal{A}, defeat \rangle$ be a ASCL. Let $\mathcal{E}$ be its grounded extension, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ its preferred (stable) extensions. If the set $\mathcal{S}$ is consistent then:*

$$\texttt{Hyp}(\mathcal{E}) = \texttt{Hyp}(\mathcal{E}_1) = \ldots = \texttt{Hyp}(\mathcal{E}_n) = \mathcal{V}$$

where $\mathcal{V}$ is the version space.

Proof *Let $\mathcal{E}_i$ be an extension under a given semantics.*

$\texttt{Hyp}(\mathcal{E}_i) \subseteq \mathcal{V}$**:** *Let $h \in \texttt{Hyp}(\mathcal{E}_i)$, then $\exists \langle h, x, u \rangle \in \mathcal{E}_i$.*
Let us assume that $\exists (x_i, u_i) \in \mathcal{S}$ such that $h(x_i) \neq u_i$. This means $\langle \emptyset, x_i, u_i \rangle$ undercuts $\langle h, x, u \rangle$ (according to Definition 13). Consequently, $\langle \emptyset, x_i, u_i \rangle$ defeats $\langle h, x, u \rangle$. However, according to Property 3, $\langle \emptyset, x_i, u_i \rangle \in \mathcal{A}_{\mathcal{S}}$, thus $\langle \emptyset, x_i, u_i \rangle \in \mathcal{E}_i$. Contradiction because $\mathcal{E}_i$ is an extension, thus by definition it is conflict-free.

$\mathcal{V} \subseteq \texttt{Hyp}(\mathcal{E}_i)$**:** *Let $h \in \mathcal{V}$, and let us assume that $h \notin \texttt{Hyp}(\mathcal{E}_i)$. Since $h \in \mathcal{V}$, then $\forall (x_i, u_i) \in \mathcal{S}$, $h(x_i) = u_i$ (1) Let $(x, u) \in \mathcal{S}$, thus $h(x) = u$ and consequently $\langle h, x, u\rangle \in \mathcal{A}$. Moreover, since $h \notin \texttt{Hyp}(\mathcal{E})$, then $\langle h, x, u\rangle \notin E$. Thus, $\exists \langle h', x', u'\rangle$ that defeats $\langle h, x, u\rangle$.*

- *Case 1: $h' = \emptyset$. This means that $\langle \emptyset, x', u'\rangle$ undercuts $\langle h, x, u\rangle$ and $h(x') \neq u'$ Contradiction with (1).*
- *Case 2: $h' \neq \emptyset$. This means that $\langle h', x', u'\rangle$ rebuts $\langle h, x, u\rangle$. Consequently, $x \equiv x'$ and $u \neq u'$. However, since $h \in \mathcal{V}$, then h is sound with $\mathcal{S}$. Thus, $\langle \emptyset, x, u\rangle$ defeats $\langle h', x', u'\rangle$, then $\langle \emptyset, x, u\rangle$ defeats $\langle h, x, u\rangle$. Since $\langle \emptyset, x, u\rangle \in \mathcal{S}$, then $\langle h, x, u\rangle \in \mathcal{F}(\mathcal{C})$ and consequently, $\langle h, x, u\rangle \in \mathcal{E}_i$. Contradiction.*

■

The above result is of great importance. It shows that to get the version space, one only needs to compute the grounded extension.
We can also show that if a given argument is is an extension $\mathcal{E}_i$, then any argument based on an hypothesis from the version space that supports the same conclusion is in that extension. Formally:

Proposition 9 *Let $\langle \mathcal{A}, defeat\rangle$ be a ASCL, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ its extensions under a given semantics. If $< h, x, u > \in \mathcal{E}_i$, then $\forall h' \in \mathcal{V}$ s.t. $h' \neq h$ if $h'(x) = u$ then $< h', x, u > \in \mathcal{E}_i$.*

Proof *Let $\mathcal{E}_i$ be a given extension, and let $\langle h, x, u\rangle \in \mathcal{E}_i$. Let $h' \in \mathcal{V}$ such that $h'(x) = u$. Let us assume that $\langle h', x, u\rangle \notin \mathcal{E}_i$.*

Case 1: *$\mathcal{E}_i \cup \{\langle h', x, u\rangle\}$ is not conflict-free. This means that $\exists \langle h", x", u"\rangle \in \mathcal{E}_i$ such that $\langle h", x", u"\rangle$ defeats $\langle h', x, u\rangle$. Consequently, $\langle h", x", u"\rangle$ undercuts $\langle h', x, u\rangle$ if $h" = \emptyset$, or $\langle h", x", u"\rangle$ rebuts $\langle h', x, u\rangle$ if $h" \neq \emptyset$.*
If $h" = \emptyset$, then $h'(x") \neq u"$, this contradicts the fact that $h' \in \mathcal{V}$.
If $h" \neq \emptyset$, then $x" \equiv x$ and $u" \neq u$ and either $h" \succeq h'$, or $h', h"$ are not comparable. Thus, $\langle h", x", u"\rangle$ rebuts $\langle h, x, u\rangle$. Since $\langle h, x, u\rangle$, $\langle h", x", u"\rangle \in \mathcal{E}_i$, then h and $h"$ are not comparable. But, this means that $\langle h, x, u\rangle$ defeats $\langle h", x", u"\rangle$, and $\langle h", x", u"\rangle$ defeats $\langle h, x, u\rangle$. Consequently, $\mathcal{E}_i$ is not conflict-free. Contradiction because $\mathcal{E}_i$ is an extension.

Case 2: *$\mathcal{E}_i$ does not defend $\langle h', x, u\rangle$. This means that $\exists \langle h", x", u"\rangle$ defeats $\langle h', x, u\rangle$.*

- *Case 1: $h" = \emptyset$. This means that $h'(x") \neq u"$. Contradiction because $h' \in \mathcal{V}$.*
- *Case 2: $h" \neq \emptyset$. This means that $x \equiv x"$, $u \neq u"$, and $h" \succeq h'$. Thus, $\langle h", x", u"\rangle$ rebuts $\langle h, x, u\rangle$.*
  If $h \succeq h"$, then $\langle h, x, u\rangle$ defeats $\langle h", x", u"\rangle$, thus $\langle h, x, u\rangle$ defends $\langle h', x, u\rangle$.
  If $h" \succeq h$, then $\langle h", x", u"\rangle$ defeats $\langle h, x, u\rangle$. However, since $\langle h, x, u\rangle \in \mathcal{E}$, then $\mathcal{E}$ defends $\langle h, x, u\rangle$ against $\langle h", x", u"\rangle$. Thus, $\mathcal{E}$ defends $\langle h', x, u\rangle$. Contradiction

■

Using the grounded extension, one can characterize the upper and the lower bounds of the version space. The upper bound corresponds to the most preferred w.r.t Pref arguments of the grounded extension, whereas the lower bound corresponds to the less preferred ones.

Proposition 10 *Let $\langle \mathcal{A}, defeat\rangle$ be a ASCL, and $\mathcal{E}$ its grounded extension.*

- *$\mathcal{V}_G$ = $\{h \mid \exists <h, x, u> \in \mathcal{E}$ s.t $\forall <h', x', u'> \in \mathcal{E}$, not $(<h', x', u'>$ Pref $<h, x, u>)\}$.*
- *$\mathcal{V}_S$ = $\{h \mid \exists <h, x, u> \in \mathcal{E}$ s.t $\forall <h', x', u'> \in \mathcal{E}$, not $(<h, x, u>$ Pref $<h', x', u'>)\}$.*

Proof

$\mathcal{V}_G$ = $\{h \mid \exists <h, x, u> \in \mathcal{E}$ s.t $\forall <h', x', u'> \in \mathcal{E}$, not $(<h', x', u'>$ Pref $<h, x, u>)\}$.

- *Let $h \in \mathcal{V}_G$, thus $h \in \mathcal{V}$, and $\forall h' \in \mathcal{V}$, $h \succeq h'$. Since $h \in \mathcal{V}$, thus, $h \in \texttt{Hyp}(\mathcal{E})$, with $\mathcal{E}$ an extension. Then, $\exists \langle h, x, u\rangle \in \mathcal{E}$. Since $h \succeq h'$ for any $h' \in \mathcal{V}$, then $h \succeq h'$ for any $h' \in \texttt{Hyp}(\mathcal{E})$. Thus, $\langle h, x, u\rangle$ Pref $\langle h', x', u'\rangle$, $\forall \langle h', x', u'\rangle \in \mathcal{E}$.*
- *Let $\langle h, x, u\rangle \in \mathcal{E}$ such that $\forall \langle h', x', u'\rangle \in \mathcal{E}$, and not$(\langle h', x', u'\rangle Pref \langle h, x, u\rangle)$. Thus, $h \in \texttt{Hyp}(\mathcal{E})$, and $\forall h' \in \texttt{Hyp}(\mathcal{E})$, not$(h' \succeq h)$, thus $h \in \mathcal{V}_G$.*

$\mathcal{V}_S$ = $\{h \mid \exists <h, x, u> \in \mathcal{E}$ s.t $\forall <h', x', u'> \in \mathcal{E}$, not $(<h, x, u>$ Pref $<h', x', u'>)\}$.

- *Let $h \in \mathcal{V}_S$, thus $\nexists h' \in \mathcal{V}$ such that $h \succeq h'$. Since $h \in \mathcal{V}_S$, then $h \in \mathcal{V}$ and consequently, $h \in \texttt{Hyp}(\mathcal{E})$. This means that $\exists \langle h, x, u\rangle \in \mathcal{E}$. Let us assume that $\exists \langle h', x', u'\rangle \in \mathcal{E}$ such that $\langle h, x, u\rangle \; Pref \; \langle h', x', u'\rangle$, thus $h \succeq h'$. Contradiction with the fact that $h \in \mathcal{V}_S$.*
- *Let $\langle h, x, u\rangle \in \mathcal{E}$ such that $\forall \langle h', x', u'\rangle \in \mathcal{E}$, and not$(\langle h, x, u\rangle Pref \langle h', x', u'\rangle)$, thus not$(h \succeq h')$. Since $h \in \mathcal{V}$, and $\forall h' \in \mathcal{V}$, not$(h \succeq h')$, then $h \in \mathcal{V}_S$.*

■

As said before, the last step of an argumentation process consists of defining the *status* of the conclusions, in our case, the classification of examples. In what follows we present two decision criteria: The first one, called universal vote, consists of accepting those classifications that are in any extension. However, it is clear that this kind of voting may not classify all the examples. Thus, we propose a second criterion, called majority vote, that allows to associate a class to each example. The conclusions here are the ones that are supported by a majority of arguments that appear in the different extensions. Formally:

Definition 17 *Let $\langle \mathcal{A}, defeat\rangle$ be a ASCL, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ its extensions under a given semantics. Let $x \in \mathcal{X}$ and $u \in \mathcal{U}$.*

Universal vote: *x is* universally classified *in u iff $\forall \mathcal{E}_i$, $\exists <h, x, u> \in \mathcal{E}_i$. UV denotes the set of all universally classified examples.*

Majority vote: *x is* majoritarily classified *in u iff $|\{< h, x, u > |\exists \mathcal{E}_i, < h, x, u > \in \mathcal{E}_i\}| \geq |\{< h, x, u' > |u' \neq u, \exists \mathcal{E}_i, < h, x, u' > \in \mathcal{E}_i\}|$. MV denotes the set of all majoritarily conclusions accepted by majority vote.*

The universally classified examples are those which are supported by arguments in all the extensions. From a learning point of view, these correspond to examples classified by the

most general hypothesis in the version space. Thus, according to Proposition 7, we have the following result:

Property 8 *Let $\langle \mathcal{A}, defeat \rangle$ be a ASCL, and $\mathcal{E}$ its grounded extension :*

$$UV = \{(x, u) | \exists < h, x, u > \in \mathcal{E}\}$$

It is easy to check that the set of universally classified examples is included in the set of majoritarily classified ones.

Property 9 *Let $\langle \mathcal{A}, defeat \rangle$ be a ASCL, and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ its extensions under a given semantics :*

$$UV \subseteq MV$$

Proof *This result follows directly from the fact that the extensions are conflict-free.* ■

Inconsistent Case

Let us now consider the case where $\mathcal{S}$ is inconsistent, with $\mathcal{S}$ can be partionned into $\mathcal{S}_1, \ldots, \mathcal{S}_n$, suth that each $\mathcal{S}_i$ is a maximal (for set inclusion) consistent subsets of $\mathcal{S}$. This means that some training examples are classified in different classes. However, all the elements of $\mathcal{S}$ are supposed to be equally preferred. In this case, two arguments supporting such conflicting training examples rebut each other, thus can defeat each other as well.

Let $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$ be the sets of arguments with an empty support and whose conclusions are respectively in the subsets of training examples $\mathcal{S}_1, \ldots, \mathcal{S}_n$. It is clear that each set $\mathcal{A}_{\mathcal{S}_i}$ is conflict-free, however, it is defeated by arguments in $\mathcal{A}_{\mathcal{S}_j}$ with $i \neq j$.

As in the consistent case, arguments with an empty support are preferred to arguments built from hypothesis. In what follows, $ASCL_i$ will denote the argumentation system in the inconsistent case. Note that all the above definitions of an argument, of defeat, do not change. It can be checked that the corresponding oriented graph of $ASCL_i$ does not contain odd length circuits.

Proposition 11

- *The graph associated to the system $ASCL_i$ has no odd length circuits.*
- *In the $ASCL_i$ $\langle \mathcal{A}, defeat \rangle$, preferred extensions and stable ones coincide.*

As a consequence of the above result, the system $ASCL_i$ has preferred extensions that are also stable. Moreover, the grounded extension of this system coincides with the intersection of all the preferred extensions. Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be the preferred extensions of that system, and $\mathcal{E}$ its grounded extension.

Proposition 12 *If $\mathcal{S}$ is inconsistent and non empty, then the $ASCL_i$ $\langle \mathcal{A}, defeat \rangle$ has still $n \geq 1$ non-empty preferred extensions. Moreover, $\mathcal{E} = \bigcap_{i=1,\ldots,n} \mathcal{E}_i$.*

However, the grounded extension can be empty in this particular case of inconsistent training examples. The above result is of great importance since it shows that even in this particular case of inconsistent training example, it is still possible to classify examples.

Note that each preferred/stable extension contains one of the sets $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$. Moreover, the same set $\mathcal{A}_{\mathcal{S}_i}$ may belong to several extensions at the same time. It can be shown that all the hypothesis that are used to build arguments in a given extension are sound with the subset of training examples of that extension. Indeed, for each consistent subset of $\mathcal{S}$, we get the extensions of the consistent case previously studied.

Case of an Empty Set of Training Examples

Another interesting case is when the set of training examples is empty. In this case, the learning problem consists of classifying examples only on the basis of a set $\mathcal{H}$ of hypothesis. This is, indeed, a particular case of the previous case where $\mathcal{S}$ is inconsistent. The corresponding argumentation system constructs then arguments only on the basis of hypothesis, thus there is no argument with an empty support. This system satisfies exactly the same properties as the $ASCL_i$. For instance, the corresponding graph has no odd length circuits. Moreover, it contains at least one non-empty preferred extension, which is also stable. The grounded extension of this sytstem is the intersection of all the preferred extensions.

Conclusion

This paper has proposed, to the best of our knowledge, the first argumentation-based framework for concept learning. This framework considers the learning problem as a process that follows four main steps: it first constructs arguments in favor of classifications of examples from a set of training examples, and a set of hypothesis. Conflicts between arguments may appear when two arguments classify the same example in different classes. Once the arguments identified, it is possible to compare them on the basis of their strengths. The idea is that arguments coming from the set of training examples are stronger than arguments built from the set of hypothesis. Similarly, arguments based on general hypothesis are stronger than arguments built from more specific hypothesis. Indeed, such preference relation between arguments ensures that during the learning process, only hypothesis that are sound with the training examples are kept, and general hypothesis are privileged to less specific ones. We have shown that acceptability semantics of the ASCL retrieves and even characterizes the version space and its upper and lower bounds. Thus, the argumentation-based approach gives another interpretation of the version space as well as its two bounds in terms of arguments. We have also shown that when the set of training examples is inconsistent, it is still possible to learn concepts, and to classify examples of it. Indeed, in this particular case, the version space is empty as it is the case in the version space learning framework. A last and not least feature of our framework consists of defining the class of each example on the basis of all the hypothesis and not only one, and also to suggest four intuitive decision criteria for that purpose.

A first extension of this framework would be to explore the proof theories in argumentation that test directly whether a given argument is in the grounded extension without com-

puting this last. This means that one may know the class of an example without exploring the whole hypothesis space.

This framework can easily be generalized to the case of classification problems in which an example can be affected to a class among a set of possible classes. It could also be extended in order to handle qualitative uncertainty and preferences on examples. It can also be extended to handle the regression problem in which the concept takes the form of a continuous function.

Acknowledgments

This work was supported by the Commission of the European Communities under contract IST-2004-002307, ASPIC project "Argumentation Service Platform with Integrated Components".

References

Amgoud, L., and Cayrol, C. 2002. Inferring from inconsistency in preference-based argumentation frameworks. *Int. Journal of Automated Reasoning* Volume 29 (2):125–169.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77:321–357.

Gómez, S. A., and Chesñevar, C. I. 2003. Integrating defeasible argumentation with fuzzy art neural networks for pattern classification. In *Proc. ECML'03*.

Mitchell, T. 1982. Generalization as search. *Artificial intelligence* 18:203–226.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing* 13:245–286.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7:25–75.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence and Law* 53:125–157.

5.5 An Abstract Model for Computing Warrant in Skeptical Argumentation Frameworks

An Abstract Model for Computing Warrant in Skeptical Argumentation Frameworks

Carlos Iván Chesñevar
Departament of Computer Science
Universitat de Lleida
Jaume II, 69 – 25001 Lleida, SPAIN
Email: cic@eps.udl.es

Guillermo Ricardo Simari
Dept. of Computer Science and Engineering
Universidad Nacional del Sur
Alem 1253 – 8000 Bahía Blanca, ARGENTINA
Email: grs@cs.uns.edu.ar

Abstract

Abstract argumentation frameworks have played a major role as a way of understanding argument-based inference, resulting in different argument-based semantics. The goal of such semantics is to characterize which are the rationally justified (or warranted) beliefs associated with a given argumentative theory. In order to make such semantics computationally attractive, suitable argument-based proof procedures are required, in which a search space of arguments is examined looking for possible candidates that warrant those beliefs. This paper introduces an abstract approach to model the computation of warrant in a skeptical abstract argumentation framework. We show that such search space can be defined as a lattice, and illustrate how the so-called dialectical constraints can play a role for guiding the efficient computation of warranted arguments.

Keywords: Argumentation, Defeasible Reasoning, Nonmonnotonic Reasoning.

Introduction and Motivations

Over the last ten years, interest in argumentation has expanded dramatically, driven in part by theoretical advances but also by successful demonstrations of a wide range of practical applications. In this context, abstract argumentation frameworks have played a major role as a way of understanding argument-based inference, resulting in different argument-based semantics. In order to compute such semantics, efficient argument-based proof procedures are required for determining when a given argument A is warranted. This involves the analysis of a potentially large search space of candidate arguments related to A by means of an attack relationship.

This paper presents a novel approach to model such search space for warrant computation in a skeptical abstract argumentation framework. We show that such search space can be defined as a lattice, and illustrate how some constraints (called dialectical constraints) can play a role for guiding the efficient computation of warranted arguments.

The rest of this paper is structured as follows. The next Section presents the basic ideas of an abstract argumentation framework with dialectical constraints, which includes several concepts common to most argument-based formalisms. The notion of argumentation line is presented, highlighting its role for modeling so-called dialectical trees as relevant useful data structures for computing warrant. After these preliminaries, the following Section shows how dialectical trees can be used to analyze the search space associated with computing warrants in an argumentation framework. We show that such search space can be represented as a lattice. Subsequently, we devote a Section to go into different criteria which can lead to compute warrant more efficiently on the basis of this lattice characterization. Finally, we discuss some related work and present the main conclusions that have been obtained.

An Abstract Argumentation Framework with Dialectical Constraints

Abstract argumentation frameworks (Dung 1993; Vreeswijk 1997; Jakobovits 1999; Jakobovits & Vermeir 1999) are formalisms for modelling defeasible argumentation in which some components remain unspecified. In such abstract frameworks usually the underlying knowledge representation language, the actual structure of an argument and the notion of attack among arguments are abstracted away, as the emphasis is put on different *argument-based semantics* which are associated with identifying sets of ultimately accepted arguments.

In this paper we are concerned with the study of warrant computation in argumentation systems, with focus on skeptical semantics for argumentation. As a basis for our analysis we will use an abstract argumentation framework (following Dung's seminal approach to abstract argumentation (Dung 1995; 1993)) enriched with the notion of *dialectical constraint*, which will allow us to model distinguished sequences of arguments. The resulting, extended framework will be called an *argumentation theory*.

Definition 1 (Dung 1995; 1993) An argumentation framework Φ is a pair $\langle \mathfrak{Args}, \mathbf{R} \rangle$, where $\mathfrak{Args}$ is a finite set of arguments and $\mathbf{R}$ is a binary relation between arguments such that $\mathbf{R} \subseteq \mathfrak{Args} \times \mathfrak{Args}$. The notation $(\mathcal{A}, \mathcal{B}) \in \mathbf{R}$ (or equivalently $\mathcal{A}\,\mathbf{R}\,\mathcal{B}$) means that $\mathcal{A}$ *attacks* $\mathcal{B}$.

Thus defined, an Argumentation Framework Φ can be seen as a collection of directed graphs (di-graphs) in which nodes correspond to arguments, and an edge between two nodes corresponds to an attack. We will write $\mathfrak{Lines}_\Phi$ to denote the set of all possible sequences of arguments

$[\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k]$ in Φ where for any pair of arguments $\mathcal{A}_i$, $\mathcal{A}_{i+1}$ it holds that $\mathcal{A}_i$ $\mathbf{R}$ $\mathcal{A}_{i+1}$, with $0 \leq i \leq k-1$. Argumentation lines define a domain onto which different kinds of *constraints* can be defined. As such constraints are related to sequences which resemble an argumentation dialogue between two parties, we call them *dialectical constraints*. Formally:

Definition 2 Let $\Phi = \langle \mathfrak{Args}, \mathbf{R} \rangle$ be an argumentation framework. A *dialectical constraint* $\mathbf{C}$ *in the context of* Φ is any function $\mathbf{C} : \mathfrak{Lines}_\Phi \rightarrow \{True, False\}$.

A dialectical constraint imposes a restriction characterizing when a given argument sequence λ is valid in a framework Φ (*i.e.*, $\mathbf{C}(\lambda) = True$). An argumentation theory is defined by combining an argumentation framework with a particular set of dialectical constraints. Formally:

Definition 3 An *argumentation theory* T (or just a *theory* T) is a pair $(\Phi, \mathbf{DC})$, where Φ is an argumentation framework, and $\mathbf{DC} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k\}$ is a finite (possibly empty) set of *dialectical constraints*.

Given a theory $T = (\Phi, \mathbf{DC})$, the intended role of $\mathbf{DC}$ is to avoid *fallacious* reasoning (Aristotle ; Hamblin 1970; Rescher 1977; Walton 1995) by imposing appropriate constraints on argumentation lines to be considered rationally *acceptable*. Such constraints are usually defined on disallowing certain moves which might lead to fallacious situations. Typical constraints to be found in $\mathbf{DC}$ are *non-circularity* (repeating the same argument twice in an argumentation line is forbidden), *commitment* (parties cannnot contradict themselves when advancing arguments), etc. It must be noted that a full formalization for dialectical constraints is outside the scope of this work. We do not claim to be able to identify every one of such constraints either, as they may vary from one particular argumentation framework to another; that is the reason why $\mathbf{DC}$ is included as a parameter in T. In this respect a similar approach is adopted in (Kakas & Toni 1999), where different characterizations of constraints give rise to different logic programming semantics.

Argumentation Lines

As already discussed before, argument games provide a useful form to characterize proof procedures for argumentation logics.[1] Such games model defeasible reasoning as a dispute between two parties (*Proponent* and *Opponent* of a claim), who exchange arguments and counterarguments, generating *dialogues*. A proposition Q is provably justified on the basis of a set of arguments if its proponent has a *winning strategy* for an argument supporting Q, i.e. every counterargument (defeater) advanced by the Opponent can be ultimately defeated by the Proponent. We believe that such argument game was first used in a computational setting in (Simari, Chesñevar, & García 1994a), and similar formalizations have been also applied in other argument-based approaches, e.g. in Prakken-Sartor's framework for argumentation based on logic programming (Prakken & Sartor 1997) and in Defeasible Logic Programming (DeLP) (García & Simari 2004) and its extensions, notably P-DeLP (Chesñevar *et al.* 2004). Dialogues in such argument games have been given different names (dialogue lines, argumentation lines, dispute lines, etc.). A discussion on such aspects of different logical models of argument can be found in (Chesñevar, Maguitman, & Loui 2000; Prakken & Vreeswijk 2002). In what follows we will borrow some basic terminology from (Chesñevar, Simari, & Godo 2005) for our formalization, which will provide the necessary elements for the intended analysis.

Definition 4 Let $T = (\Phi, \mathbf{DC})$ be an argumentation theory. An *argumentation line* λ in T is any finite sequence of arguments $[\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n]$ as defined before. We will say that λ is *rooted in* $\mathcal{A}_0$, and that the *length* of λ is $n+1$, writing $\mid \lambda \mid = s$ to denote that λ has s arguments. We will also write $\mathfrak{Lines}_\mathcal{A}$ to denote the set of all argumentation lines rooted in $\mathcal{A}$ in the theory T.

Definition 5 Let T be an argumentation theory and let $\lambda = [\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n]$ be an argumentation line in T. Then $\lambda' = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k]$, $k \leq n$, will be called an *initial argumentation segment* in λ of length k, denoted $\lfloor \lambda \rfloor_k$. When $k < n$ we will say that λ' is a proper initial argumentation segment in λ. We will use the term *initial segment* to refer to initial argumentation segments when no confusion arises.

Example 1 Consider a theory $T = (\Phi, \mathbf{DC})$, with $\mathbf{DC} = \emptyset$, where the set $\mathfrak{Args}$ is $\{\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$, and assume that the following relationships hold: $\mathcal{A}_1$ defeats $\mathcal{A}_0$, $\mathcal{A}_2$ defeats $\mathcal{A}_0$, $\mathcal{A}_3$ defeats $\mathcal{A}_0$, $\mathcal{A}_4$ defeats $\mathcal{A}_1$. Three different argumentation lines rooted in $\mathcal{A}_0$ can be obtained, namely:

$$\begin{aligned} \lambda_1 &= [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_4] \\ \lambda_2 &= [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2] \\ \lambda_3 &= [\mathcal{A}_0, \mathcal{A}_3] \end{aligned}$$

In particular, $\lfloor \lambda_1 \rfloor_2 = [\mathcal{A}_0, \mathcal{A}_1]$ is an initial argumentation segment in λ_1.

Example 2 Consider a theory $T' = (\Phi, \mathbf{DC})$ where the set $\mathfrak{Args}$ is $\{\mathcal{A}_0, \mathcal{A}_1\}$, and assume that the following relationships hold: $\mathcal{A}_0$ defeats $\mathcal{A}_1$, and $\mathcal{A}_1$ defeats $\mathcal{A}_0$. An infinite number of argumentation lines rooted in $\mathcal{A}_0$ can be obtained (e.g. $\lambda_1 = [\mathcal{A}_0]$, $\lambda_2 = [\mathcal{A}_0, \mathcal{A}_1]$, $\lambda_3 = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_0]$, $\lambda_4 = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_0, \mathcal{A}_1]$, etc.).

Remark 1 Note that from Def. 4, given an argumentation line $[\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$ every subsequence $[\mathcal{A}_i, \mathcal{A}_{i+1}, \ldots \mathcal{A}_{i+k}]$ with $0 \leq i$, $i+k \leq n$ is also an argumentation line. In particular, every initial argumentation segment is also an argumentation line.

[1] See an in-depth discussion in (Prakken 2005).

Intuitively, an argumentation line λ is acceptable iff it satisfies every dialectical constraint of the theory it belongs to. Formally:

Definition 6 Given an argumentation theory $T = (\Phi, \mathbf{DC})$, an argumentation line λ is *acceptable* wrt T iff $\mathbf{C}_i(\lambda) = True$, for every $\mathbf{C}_i \in \mathbf{DC}$.

In what follows, we will assume without loss of generality that the notion of acceptability imposed by dialectical constraints is such that if λ is acceptable wrt a theory $T = (\Phi, \mathbf{DC})$, then any subsequence of λ is also acceptable.

Assumption 1 If λ is an acceptable argumentation line wrt a theory $T = (\Phi, \mathbf{DC})$, then any subsequence of λ is also acceptable wrt T.

Example 3 Consider the theory T' in Ex. 2, and assume that $\mathbf{DC}$={ Repetition of arguments is not allowed }. Then λ_1 and λ_2 are acceptable argumentation lines in T', but λ_3 and λ_4 are not.

Definition 7 Let T be an argumentation theory, and let λ and λ' be two acceptable argumentation lines in T. We will say that λ' *extends* λ in T iff $\lambda = \lfloor \lambda' \rfloor_k$, for some $k < \mid \lambda' \mid$, that is, λ' extends λ iff λ is a proper initial argumentation segment of λ'.

Definition 8 Let T be an argumentation theory, and let λ be an acceptable argumentation line in T. We will say that λ is *exhaustive* if there is no acceptable argumentation line λ' in T such that $\mid \lambda \mid < \mid \lambda' \mid$, and for some k, $\lambda = \lfloor \lambda' \rfloor_k$, that is, there is no λ' such that extends λ. Non-exhaustive argumentation lines will be referred to as *partial* argumentation lines.

Example 4 Consider the theory T presented in Ex. 1. Then λ_1, λ_2 and λ_3 are exhaustive argumentation lines whereas $\lfloor \lambda_1 \rfloor_2$ is a partial argumentation line. In the case of the theory T' in Ex. 2, the argumentation line λ_2 extends λ_1. Argumentation line λ_2 is exhaustive, as it cannot be further extended on the basis of T' with the dialectical constraint introduced in Ex. 3.

We will distinguish the set $S = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ of argumentation lines rooted in the same initial argument and with the property of not containing lines that are initial subsequences of other lines in the set.

Definition 9 Given a theory T, a set $S = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of argumentation lines rooted in a given argument $\mathcal{A}$, denoted $S_\mathcal{A}$, is called a *bundle set* wrt T iff there is no pair $\lambda_i, \lambda_j \in S_\mathcal{A}$ such that λ_i extends λ_j.

Example 5 Consider the theory $T = (\Phi, \mathbf{DC})$ from Ex. 1, and the argumentation lines λ_1, λ_2, and λ_3. Then $S_{\mathcal{A}_0} = \{\lambda_1, \lambda_2, \lambda_3\}$ is a bundle set of argumentation lines wrt T.

As we will see next, a bundle set of argumentation lines rooted in a given argument $\mathcal{A}$ provides the basis for conceptualizing a tree structure called *dialectical tree*.

Dialectical Trees

A bundle set $S_\mathcal{A}$ consists of argumentation lines rooted in a given argument $\mathcal{A}$ which can be "put" together in a tree structure. Formally:

Definition 10 Let T be a theory, and let $\mathcal{A}$ be an argument in T, and let $S_\mathcal{A} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a bundle set of argumentation lines rooted in $\mathcal{A}$. Then, the *dialectical tree* rooted in $\mathcal{A}$ *based on* $S_\mathcal{A}$, denoted $\mathcal{T}_\mathcal{A}$, is a tree structure defined as follows:

1. The root node of $\mathcal{T}_\mathcal{A}$ is $\mathcal{A}$.
2. Let F={$\mathsf{tail}(\lambda)$, for every $\lambda \in S_\mathcal{A}$}, and H={$\mathsf{head}(\lambda)$, for every $\lambda \in F$}.[2]
   If $H = \emptyset$ then $\mathcal{T}_\mathcal{A}$ has no subtrees.
   Otherwise, if $H = \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$, then for every $\mathcal{B}_i \in H$, we define

   $$\textit{getBundle}(\mathcal{B}_i) = \{\lambda \in F \mid \mathsf{head}(\lambda) = \mathcal{B}_i\}$$

   We put $\mathcal{T}_{\mathcal{B}_i}$ as an immediate subtree of $\mathcal{A}$, where $\mathcal{T}_{\mathcal{B}_i}$ is a dialectical tree based on $\textit{getBundle}(\mathcal{B}_i)$.

We will write $\mathfrak{Tree}_\mathcal{A}$ to denote the family of all possible dialectical trees based on $\mathcal{A}$. We will represent as $\mathfrak{Tree}_T$ the family of all possible dialectical trees in the theory T.

Example 6 Consider the theory $T = (\Phi, \mathbf{DC})$ from Ex. 1. In that theory it holds that $S_{\mathcal{A}_0} = \{\lambda_1, \lambda_2, \lambda_3\}$ is a bundle set. Fig. 1(a) shows an associated dialectical tree $\mathcal{T}_{\mathcal{A}_0}$.

The above definition shows how to build a dialectical tree from a bundle set of argumentation lines rooted in a given argument. It is important to note that the "shape" of the resulting tree will depend on the order in which the subtrees are attached. Each possible order will produce a tree with a different geometric configuration. All the differently conformed trees are nevertheless "equivalent" in the sense that they will contain exactly the same argumentation lines as branches from its root to its leaves. This observation is formalized by introducing the following relation which can be trivially shown to be an equivalence relation.

Definition 11 Let T be a theory, and let $\mathfrak{Tree}_\mathcal{A}$ be the set of all possible dialectical trees rooted in an argument $\mathcal{A}$ in theory T. We will say that $\mathcal{T}_\mathcal{A}$ is equivalent to $\mathcal{T}'_\mathcal{A}$, denoted $\mathcal{T}_\mathcal{A} \equiv_\tau \mathcal{T}'_\mathcal{A}$ iff they are obtained from the the same bundle set $S_\mathcal{A}$ of argumentation lines rooted in $\mathcal{A}$.

Given an argument $\mathcal{A}$, there is a one-to-one correspondence between a bundle set $S_\mathcal{A}$ of argumentation lines rooted in $\mathcal{A}$ and the corresponding equivalence class of dialectical trees that share the same bundle set as their origin (as specified in Def. 10). In fact, a dialectical tree $\mathcal{T}_\mathcal{A}$ based

[2] The functions $\mathsf{head}(\cdot)$ and $\mathsf{tail}(\cdot)$ have the usual meaning in list processing.

on $S_{\mathcal{A}}$ is just *an alternative way* of expressing the same information already present in $S_{\mathcal{A}}$. Each member of an equivalence class represents a different way in which a tree could be built. Each particular computational method used to generate the tree from the bundle set will produce one particular member on the equivalence class. In that manner, the equivalence relation will represent a tool for exploring the computational process of warrant and as we will see later, trees provide a powerful way of conceptualize the computation of warranted arguments. Next, we will define mappings which allow to re-formulate a bundle set $S_{\mathcal{A}}$ as a dialectical tree $\mathcal{T}_{\mathcal{A}}$ and viceversa.

Definition 12 Let T be an argumentative theory, and let $S_{\mathcal{A}}$ be a bundle set of argumentation lines rooted in an argument $\mathcal{A}$ of T. We define the mapping

$$\mathbb{T} : \wp(\mathfrak{Lines}_{\mathcal{A}}) \setminus \{\emptyset\} \mapsto \overline{\mathfrak{Tree}_{\mathcal{A}}}$$

as $\mathbb{T}(S_{\mathcal{A}}) =_{def} \overline{\mathcal{T}_{\mathcal{A}}}$, where $\overline{\mathfrak{Tree}_{\mathcal{A}}}$ is the quotient set of $\mathfrak{Tree}_{\mathcal{A}}$ by $\equiv_{\tau}$, and $\overline{\mathcal{T}_{\mathcal{A}}}$ denotes the equivalence class of $\mathcal{T}_{\mathcal{A}}$.

Proposition 1 For any argument $\mathcal{A}$ in an argumentative theory T, the mapping $\mathbb{T}$ is a bijection.[3]

As the mapping $\mathbb{T}$ is a bijection, we can also define the inverse mapping $\mathbb{S} =_{def} \mathbb{T}^{-1}$ which allow us to determine the associated bundle set of argumentation lines corresponding to an arbitrary class of dialectical trees rooted in an argument $\mathcal{A}$.

In what follows, we will use indistinctly a *set notation* (a bundle set of argumentation lines rooted in an argument $\mathcal{A}$) or a *tree notation* (a dialectical tree rooted in $\mathcal{A}$), as the former mappings $\mathbb{S}$ and $\mathbb{T}$ allow us to go from any of these notation to the other.

The following proposition shows that dialectical trees can be thought of as structures in which any subtree $\mathcal{T}'_{\mathcal{A}}$ of a dialectical tree $\mathcal{T}_{\mathcal{A}}$ is also a dialectical tree.

Proposition 2 Let T be a theory, and $\mathcal{T}_{\mathcal{A}}$ a dialectical tree in T. Then it holds that any subtree $\mathcal{T}'_{\mathcal{A}}$ of $\mathcal{T}_{\mathcal{A}}$, rooted in $\mathcal{A}$, is also a dialectical tree wrt T.

Acceptable dialectical trees

The notion of acceptable argumentation line will be used to characterize acceptable dialectical trees, which will be fundamental as a basis for formalizing the computation of warrant in our setting.

Definition 13 Let T be a theory, a dialectical tree $\mathcal{T}_{\mathcal{A}}$ in T is acceptable iff every argumentation line in the associated bundle set $\mathbb{S}(\overline{\mathcal{T}_{\mathcal{A}}})$ is acceptable. We will distinguish the subset $\mathfrak{AT}\mathfrak{ree}_{\mathcal{A}}$ (resp. $\mathfrak{AT}\mathfrak{ree}_{T}$) of all acceptable dialectical trees in $\mathfrak{Tree}_{\mathcal{A}}$ (resp. $\mathfrak{Tree}_{T}$).

As acceptable dialectical trees are a subclass of dialectical trees, all the properties previously shown apply also to them. In the sequel, we will just write "dialectical trees" to refer to acceptable dialectical trees, unless stated otherwise.

[3]Proofs not included for space reasons.

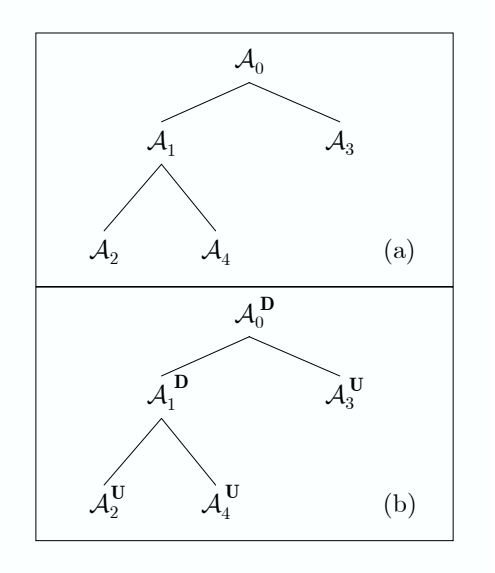


Figure 1: (a) Dialectical tree and (b) marked dialectical tree for Example 6

Definition 14 A dialectical tree $\mathcal{T}_{\mathcal{A}}$ will be called *exhaustive* iff it is constructed from the set $S_{\mathcal{A}}$ of all possible exhaustive argumentation lines rooted in $\mathcal{A}$, otherwise $\mathcal{T}_{\mathcal{A}}$ will be called *partial*.

Besides, the exhaustive dialectical tree for any argument $\mathcal{A}$ can be proven to be unique.

Proposition 3 Let T be a theory, and let $\mathcal{A}$ be an argument in T. Then there exists a unique exhaustive dialectical tree $\mathcal{T}_{\mathcal{A}}$ in T (up to an equivalence wrt $\equiv_{\tau}$ as defined in Def. 11)

Acceptable dialectical trees allow to determine whether the root node of the tree is to be accepted (ultimately *undefeated*) or rejected (ultimately *defeated*) as a rationally justified belief. A *marking function* provides a definition of such acceptance criterion. Formally:

Definition 15 Let T be a theory. A marking criterion for T is a function $Mark : \mathfrak{Tree}_{T} \rightarrow \{D, U\}$. We will write $Mark(\mathcal{T}_i) = U$ (resp. $Mark(\mathcal{T}_i) = D$) to denote that the root node of $\mathcal{T}_i$ is marked as U-node (resp. D-node).

Several marking criteria can be defined for capturing skeptical semantics for argumentation. A particular criterion (which we will later use in our analysis for strategies for computing warrant) is the AND-OR marking of a dialectical tree (Simari, Chesñevar, & García 1994a), which corresponds to Dung's grounded semantics (Dung 1995).

Definition 16 Let T be a theory, and let $\mathcal{T}_{\mathcal{A}}$ be a dialectical tree. The and-or marking of $\mathcal{T}_{\mathcal{A}}$ is defined as follows:

1. If $\mathcal{T}_{\mathcal{A}}$ has no subtrees, then $Mark(\mathcal{T}_{\mathcal{A}}) = U$.
2. If $\mathcal{T}_{\mathcal{A}}$ has subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ then

(a) *Mark*$(\mathcal{T}_\mathcal{A}) = U$ iff *Mark*$(\mathcal{T}_i) = D$, for all $i = 1 \ldots k$.
(b) *Mark*$(\mathcal{T}_\mathcal{A}) = D$ iff there exists $\mathcal{T}_i$ such that *Mark*$(\mathcal{T}_i) = U$, for some $i = 1 \ldots k$.

Proposition 4 Let T be a theory, and let $\mathcal{T}_\mathcal{A}$ be a dialectical tree. The and-or marking defined in Def. 16 assigns the same mark to all the members of $\overline{\mathcal{T}_\mathcal{A}}$.

Remark 2 As a design criterion, it would be sensible to require that a particular marking criterion would respect that every member of a given equivalence class will be marked in the same way. This will provide an "invariance" of marking with respect to the particular way the algorithm introduced in Def. 10 builds the tree. In such manner, that invariance will allow to work with the bundle set disregarding the circumstantial element of the equivalence class at hand. Each marking procedure is affected by the geometric properties of the tree. For instance, the classical and-or tree traversal will work best with trees that have their shortest branches to the left (Simari, Chesñevar, & García 1994a; Simari, Chesñevar, & García 1994b; Chesñevar, Simari, & Godo 2005), but other procedures could work better on different configurations. Working with the bundle set, and transforming it in a *bundle list* by using some preprocessing algorithm could result in significant speed-ups. Pursuing these observations is outside the scope and length restrictions of this paper, but has been addressed elsewhere (Chesñevar & Simari 2005).

Definition 17 Let T be an argumentative theory and *Mark* a marking criterion for T. An argument $\mathcal{A}$ is a *warranted argument* (or just *warrant*) in T iff the exhaustive dialectical tree $\mathcal{T}_\mathcal{A}$ is such that *Mark*$(\mathcal{T}_\mathcal{A}) = U$.

Example 7 Consider the exhaustive dialectical tree $\mathcal{T}_{\mathcal{A}_0}$ in Ex. 6 shown in Fig. 1(a). Fig. 1(b) shows the corresponding marking by applying Def. 16, showing that $\mathcal{A}_0$ –the root of $\mathcal{T}_{\mathcal{A}_0}$– is an ultimately defeated argument, i.e. $Mark(\mathcal{T}_{\mathcal{A}_0}) = D$. Hence $\mathcal{A}_0$ is not a warranted argument. In Fig. 2 the and-or marking from deep-first, left to right, in (a) will have to traverse the whole tree, meanwhile in (b) only visits two modes. Both trees belong to same equivalent class.

Warrant Computation via Dialectical Trees

As stated in the introduction, our main concern is to model warrant computation in skeptical argumentation frameworks. Fix-point definitions are very expressive declaratively, but tree structures lend themselves naturally to implementation. In fact, some implementations of skeptical argumentation systems (e.g. DeLP (García & Simari 2004)) rely on tree structures (such as dialectical trees) which can be computed by performing backward chaining at two levels. On the one hand, arguments are computed by backward chaining from a query (goal) using a logic programming approach (e.g. SLD resolution). On the other hand, dialectical trees can be computed by recursively analyzing defeaters for a given argument, defeaters for those defeaters, and so on. In particular, in more complex and general settings (such as admissibility semantics) dialectical proof procedures have been developed (Dung, Kowalski, & Toni 2006) using a similar strategy to compute warranted belief.

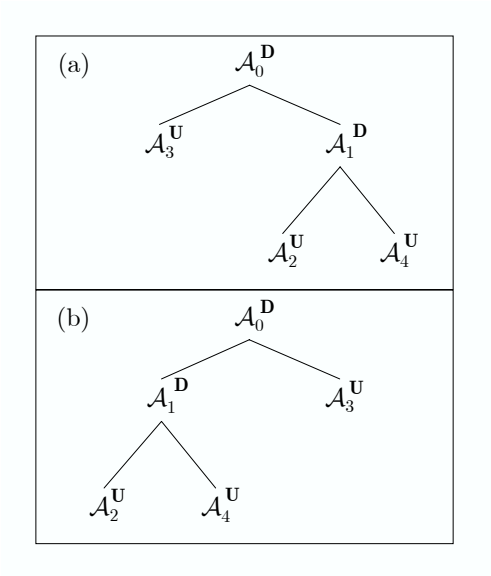


Figure 2: (a) Dialectical tree and (b) Symmetric dialectical tree for Example 6

In our abstract model we will use dialectical trees to formalize warrant computation. As indicated in (Chesñevar, Simari, & Godo 2005), the process of building an arbitrary dialectical tree $\mathcal{T}_{\mathcal{A}_0}$ can be thought of as a *computation* starting from an initial tree (consisting of a single node) and evolving into more complex trees by adding new arguments (nodes) stepwise. Elementary steps in this computation can be related by means of a precedence relationship "$\sqsubseteq$" among trees:

Definition 18 Let T be a theory, $\mathcal{A}$ an argument and let $\mathcal{T}_\mathcal{A}$, $\mathcal{T}'_\mathcal{A}$ be acceptable dialectical trees rooted in $\mathcal{A}$. We define a relationship $\sqsubset \subseteq \mathfrak{Tree}_\mathcal{A} \times \mathfrak{Tree}_\mathcal{A}$. We will write $\mathcal{T}_\mathcal{A} \sqsubset \mathcal{T}'_\mathcal{A}$ whenever $\mathcal{T}'_\mathcal{A}$ can be obtained from $\mathcal{T}_\mathcal{A}$ by extending some argumentation line λ in $\mathcal{T}_\mathcal{A}$ by exactly one argument. As usual, we will write $\mathcal{T}_\mathcal{A} \sqsubseteq \mathcal{T}'_\mathcal{A}$ iff $\mathcal{T}_\mathcal{A} = \mathcal{T}'_\mathcal{A}$ or $\mathcal{T}_\mathcal{A} \sqsubset \mathcal{T}'_\mathcal{A}$. We will also write $\mathcal{T}_\mathcal{A} \sqsubseteq_* \mathcal{T}'_\mathcal{A}$ iff there exists a (possibly empty) sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ such that $\mathcal{T}_\mathcal{A} = \mathcal{T}_1 \sqsubseteq \ldots \sqsubseteq \mathcal{T}_k = \mathcal{T}'_\mathcal{A}$.

From Def. 18 the notion of exhaustive dialectical tree can be recast as follows: A dialectical tree $\mathcal{T}_i$ is exhaustive iff there is no $\mathcal{T}_j \neq \mathcal{T}_i$ such that $\mathcal{T}_i \sqsubset \mathcal{T}_j$. Every dialectical tree $\mathcal{T}_i$ can be seen as a 'snapshot' of the status of a disputation between two parties (proponent and opponent), and the relationship "$\sqsubseteq$" allows to capture the evolution of such disputation.[4] In particular, note that for any argumentative theory

[4] Note however that $\mathcal{T}_i \sqsubseteq \mathcal{T}_j$ does not imply that one party has advanced some argument in $\mathcal{T}_i$ and the other party has replied in $\mathcal{T}_j$. Thus our framework provides a setup to define *unique*- and

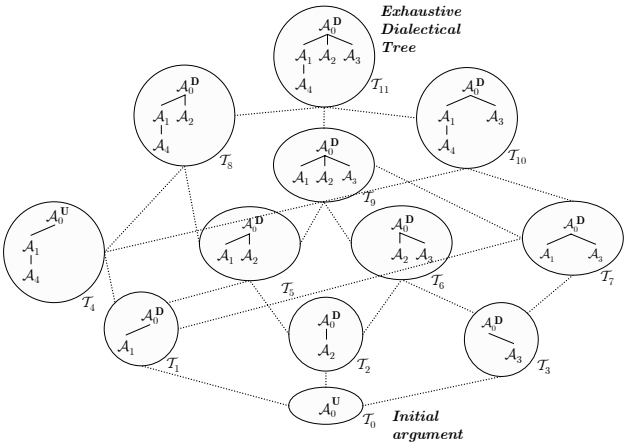


Figure 3: Lattice for all possible dialectical trees rooted in an argument $\mathcal{A}_0$ (Example 8)

T, given an argument $\mathcal{A}$ the ordered set $(\mathfrak{Tree}_{\mathcal{A}}, \sqsubseteq_*)$ is a poset, where the least element is $\mathcal{A}$ and the greatest element is the exhaustive dialectical tree $\mathcal{T}_{\mathcal{A}}$.

We are now concerned with the following question: *can we enumerate all possible ways of computing the exhaustive dialectical tree $\mathcal{T}_{\mathcal{A}}$ rooted in a given initial argument $\mathcal{A}$?* The answer is yes. In fact, as we will see in the next definitions, we can provide a lattice characterization for the space of all possible dialectical trees rooted in a given argument $\mathcal{A}$. In order to characterize a lattice for dialectical trees we will provide two operations:

- *Join* of dialectical trees ($\vee$), which given two dialectical trees $\mathcal{T}_1$ and $\mathcal{T}_2$ will compute the "union" of $\mathcal{T}_1$ and $\mathcal{T}_2$, in the sense that it will contain all defeaters present either in $\mathcal{T}_1$ *or* in $\mathcal{T}_2$.
- *Meet* of dialectical trees ($\wedge$), which given two dialectical trees $\mathcal{T}_1$ and $\mathcal{T}_2$ will compute the "intersection" of $\mathcal{T}_1$ and $\mathcal{T}_2$, in the sense that it will contain all defeaters present only in $\mathcal{T}_1$ *and* in $\mathcal{T}_2$.

Definition 19 Let T be an argumentative theory, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be dialectical trees rooted in $\mathcal{A}$. We define the *meet* and *join* of $\mathcal{T}_1$ and $\mathcal{T}_2$, (written $\mathcal{T}_1 \wedge \mathcal{T}_2$ and $\mathcal{T}_1 \vee \mathcal{T}_2$) as follows:

- λ is an argumentation line in $\mathcal{T}_1 \vee \mathcal{T}_2$ iff
  1. $\lambda \in \mathcal{T}_1$ and there is no $\lambda' \in \mathcal{T}_2$ such that $\lambda' \text{ extends } \lambda$, or
  2. $\lambda \in \mathcal{T}_2$ and there is no $\lambda' \in \mathcal{T}_1$ such that $\lambda' \text{ extends } \lambda$
- λ is an argumentation line in $\mathcal{T}_1 \wedge \mathcal{T}_2$ iff $\lambda = \lfloor \lambda_1 \rfloor_k = \lfloor \lambda_2 \rfloor_k$, for some $k > 0$ such that $\lambda_1 \in \mathcal{T}_1$ and $\lambda_2 \in \mathcal{T}_2$ and there is no λ' that extends λ satisfying this situation.

The next two results follow naturally from the previous definition.

Proposition 5 The operations $\wedge$ and $\vee$ are well-defined, i.e. for any dialectical trees $\mathcal{T}_1$ and $\mathcal{T}_2$ rooted in a given argument $\mathcal{A}$, $\mathcal{T}_1 \wedge \mathcal{T}_2$ and $\mathcal{T}_1 \vee \mathcal{T}_2$ are also dialectical trees rooted in $\mathcal{A}$.

Proposition 6 Let T be an argumentation theory, and λ an acceptable argumentation line in T. Then it holds that

1. $\lambda \in \mathcal{T}_1 \vee \mathcal{T}_2$ iff $\lambda \in \mathcal{T}_1$ or $\lambda \in \mathcal{T}_2$
2. $\lambda \in \mathcal{T}_1 \wedge \mathcal{T}_2$ iff $\lambda \in \mathcal{T}_1$ and $\lambda \in \mathcal{T}_2$
3. $\lambda \notin \mathcal{T}_1 \wedge \mathcal{T}_2$ iff $\lambda \notin \mathcal{T}_1$ or $\lambda \notin \mathcal{T}_2$

The next lemma shows that for any argumentation theory T the set of all possible acceptable dialectical trees rooted in a particular argument can be conceptualized as a lattice.

Lemma 1 Let $\mathcal{A}$ be an argument in a theory T, and let $(\mathfrak{ATree}_{\mathcal{A}}, \sqsubseteq_*)$ be the associated poset. Then $(\mathfrak{ATree}_{\mathcal{A}}, \vee, \wedge)$ is a lattice.

Given the lattice $(\mathfrak{ATree}_{\mathcal{A}}, \vee, \wedge)$, we will write $\mathcal{T}_{\mathcal{A}}^{\perp}$ to denote the bottom element of the lattice (i.e., the dialectical tree involving only $\mathcal{A}$ as root node) and $\mathcal{T}_{\mathcal{A}}^{\top}$ to denote the top element of the lattice (i.e., the exhaustive dialectical tree).

Example 8 Consider the theory T from Ex. 1, and the exhaustive dialectical tree rooted in $\mathcal{A}_0$ shown in Ex. 6. The complete lattice associated with $\mathcal{A}_0$ is shown in Fig. 3.

Computing Warrant Efficiently

In the preceding Section we have shown that given an argumentative theory T, for any argument $\mathcal{A}$ in T there is a lattice $(\mathfrak{ATree}_{\mathcal{A}}, \vee, \wedge)$ whose bottom element is a dialectical tree with a single node (the argument $\mathcal{A}$ itself) and whose top element is the exhaustive dialectical tree $\mathcal{T}_{\mathcal{A}}$. In that lattice, whenever $\mathcal{T}_k = \mathcal{T}_i \vee \mathcal{T}_j$ it is the case that $\mathcal{T}_i \sqsubseteq \mathcal{T}_k$ and $\mathcal{T}_j \sqsubseteq \mathcal{T}_k$.

In Fig. 3 corresponding to Example 8 we can see that for dialectical trees $\mathcal{T}_2$ and $\mathcal{T}_3$, it holds that *Mark*($\mathcal{T}_2$) = *Mark*($\mathcal{T}_3$) = D (assuming that *Mark* is defined as in Def. 16). Clearly, it is the case that any tree $\mathcal{T}_i$ where $\mathcal{T}_2 \sqsubseteq \mathcal{T}_i$ or $\mathcal{T}_3 \sqsubseteq \mathcal{T}_i$ satisfies that *Mark*($\mathcal{T}_i$) = D. In other words, whichever is the way the tree $\mathcal{T}_2$ (or $\mathcal{T}_3$) evolves into a new tree in $(\mathfrak{ATree}_{\mathcal{A}_0}, \vee, \wedge)$ it turns out that the associated marking remains unchanged. We formalize that situation as follows:

Definition 20 Let T be an argumentation theory, and let $\mathcal{T}_{\mathcal{A}}$ be a dialectical tree, such that for every $\mathcal{T}'_{\mathcal{A}}$ evolving from $\mathcal{T}_{\mathcal{A}}$ (*i.e.*, $\mathcal{T}_{\mathcal{A}} \sqsubseteq_* \mathcal{T}'_{\mathcal{A}}$) it holds that *Mark*($\mathcal{T}_{\mathcal{A}}$) = *Mark*($\mathcal{T}'_{\mathcal{A}}$). Then $\mathcal{T}_{\mathcal{A}}$ is a *settled dialectical tree* in T.

Now we have a natural, alternative way of characterizing warrant.

Proposition 7 Let T be a theory, and let $\mathcal{A}$ be an argument in T. Then $\mathcal{A}$ is a warrant wrt T iff *Mark*($\mathcal{T}_{\mathcal{A}}$) = U, where $\mathcal{T}_{\mathcal{A}}$ is a settled dialectical tree.

Clearly, computing settled dialectical trees is less expensive than computing exhaustive dialectical trees, as fewer nodes (arguments) are involved in the former case. Following Hunter's approach (Hunter 2004), in what follows we

multi-move protocols as defined by Prakken (Prakken 2005).

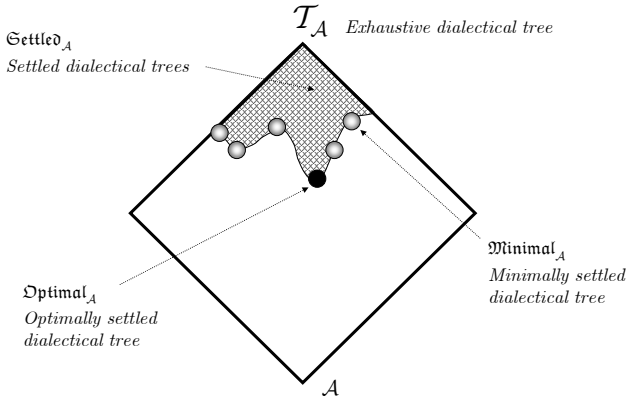


Figure 4: Search space for computing dialectical trees rooted in $\mathcal{A}$

will formalize the *cost* of computing a dialectical tree as a function $\mathsf{cost} : \mathfrak{Tree}_T \rightarrow \Re$. As explained in (Hunter 2004), several issues can be considered when computing such cost.

The next definition refines the class of settled dialectical trees by distinguishing those trees involving *as few arguments as possible* in order to determine whether the root of the tree is ultimately a warranted argument according to the marking procedure. From the many possible minimally settled dialectical trees rooted in a given argument $\mathcal{A}$, a dialectical tree $\mathcal{T}$ is *optimally settled* if there is no $\mathcal{T}'$ that is less expensive than $\mathcal{T}$.

Definition 21 A dialectical tree $\mathcal{T}$ is a *minimally settled dialectical tree* iff there is no $\mathcal{T}' \sqsubset \mathcal{T}$ such that $\mathcal{T}'$ is a settled dialectical tree. A dialectical tree $\mathcal{T}$ is an *optimally settled dialectical tree* iff $\mathcal{T}$ is minimally settled, and for any other settled tree $\mathcal{T}'$, $\mathsf{cost}(\mathcal{T}) \leq \mathsf{cost}(\mathcal{T}')$.

Example 9 Consider the theory T from Ex. 1, and the complete lattice $(\mathfrak{ATree}_{\mathcal{A}_0}, \vee, \wedge)$ shown in Fig. 3. Then $\mathcal{T}_2$ and $\mathcal{T}_3$ are minimally settled dialectical trees.

Let $\mathfrak{Settled}_{\mathcal{A}}$, $\mathfrak{Minimal}_{\mathcal{A}}$ and $\mathfrak{Optimal}_{\mathcal{A}}$ be the sets of all settled, minimally settled and optimally settled dialectical trees for an argument $\mathcal{A}$, resp. Clearly, it holds that

$$\mathfrak{Optimal}_{\mathcal{A}} \subseteq \mathfrak{Minimal}_{\mathcal{A}} \subseteq \mathfrak{Settled}_{\mathcal{A}} \subseteq \mathfrak{ATree}_{\mathcal{A}}.$$

The sets $\mathfrak{Settled}_{\mathcal{A}}$, $\mathfrak{Minimal}_{\mathcal{A}}$ and $\mathfrak{Optimal}_{\mathcal{A}}$ can be identified in any lattice $(\mathfrak{ATree}_{\mathcal{A}}, \vee, \wedge)$, as shown in Figure 4. The borderline on top of the lattice denotes all possible minimally settled dialectical trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ rooted in $\mathcal{A}$. Some of such trees in that set may be optimal. Any dialectical tree that evolves from settled dialectical trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ will be also a settled dialectical tree. In particular, the exhaustive dialectical tree is also settled.

Dialectical Constraints (Revisited)

As we have analyzed in the previous Section, the lattice associated with any argument $\mathcal{A}$ accounts for the whole search space for detecting if $\mathcal{A}$ is warranted. To do so it is not necessary to compute the exhaustive dialectical tree rooted in $\mathcal{A}$; rather, it suffices to focus search on settled dialectical trees, as they involve less nodes and are consequently more efficient.

When determining whether a conclusion is warranted, argument-based inference engines are supposed to compute a sequence of dialectical trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ such that $\mathcal{T}_k$ is a settled dialectical tree. For skeptical argumentation semantics, argument-based engines like DeLP (García & Simari 2004; Chesñevar *et al.* 2003; Simari, Chesñevar, & García 1994a) use *depth-first search* to generate dialectical trees for queries and determine if a given literal is warranted. Such search can be improved by applying $\alpha - \beta$ pruning, so that not every node (argument) is computed. In other words, depth-first search favors naturally the computation of settled dialectical trees.

The natural question that arises next is how to compute *minimally settled trees*. Given a theory $T = (\Phi, \mathbf{DC})$, it turns out that the set of dialectical constraints $\mathbf{DC}$ can help to provide a way of approximating such minimally settled trees, based on the fact that in depth-first search the *order* in which branches are generated is important: should shorter branches be computed before longer ones, then the resulting search space can be proven to be smaller on an average search tree (Chesñevar, Simari, & Godo 2005). Usually heuristics are required to anticipate which branches are likely to be shorter than the average.

Constraints in $\mathbf{DC}$ can help provide such kind of heuristics. Thus, for example, in Defeasible Logic Programming (García & Simari 2004; Chesñevar *et al.* 2003) and Possibilistic Defeasible Logic Programming (Chesñevar *et al.* 2004) the set $\mathbf{DC}$ includes as a constraint that *arguments advanced by the proponent (resp. opponent) should not be contradictory* in any argumentation line. The following heuristics (Chesñevar, Simari, & Godo 2005) can be shown to favor the computation of shorter argumentation lines when applying depth-first search in the context of Possibilistic Defeasible Logic Programming: *if the current argument $\mathcal{A}_0$ is a leaf node in a dialectical tree $\mathcal{T}$, and has different candidate defeaters $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$, then the $\mathcal{A}_i$ which shares as many literals as possible with $\mathcal{A}_0$ should be chosen when performing the depth-first computation of $\mathcal{T}_{\mathcal{A}_0}$.*

By applying the above heuristics it can be shown that the branching factor for arguments below $\mathcal{A}_0$ is reduced. In other words, depth-first computation of dialectical trees favors naturally the construction of minimally settled dialectical trees, whereas by applying the above heuristics an approximation to optimally settled dialectical trees is obtained.

Relevance in Dialectical Trees

In (Prakken 2000) the notion of *relevance* was introduced in the context of argument games and the characterization of protocols for liberal disputes. According to (Prakken 2000), a move is relevant in a dispute D iff it changes the disputa-

tional status of D's initial move.[5] In our context, dialectical trees correspond to such disputes. In the setting presented in (Prakken 2000), moves are performed by both parties involved in a dispute (Proponent and Opponent).

Interestingly, there is a clear relation between minimally settled dialectical trees and this notion of relevance, as the notion of extending an argumentation line by one argument (as introduced in Def. 18) can be recast as performing a move.

Definition 22 Let $T = (\Phi, \mathbf{DC})$ be an argumentation theory, and let $\mathcal{T}_{\mathcal{A}_1}$, $\mathcal{T}'_{\mathcal{A}_1}$ be acceptable dialectical trees. We will say that there is a *move* M from $\mathcal{T}_{\mathcal{A}}$ to $\mathcal{T}'_{\mathcal{A}}$, denoted as *Move*$(\mathcal{T}_{\mathcal{A}}, \mathcal{T}'_{\mathcal{A}})$, iff $\mathcal{T}_{\mathcal{A}} \sqsubset \mathcal{T}'_{\mathcal{A}}$.

It must be remarked that a proper conceptualization of move in argumentation demands more parameters, such as identifying the argumentation line in which a argument is introduced, who is the player (Proponent or Opponent) making the move, etc. Such an approach has been formalized by (Prakken 2000; 2005). Our approach in this case is intentionally over-simplified, as it just aims to relate the notion of relevance and the notion of minimally settled dialectical trees. In fact, note that Def. 22 allows us to formalize the computation of an acceptable dialectical tree $\mathcal{T}_k$ rooted in $\mathcal{A}_0$ as a sequence of moves *Move*$(\mathcal{T}_0, \mathcal{T}_1)$, *Move*$(\mathcal{T}_1, \mathcal{T}_2)$, ..., *Move*$(\mathcal{T}_{k-1}, \mathcal{T}_k)$, where $\mathcal{T}_0$ is a dialectical tree with a single node $\mathcal{T}^{\perp}_{\mathcal{A}_0}$. Following Prakken's notion of relevance, we can express this concept in our setting as follows:

Definition 23 A move M = *Move*$(\mathcal{T}_{\mathcal{A}}, \mathcal{T}'_{\mathcal{A}})$ is *relevant* iff *Mark*$(\mathcal{T}_{\mathcal{A}}) \neq$ *Mark*$(\mathcal{T}'_{\mathcal{A}})$.

The following proposition shows that minimally settled trees are only those obtained by performing a sequence of relevant moves ending in a settled dialectical tree.

Proposition 8 Let T be an argumentation theory, and let $\mathcal{T}_{\mathcal{A}}$ be a dialectical tree. Then $\mathcal{T}_{\mathcal{A}}$ is minimally settled iff there is a sequence of moves $M_1, M_2, \ldots, M_k$ such that every move M_i is relevant, and M_k results in a settled dialectical tree.

Related Work

Dialectical constraints have motivated research in argumentation theory in different directions. As stated before, the main role of such constraints is to avoid *fallacious* reasoning (Aristotle ; Hamblin 1970; Rescher 1977; Walton 1995). In our proposal dialectical constraint are left as a particular parameter to be included in the argumentation theory. It must be remarked that different formalizations of argument-based dialectical proof procedures have included particular dialectical constraints as part of their specification. In (Simari, Chesñevar, & García 1994a; Simari, Chesñevar, & García 1994b), an approach to model different dialectical constraints was presented. These constraints were applied as part of the procedure used for constructing dialectical trees by discarding "ill-formed" argumentation lines. In (Besnard & Hunter 2001) the authors present a logic of argumentation which disallows repetition of arguments in argument trees (Besnard & Hunter 2001, p.215):

> *For no node* (ϕ, β) *with ancestor nodes* (ϕ_1, β_1), $(\phi_2, \beta_2), \ldots, (\phi_k, \beta_k)$ *is* ϕ *a subset of* $\phi_1 \cup \phi_2 \cup \ldots \cup \phi_k$.

In a similar manner, other approaches (like (Kakas & Toni 1999)) compute different semantics for logic programming on the basis of an argumentative approach formalized in terms of trees. Some properties can be used to render the construction of such trees more efficient. Thus, in the case of computing well-founded semantics via trees, defense nodes (which account for Proponent's argument in an argumentation line) cannot attack any other defense node in the tree. Similarly, in (Dung, Kowalski, & Toni 2006) the notion of dispute tree is used to compute assumption-based, admissible argumentation. As the authors indicate, in order for an abstract dispute tree to be *admissible*, there is a further requirement that "the proponent does not attack itself". Such kind of restrictions can be seen as particular dialectical constraint in the context of our proposal.

Recently there have been other research oriented towards formalizing dialectical proof procedures for argumentation. To the best of our knowledge, none of such works formalizes the dialectical search space through a lattice as presented in this paper. Our work complements previous research concerning the dynamics of argumentation, notably (Prakken 2001) and (Brewka 2001). In particular, Prakken (Prakken 2001) has analyzed the exchange of arguments in the context of dynamic disputes. Our approach can also be understood in the light of his characterization of dialectical proof theories (Prakken 2005). However, although Prakken develops a very comprehensive general framework, in our understanding some important computational issues (e.g. search space considerations) are not taken into account. Hunter (Hunter 2004) analyzes the search space associated with dialectical trees taking into account novel features such as the *resonance* of arguments. His interesting formalization combines a number of features that allow to assess the impact of dialectical trees, contrasting shallow vs. deep trees. However, search space considerations as modeled in this paper are outside the scope of his approach. In (Kakas & Toni 1999) a throughout analysis of various argumentation semantics for logic programming is presented on the basis of parametric variations of derivation trees. In contrast with that approach, our aim in this paper was not to characterize different emerging semantics, but rather to focus on the role of dialectical trees as a way of modeling the search space when computing warrants. Besides, in (Kakas & Toni 1999) the authors concentrate in normal logic programming, whereas our approach is more generic.

Conclusions. Future Work

In this paper we have presented a novel approach to model the search space associated with warrant computation in an

[5] The notion of relevance as well as some interesting properties were further studied and refined (Prakken 2005).

abstract argumentation framework. We have shown how the notion of dialectical tree can be used constructively to model different stages in the process of computing warranted arguments. We have shown how the process of computing warrant can be recast into computing dialectical trees within a lattice, illustrating how dialectical constraints can play a role for guiding an efficient computation of warranted literals.

Part of our future work is related to studying theoretical properties of the proposed framework, analyzing their incidence for developing efficient argument-based inference engines. In this context we think that the notion of equivalence classes associated with dialectical trees can be specially useful as discussed in Remark 2. Research in this direction is currently being pursued.

Acknowledgements We thank anonymous reviewers for comments which helped to improve the final version of this paper. This research was partially supported by Projects TIC2003-00950, TIN2004-07933-C03-01/03, by Ramón y Cajal Program (Ministerio de Ciencia y Tecnología, Spain), by CONICET (Argentina), by the Secretaría General de Ciencia y Tecnología de la Universidad Nacional del Sur and by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13096).

References

Aristotle. *On Sophistical Refutations*. eBooks@Adelaide, Translated by W. A. Pickard-Cambridge, web edition at `http://etext.library.adelaide.edu.au/a/aristotle/sophistical/`, The University of Adelaide Library, 2004 edition.

Besnard, P., and Hunter, A. 2001. A logic-based theory of deductive arguments. *Artificial Intelligence* 1:2(128):203–235.

Brewka, G. 2001. Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *J. of Logic and Computation* 11(2):257–282.

Chesñevar, C., and Simari, G. 2005. Computing warrant in an abstract model for skeptical argumentation frameworks: formalization and properties. In *Technical Report 343 – University of LLeida & LIDIA*.

Chesñevar, C.; Dix, J.; Stolzenburg, F.; and Simari, G. 2003. Relating Defeasible and Normal Logic Programming through Transformation Properties. *Theoretical Computer Science* 290(1):499–529.

Chesñevar, C. I.; Simari, G.; Alsinet, T.; and Godo, L. 2004. A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge. In *Proc. of the Intl. Conf. in Uncertainty in Art. Intelligence. (UAI 2004). Banff, Canada*, 76–84.

Chesñevar, C.; Maguitman, A.; and Loui, R. 2000. Logical Models of Argument. *ACM Computing Surveys* 32(4):337–383.

Chesñevar, C.; Simari, G.; and Godo, L. 2005. Computing dialectical trees efficiently in possibilistic defeasible logic programming. *LNAI/LNCS Springer Series (Proc. of the 8th Intl. Conference on Logic Programming and Nonmonotonic Reasoning LPNMR 2005)* 158–171.

Dung, P.; Kowalski, R.; and Toni, F. 2006. Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence* 170(2):114–159.

Dung, P. 1993. On the Acceptability of Arguments and its Fundamental Role in Nomonotonic Reasoning and Logic Programming. In *Proceedings of the 13th. International Joint Conference in Artificial Intelligence (IJCAI), Chambáery, Francia*, 852–857.

Dung, P. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–358.

García, A., and Simari, G. 2004. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Hamblin, C. L. 1970. *Fallacies*. Methuen, London.

Hunter, A. 2004. Towards Higher Impact Argumentation. In *Proc. of the 19th American National Conf. on Artificial Intelligence (AAAI'2004)*, 275–280. MIT Press.

Jakobovits, H., and Vermeir, D. 1999. Dialectic semantics for argumentation frameworks. In *ICAIL*, 53–62.

Jakobovits, H. 1999. Robust semantics for argumentation frameworks. *Journal of Logic and Computation* 9(2):215–261.

Kakas, A., and Toni, F. 1999. Computing argumentation in logic programming. *Journal of Logic Programming* 9(4):515:562.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics* 7:25–75.

Prakken, H., and Vreeswijk, G. 2002. Logical Systems for Defeasible Argumentation. In Gabbay, D., and F.Guenther., eds., *Handbook of Philosophical Logic*. Kluwer Academic Publishers. 219–318.

Prakken, H. 2000. Relating protocols for dynamic dispute with logics for defeasible argumentation. *Synthese (to appear)*.

Prakken, H. 2001. Relating protocols for dynamic dispute with logics for defeasible argumentation. *Synthese (special issue on New Perspectives in Dialogical Logic)* 127(4):187:219.

Prakken, H. 2005. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation* 15:1009–1040.

Rescher, N. 1977. *Dialectics, a Controversy-Oriented Approach to the Theory of Knowledge*. Albany, USA: State University of New York Press.

Simari, G.; Chesñevar, C.; and García, A. 1994a. The Role of Dialectics in Defeasible Argumentation. In *Anales de la XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación*, 260–281. Universidad de Concepción, Concepción (Chile).

Simari, G. R.; Chesñevar, C. I.; and García, A. J. 1994b. Focusing Inference in Defeasible Argumentation. In *Proc. of IV Iberoamerican Congress on Artificial Intelligence,IBERAMIA'94*.

Vreeswijk, G. 1997. Abstract Argumentation Systems. *Artificial Intelligence* 90:225–279.

Walton, D. 1995. *A Pragmatic Theory of Fallacy (Studies in Rhetoric and Communication Series)*. The University of Alabama Press, Tuscaloosa.

5.6 Managing Deceitful Arguments with X-logics

Managing Deceitful Arguments with X-Logics

Geoffroy Aubry and **Vincent Risch**
InCA team, LSIS - UMR CNRS 6168
Domaine universitaire de Saint-Jérôme
13397 Marseilles cedex 20, France
email: {geoffroy.aubry, vincent.risch}@lsis.org

Abstract

In most works on negotiation dialogues, agents are supposed to be ideally honest. However, there are many situations where such a behaviour cannot always be expected from the agents (*e.g.* advertising, political negotiation, etc.). The aim of this paper is to reconsider the role of deceitful arguments in argumentation frameworks. We propose a logical tool for representing and handling deceitful arguments in a dialogue between two formal agents having to face their respective knowledge and trying to convince each other. X-logics, a nonmonotonic extension of classical propositional logics, is used as the background formalism for representing the reasoning of the agents on arguments. Starting from a previous work dedicated to the generation of new arguments, we propose to define the notion of lie as a new kind of possible agent's answer. Finally we describe the way an agent may trick and how the other agent may detect it.

Introduction

Logics has been historically assigned by philosophy the task of defining the rules of correct reasoning. As such, it was first considered as a tool in argumentation and rhetoric, before its formal developments in the foundation of mathematics meant to "delimit" it (in a large sense) to proof theory. A major distinction then arised from this situation: whereas the inner nature of arguments makes them questionable, theorems are held to be beyond dispute. A definitive break seemed to be set between logics and argumentation, leading a philosopher like Perelman to regard human argumentation as being beyond the reach of formal logics. New developments in logics however have conducted to reconsider partly this pessimistic view. In recent years the representation and the simulation of simplified models of argumentation with logical tools has been the object of important progresses (Prakken & Vreeswijk 2002). Obviously, all these models initially assume a form of exchange of arguments. In addition they address and characterize many distinct fundamental notions such as (for instance) acceptability (Dung 1995), preference among arguments (Amgoud & Cayrol 1998), argumentation trees (Besnard & Hunter 2001), relative strenght of arguments (Bench-Capon 2003), or dialectic proofs (Dung, Kowalski, & Toni 2006). Another important feature of these models is also that they assume some kind of rational behaviour in the exchange of arguments, that is that the agents (implicitly or explicitly) involved can be fully trusted. However, there are many contexts in which this assumption seems much too strong, typically situations in which the object of the discussion covers important issues for at least one part (*e.g.* financial or political negotiation, or yet, advertising). Actually the purpose of argumentation as a part of rhetoric is indeed to propose and stress values to which each agent believes or *feint to believe* regarding her respective goals. In this respect, the ability to represent, manage, and detect deceitful arguments appears as a major step toward a complete formal theory of argumentation, and as such, was already questioned in (Hamblin 1970)'s pioneering work. In this paper, and following (Aubry & Risch 2005), we address this question via the use of X-logics, a nonmonotonic extension of classical propositional logics. Extending their approach, devoted to the question of the generation of new arguments, we propose a logical approach to the notion of deceitful argument, and show how agents may generate lies while maintaining the consistency of their knowledge. The question of the detection of lies is briefly considered via the notion of *commitment store* (Hamblin 1970). Our paper is organized as follows: section 2 below briefly introduces standard notations, section 3 recalls X-logics, section 4 introduces the notions of agent, attitudes, kinds of answers and arguments, while section 5 concerns deceitful arguments.

Notations

Formally our language is classical propositional logic denoted by $\mathcal{L}$. Formulas are denoted by lowercase letters whereas sets of formulas are denoted by shift case letters. The symbols $\top$ and $\bot$ are the usual truth values, and $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$ the usual connectors. Classical consequence relation is denoted by $\vdash$. A finite set E of formulas is logically interpreted by the conjunction of its elements, that is a sentence. We abuse the notation $\neg E$ as a shorthand for the negation of the conjunction of the formulas in E, *e.g.* $E = \{e_1, \ldots, e_n\}$, hence $\neg E = \neg e_1 \vee \cdots \vee \neg e_n$. We denote by $\overline{E}$ the set of classical consequences of E (*i.e.* $\overline{E} = \{f \mid E \vdash f\}$), and by 2^E, the powerset of E. A finite *consistent* set of formulas is called a *knowledge base*.

X-logics

X-logics were defined in (Siegel & Forget 1996) as an attempt for defining a proof theory for nonmonotonic logics from any classical logic with a given set X of formulas. Whereas classically $K \vdash f$ iff $\overline{K \cup \{f\}} = \overline{K}$, X-logics can be considered as a generalization (hence a weakening) of $\vdash$, namely $\vdash_X$, defined such that $K \vdash_X f$ iff $\overline{K \cup \{f\}} \cap X = \overline{K} \cap X$, *i.e.* $\vdash_X$ is monotonic only on X. When $X = \mathcal{L}$, $\vdash_X$ amounts to be just $\vdash$. If $X = \{\bot\}$ then $K \vdash_X f$ is equivalent to $K \nvdash \neg f$ which describes the consistency relation between K and f ("$K \wedge f$ is satisfiable" holds), provided K is consistent by itself. If $X = \varnothing$, all the formulas can be entailed. Note that $K \vdash_X f$ if every theorem (regarding $\vdash$) of $K \cup \{f\}$ which is in X is a theorem of K (by adding f to K the set of classical theorems which are in X does not grow). Indeed, since classical consequence relation is monotonic, in order to check whether $K \vdash_X f$ it is sufficient to check whether $\overline{K \cup \{f\}} \cap X \subseteq \overline{K}$. In other words, $K \vdash_X f$ iff $\forall x \in X \setminus \overline{K},\ K \cup \{f\} \nvdash x$. Although this was already proved independently, this shows that X-logics are supraclassical. Actually and as shown in (Bochman 2003), X-logics coincide with permissive inference relations which are completely characterized by Left Logical Equivalence, Right Weakening, Reflexivity, Conjunctive Cautious Monotony, Cut and Or.

Let us make use of the following terminology: if $K \vdash_X f$ we say that f is *compatible* with K regarding X, and *incompatible* otherwise. The notion of compatibility encompasses the notion of consistency, whereas formulas can be incompatible with K regarding X without being inconsistent with K. The following properties obviously hold:

Property 1.

1. (metacoherence) *A formula cannot be both compatible and incompatible.*
2. (paraconsistency of compatibility) *Both a formula and its negation can be compatible with K regarding X.*
3. (paraconsistency of incompatibility) *Both a formula and its negation can be incompatible with K regarding X.*

Example 2.

- $\{a\} \vdash_{\{\bot\}} a \wedge b$, and $\{a\} \vdash_{\{\bot\}} \neg(a \wedge b)$
- $\{a\} \nvdash_{\{\bot, b, \neg b\}} a \wedge b$, and $\{a\} \nvdash_{\{\bot, b, \neg b\}} \neg(a \wedge b)$
- $\{a\} \vdash_{\{b \wedge c\}} b$, but $\{a, c\} \nvdash_{\{b \wedge c\}} b$

Agents, attitudes, answers and arguments

In the literature, some argumentation theories consider the notion of *proponent-opponent* (Rescher 1977; Vreeswijk 1992) whereas other describe argumentation systems in which arguments made from a unique set of formulas are linked together, in a kind of abstract game among arguments (Lin & Shoham 1989; Dung 1995; Amgoud & Cayrol 1998; Besnard & Hunter 2001). Following (Simari & Loui 1992; Amgoud & Parsons 2002), (Aubry & Risch 2005), we introduce a notion of *agent*, but with the objective to map each agent with a unique X-inference.

Definition 3. *(Aubry & Risch 2005) An* agent *is a couple $[K, X]$ where K is a knowledge base, and $X \supseteq \{\bot\}$, a set of formulas. The set of agents, a subset of $2^{\mathcal{L}} \times 2^{\mathcal{L}}$, is denoted by $\mathcal{A}$.*

Compatibility extends naturally to the notion of admissibility of a formula by an agent, *i.e.* a formula is *admissible* by an agent $[K, X]$ iff this formula is compatible with K regarding X; it is *non-admissible* otherwise [1]. Intuitively K is used as a representation of the factual knowledge of an agent, whereas X corresponds to formulas that an agent cannot admit unless they are part of her factual knowledge. In other words, formulas in X delineate negatively the hopes of the agent (since the agent does not admit these formulas), whereas the positive counterpart indeed should correspond to the agent's expectations (the agent accept everything but formulas of X, unless she has to take account of them because there are already part of her knowledge). In the following, we will call X the *forbidden formulas*. The requirement that X contains at least the contradiction is motivated by the natural expectation that an agent should reason consistently. The notion of admissibility determines four possible distinct *attitudes* that an agent may adopt concerning a given formula, as shown in (Aubry & Risch 2005):

Definition 4 (Attitudes)**.** *(Aubry & Risch 2005) Consider an agent $[K, X]$ and a formula f:*

- $[K, X]$ *is* for f *iff* $K \vdash_X f$ *and* $K \nvdash_X \neg f$
- $[K, X]$ *is* neutral *about* f *iff* $K \vdash_X f$ *and* $K \vdash_X \neg f$
- $[K, X]$ *is* puzzled *by* f *iff* $K \nvdash_X f$ *and* $K \nvdash_X \neg f$
- $[K, X]$ *is* against f *iff* $K \nvdash_X f$ *and* $K \vdash_X \neg f$

By extension, an agent is for *(resp.* neutral *about,* against*,* puzzled *by) a set of formulas iff she is* for *(resp.* neutral *about,* against*,* puzzled *by) the conjunction of the formulas of this set.*

In figure 1, the four corners of the median layout are associated with the attitudes, and are clearly generated by both the two edges above and below corresponding to the admissible or non-admissible character of f and $\neg f$ respectively.

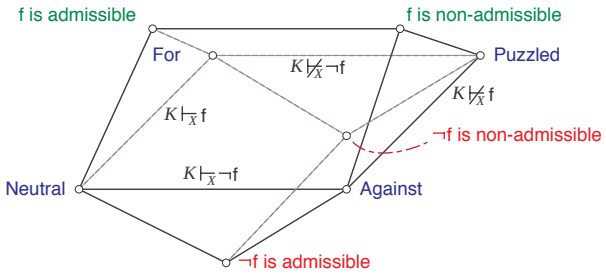


Figure 1: Generative octahedron of attitudes of an agent $[K, X]$ in front of a formula f

[1] Note that our notion of admissibility differs from the notion previously defined in (Bondarenko *et al.* 1997) and in (Dung, Kowalski, & Toni 2006).

Yet, as shown in (Aubry & Risch 2005), given an agent Φ and a formula f:

- Φ is for f iff she is against $\neg f$,
- Φ is neutral about f iff she is neutral about $\neg f$,
- Φ is puzzled by f iff she is puzzled by $\neg f$,
- Φ is for the tautologies and against the contradictions.

The different possible attitudes of an agent in front of a set of formulas yield a partition of this set. *Confrontation operators* are introduced in order to associate each attitude with the formulas of a particular partition.

Definition 5. *(Aubry & Risch 2005) The* operator $|_+$ *(resp.* $|_0, |_-$ *and* $|_p$*) maps an agent and a set* E *of formulas with the subsets of* E *such that this agent is* for *(resp.* neutral *about,* against *or* puzzled *by) these subsets:*

$$\begin{array}{rccc} |_+ : & \mathcal{A} \times 2^{\mathcal{L}} & \longrightarrow & 2^{2^{\mathcal{L}}} \\ & \Phi \,|_+\, E & \longmapsto & \{P \subseteq E \mid \Phi \textit{ is } \textsf{for } P\} \end{array}$$

As shown further down, confrontation operators are meant to be used by an agent for deciding precisely which are the points of agreement or disagreement she has with the different parts of a given argument.

Example 6. Consider Φ an agent with the following knowledge base and set of forbidden formulas (where A and B denote respectively "Amelia" and "Brandon"): $K_\Phi = \{\textit{B-comes} \Rightarrow \textit{Annoyed}, \neg\textit{B-comes} \Rightarrow \textit{Sad}, \textit{A-comes} \Leftrightarrow \textit{Happy}\}$, $X_\Phi = \{\bot, \textit{Annoyed}, \textit{Sad}\}$. The attitudes of Φ regarding the set $E = \{\textit{B-comes}, \textit{A-comes}, \textit{B-comes} \Rightarrow \textit{Annoyed}, \textit{A-comes} \wedge \neg\textit{Happy}\}$ yield the following partition of E:

$\Phi \,|_+\, E = \big\{\{\textit{B-comes} \Rightarrow \textit{Annoyed}\}\big\}$
$\Phi \,|_0\, E = \big\{\{\textit{A-comes}\}, \{\textit{A-comes}, \textit{B-comes} \Rightarrow \textit{Annoyed}\}\big\}$
$\Phi \,|_-\, E = \big\{\{\textit{A-comes} \wedge \neg\textit{Happy}\}, \{\textit{B-comes}, \textit{A-comes}\}, *\big\}$
$\Phi \,|_p\, E = \big\{\{\textit{B-comes}\}, \{\textit{B-comes}, \textit{B-comes} \Rightarrow \textit{Annoyed}\}\big\}$

The symbol $*$ in $\Phi \,|_-\, E$ stands for each subset of E containing either of the two other sets stated in $\Phi \,|_-\, E$.

In order to link the attitudes of an agent with the construction of new arguments (Aubry & Risch 2005) make use of the following notion of *answer*. Roughly, an answer is a set of formulas fixed by the attitude of an agent regarding a given set of formulas this agent is faced with. Consider $X = \{x_1, \ldots, x_n\}$, and let us use conceivable (X) as a shorthand for $\{\neg x_1, \ldots, \neg x_n\}$.

Definition 7 (Answer)**.** *(Aubry & Risch 2005) An* answer *of the agent* $[K, X]$ *to a consistent set* A *of formulas is a consistent set* R *of formulas such that, for some* $K' \subseteq K$ *and some* $X' \subseteq (X \setminus \{\bot\})$*:*

1. $R = K' \cup$ conceivable (X')
2. $K' \nvdash_{\{\bot\} \cup X'} A$

The set of answers given by $[K, X]$ *to* A *is written* $\mathcal{R}^A_{[K,X]}$.

The first point constrains answers to contain only knowledge or negations of forbidden formulas of the agent. The second point specifies that an answer to a set A necessarily contains formulas which are conflicting with A: $\exists x \in (\{\bot\} \cup X') \setminus \overline{K'},\ K' \cup A \vdash x$. In other words, an answer $K' \cup$ conceivable (X') to A is such that A is non-admissible by the "virtual" agent $[K', \{\bot\} \cup X']$. The contradiction is present here for technical reasons: for instance, this allows to answer $R = K' = \{a\}$ to $A = \{\neg a\}$ without having to add $X' = \{\bot\}$ in R.

Following directly from definition 7: *(1)* no answer is empty, *(2)* any answer of an agent to a given set of formulas is inconsistent with this set. In addition, the following property holds:

Property 8 (Existence of an answer)**.** *(Aubry & Risch 2005) If an agent is* against *or* puzzled *about a subset of a consistent set of formulas, then there exists an answer of this agent to this set. The opposite does not hold.*

In the following, we consider two easy but important refinements of definition 7, namely *coherent*, and *relevant* answers. Let us define the first of these two notions:

Definition 9 (Coherent answer)**.** *An answer of the agent* $\Phi = [K, X]$ *is called a* coherent answer *of* Φ *iff it is consistent with* K.

As stated further down, each answer of an agent Φ to A (and especially coherent answers) is potentially the support of a counterargument of Φ (that is the reason why to believe the conclusion of this counterargument) to an argument containing A. Note however this does not limit agents to generate only disputing arguments (remind that Φ is for A iff she is against $\neg A$). Answers that are not coherent will drive us to deceitful arguments, and as such definition 9 plays an important role as the counterpart of the notion of lie, considered further down.

Let us now come to the second refinement of definition 7. In most works on argumentation, only minimal arguments are considered: they only contains formulas necessary to elaborate the conclusion of the argument. Among the possible answers of an agent to a set of formulas A, some of them are included in others: they contain less superfluous information. In the limit case, the minimal answers are only made with formulas necessary to elaborate the conclusion $\neg A$. Such answers are called *relevant* [2].

Definition 10 (Relevant answer)**.** *(Aubry & Risch 2005) An answer of the agent* Φ *to a set* A *of formulas is called* relevant *iff it does not contain any other answer of* $\mathcal{R}^A_\Phi$*. The set of relevant answers given by* Φ *to* A *is written* $\mathcal{R}r^A_\Phi$.

$$\mathcal{R}r^A_\Phi \;=\; \big\{R \in \mathcal{R}^A_\Phi \mid \forall R' \subset R,\ R' \notin \mathcal{R}^A_\Phi\big\}$$

Now, given an agent $[K, X]$ facing a set A of formulas, her set of relevant answers can be shared among three subsets. The first contains the answers only made from the knowledge of this agent. Hence this subset is not empty when $K \cup A$ is inconsistent. The second category of answers is made of those exclusively constructed from the forbidden

[2] Note that the notion of *relevant move* previously defined in (Prakken 2005) has a different meaning, since it is defined in the context of a dialogue.

formulas of this agent. Finally, the relevant answers neither in the first nor in the second subset are made from at least one formula of the set of knowledge of this agent and one formula of the set of forbidden formulas of this agent.

This partitioning can be considered via "virtual" agents. The answers only made from the knowledge of the agent $[K, X]$ are actually answers of an agent $[K, \{\bot\}]$, while the answers only made of the forbidden formulas of the agent $[K, X]$ are answers of the agent $[\varnothing, X]$.

These different sets of relevant answers are fully characterized in (Aubry & Risch 2005).

Answers are used as a tool for generating new arguments by an agent, where arguments are defined following a very common intuitive view (Simari & Loui 1992; Elvang-Gøransson, Krause, & Fox 1993; Amgoud & Cayrol 1998; Besnard & Hunter 2001), *i.e.* an *argument* is a set of relevant formulas that can be used to classically prove some formula, together with that formula. This notion is made more precise here by taking account of both notions of agent and answer.

Definition 11 (Argument)**.** *(Aubry & Risch 2005) Consider A, a set of formulas. An* argument *α of an agent Φ is any pair $\langle R, \neg A\rangle$ such that R is a relevant answer of Φ to A. The set of arguments of an agent Φ is written $\mathcal{A}rg_\Phi$, i.e.*

$$\forall \Phi, \forall R, \forall A, \quad \langle R, \neg A\rangle \in \mathcal{A}rg_\Phi \text{ iff } R \in \mathcal{R}r_\Phi^A$$

The set of all arguments is denoted by $\mathcal{A}rg$. Finally, R is called the support *of the argument, denoted by* $\mathsf{supp}(\alpha)$*, while $\neg A$ is called the* conclusion *of the argument, denoted by* $\mathsf{concl}(\alpha)$.

From the definition of an answer (definition 7), we have indeed that the conclusion of an argument is classically entailed by the support of this argument. Let us now address the question of the use of arguments by an agent. The two complementary notions of *attack* and *defense* of an argument in the context of a dispute are well known in philosophy. Following a solid tradition (*e.g.* (Schopenhauer 2004)) we consider that an argument can be attacked (resp. defended) either on the premises (the support) or on the conclusion. Hence we allow arguments to be decomposed into *elements* that an agent can analyze for such further attack or defense. Thus, the elements of an argument are taken as parts of the support, together with the conclusion:

Definition 12 (Elements of an argument)**.** *The* elements *of an argument $\langle S, c\rangle$, written* $\mathsf{elements}(\langle S, c\rangle)$*, are given by a mapping from $\mathcal{A}rg$ to $2^{2^\mathcal{L}}$ such that:*

$$\mathsf{elements}(\langle S, c\rangle) = \{E \in 2^\mathcal{L} \mid E \subseteq S\} \cup \{\{c\}\}$$

The two relations of attack and defense are then defined classically. Note however that generally, defending arguments is considered through *reinstatement*: an argument that is defeated by another argument can be justified only if it is reinstated by a third argument (this corresponds to (Dung 1995)'s notion of *acceptability*). Actually, the notions of attitude and the generation of agent's answers allow us to define a pure relation of defense of arguments.

Definition 13. *The set of arguments of an agent Φ* attacking *(resp.* defending*) an argument α is written $\mathcal{A}rg_\Phi^{\mathsf{att}(\alpha)}$ (resp. $\mathcal{A}rg_\Phi^{\mathsf{def}(\alpha)}$), where:*

- $\mathcal{A}rg_\Phi^{\mathsf{att}(\alpha)} = \{\langle R, \neg A\rangle \mid R \in \mathcal{R}r_\Phi^A,\ A \in \mathsf{elements}(\alpha)\}$
- $\mathcal{A}rg_\Phi^{\mathsf{def}(\alpha)} = \{\langle R, \bigwedge_{a\in A} A\rangle \mid R \in \mathcal{R}r_\Phi^{\{\neg A\}},\ A \in \mathsf{elements}(\alpha)\}$

The following property links the attack and the defense of arguments with the attitudes of an agent:

Property 14. *Consider $t \in \{\mathsf{supp}(), \mathsf{concl}()\}$. $\forall \Phi, \forall A, \forall \alpha$:*

- $A \in \Phi \mid_- t(\alpha) \Rightarrow \exists R,\ \langle R, \neg A\rangle \in \mathcal{A}rg_\Phi^{\mathsf{att}(\alpha)}$
- $A \in \Phi \mid_+ t(\alpha) \Rightarrow \exists R,\ \langle R, \bigwedge_{a\in A} a\rangle \in \mathcal{A}rg_\Phi^{\mathsf{def}(\alpha)}$
- $A \in \Phi \mid_p t(\alpha) \Rightarrow \left\{\begin{array}{l}\exists R_1, \exists R_2,\\ \langle R_1, \neg A\rangle \in \mathcal{A}rg_\Phi^{\mathsf{att}(\alpha)}\\ \langle R_2, \bigwedge_{a\in A} a\rangle \in \mathcal{A}rg_\Phi^{\mathsf{def}(\alpha)}\end{array}\right.$

Sketch of proof. Applying property 8 then definition 11. □

Note that property 14 makes our intuition about the attitudes of an agent to coincide with her expected behaviour: an agent is puzzled by one element of an argument because she can both attack and defend this argument. Similarly, the fact that an agent is neutral about some element of an argument does not allow this agent to construct any coherent answer, and hence does not allow her to attack or to defend this argument (unless by lying as seen further down).

Example 15. Consider two agents Φ and Ψ arguing about the text of the future European constitution (denoted by x) in order to decide whether x should be accepted or not. The knowledge of Φ regarding x is that x mentions the concept of trade market, that x pretends to be a constitution as well as to set up new political foundations for Europe, and that the need of political foundations requires a constitution anyway. On the other hand, Φ is not ready to give up the idea that a constitution should not include any reference to trade markets. Hence, we have $K_\Phi = \{x, x \Rightarrow \textit{tradeM}, x \Rightarrow \textit{constitution}, x \Rightarrow \textit{newPoliticalF}, \textit{newPoliticalF} \Rightarrow \textit{constitution}\}$, and $X_\Phi = \{\bot, \neg((\textit{constitution} \wedge \textit{tradeM}) \Rightarrow \neg\textit{admissible})\}$. The knowledge of the agent Ψ is that x refers to trade markets, and as such, has to be considered a treaty rather than a constitution. Moreover Ψ thinks that a treaty can be accepted. Hence, $K_\Psi = \{x, x \Rightarrow \textit{tradeM}, x \Rightarrow \textit{treaty}, \textit{treaty} \Leftrightarrow \neg\textit{constitution}, \textit{treaty} \Rightarrow \textit{admissible}\}$, while $X_\Psi = \{\bot\}$.

Assume now the claim made by Ψ in front of Φ that x should be accepted, *i.e.* $\alpha^1_\Psi = \langle\{x \Rightarrow \textit{treaty}, \textit{treaty} \Rightarrow \textit{admissible}\}, x \Rightarrow \textit{admissible}\rangle$.

Since $\{x \Rightarrow \textit{admissible}\} \in \Phi \mid_- \mathsf{concl}(\alpha^1_\Psi)$, Φ can compute a counterargument: $\alpha^1_\Phi = \langle\{x, x \Rightarrow \textit{tradeM}, x \Rightarrow \textit{constitution}, (\textit{constitution} \wedge \textit{tradeM}) \Rightarrow \neg\textit{admissible})\}, \neg(x \Rightarrow \textit{admissible})\rangle$.

Now Ψ analyzes α^1_Φ: $\Psi \mid_+ \mathsf{supp}(\alpha^1_\Phi) = \{\{x\}, \{x \Rightarrow \textit{tradeM}\}, \{(\textit{constitution} \wedge \textit{tradeM}) \Rightarrow \neg\textit{admissible})\}, *\}$, where $*$ stands for each subset of the set containing $*$, and Ψ

is against any other elements of α^1_Φ. Ψ set up the following argument: $\alpha^2_\Psi = \langle\{x, x \Rightarrow \textit{treaty}, \textit{treaty} \Leftrightarrow \neg\textit{constitution}\}, \neg(x \Rightarrow \textit{constitution})\rangle$. On her turn, Φ defends her position: $\alpha^2_\Phi = \langle\{x \Rightarrow \textit{newPoliticalF}, \textit{newPoliticalF} \Rightarrow \textit{constitution}\}, \neg\neg(x \Rightarrow \textit{constitution})\rangle$.

Deceitful arguments

Deceitful arguments are untruthful arguments made with the intention to deceive. An agent using such kind of argument cheats with her current knowledge about the world, and does so with the intention to trick the other agent. We consider here two ways of cheating: the first one has to do with some kind of dilution of an argument, whereas the second one refers directly to lies. Let us informaly consider dilution first.

Given an initial argument, in order to narrow the possibilities of counterarguments to consider, (Besnard & Hunter 2001) introduce the following notion of *conservativity*:

Definition 16. *(Besnard & Hunter 2001) An argument $\langle S, c\rangle$ is* more conservative *than an argument $\langle S', c'\rangle$ iff $S \subseteq S'$ and $c' \vdash c$.*

Selecting only the most conservative counterarguments allows to summarize the different possibilities of answer (as shown in (Aubry & Risch 2005)). However, depending on the goal of the agent beginning a discussion, the most conservative argument may not be the most suitable one as first argument:

Example 17. Consider a commercial agent Φ who wants sell a scooter of trademark *label-Z* to a client Ψ. $\Phi = [\{\textit{scooter}, \textit{label-Z}, \textit{scooter} \Rightarrow \textit{edgeOut}, \textit{edgeOut} \Rightarrow \neg\textit{trafficJam}\}, \{\bot\}]$.

Φ uses the following argument: $\langle\{\textit{scooter} \Rightarrow \textit{edgeOut}, \textit{edgeOut} \Rightarrow \neg\textit{trafficJam}\}, (\textit{scooter} \wedge \textit{label-Z}) \Rightarrow \neg\textit{trafficJam}\rangle$. But the agent Ψ considers it as a *diluted argument* since she does not understand why an unsophisticated scooter is not enough to avoid the traffic jams. She can then address the following argument to the commercial agent: $\langle\{\textit{scooter} \Rightarrow \textit{edgeOut}, \textit{edgeOut} \Rightarrow \neg\textit{trafficJam}\}, \textit{scooter} \Rightarrow \neg\textit{trafficJam}\rangle$. Since Φ is for each element of this argument, she cannot generate any counterargument.

Hence depending on how much Ψ is careful, the first argument of Φ seems to be useless while a less conservative argument may appear more appropriate. A new kind of answer can assist our commercial agent to stick up for herself, which drive us two the second way by which an agent may cheat:

Definition 18 (Lie)**.** *M is a* lie *of the agent Φ regarding the set of formulas A iff both M is a relevant answer of Φ to A, and M is not a coherent answer of Φ. The set of lies of Φ regarding A is denoted by $\mathcal{RL}^A_\Phi$.*

The ability for an agent to construct answers inconsistent with her knowledge relies on the use of an X-inference in the definition of an answer (definition 7), and precisely on the fact that an agent $[K, X]$ can use a formula from $X \cap \overline{K}$. Indeed, $K \nvdash_X A$ iff $\exists x \in X \setminus \overline{K},\ K \cup A \vdash x$. But a given answer $R = K' \cup \text{conceivable}(X')$ of this agent to A only satisfies $K' \nvdash_{\{\bot\}\cup X'} A$, that is: $\exists x \in (\{\bot\} \cup X') \setminus \overline{K'},\ K' \cup A \vdash x$. Hence a formula of $X \cap \overline{K} \setminus K'$ is the reason why $K \nvdash_X A$, which explains why the answer is inconsistent with the knowledge of the agent.

Property 19. *Let M be a relevant answer of the agent $[K, X]$ to the set A of formulas. M is a lie of this agent regarding A iff there exists a formula of $M \setminus K$ inconsistent with K.*

Proof. ($\Rightarrow$) Since $M \in \mathcal{RL}^A_{[K,X]}$, $K \cup (M \setminus K) \vdash \bot$. It has been proved in (Aubry & Risch 2005) that for any answer R of $[K, X]$, $R \setminus K$ is either empty or a singleton. Hence $\exists m \in M \setminus K,\ K \cup \{m\} \vdash \bot$. ($\Leftarrow$) By assumption, $\exists m \in M \setminus K,\ K \cup \{m\} \vdash \bot$. Hence $K \cup M \vdash \bot$. From definition 18, it follows that $M \in \mathcal{RL}^A_{[K,X]}$. □

Corollary 20. *For every agent $[K, X]$ and for every set of formulas A, if the intersection of X and the deductive closure of K is empty, then this agent cannot generate any lie regarding A: $\forall [K, X], \forall A, X \cap \overline{K} = \varnothing \ \Rightarrow\ \mathcal{RL}^A_{[K,X]} = \varnothing$*

Proof. We prove the contrapositive, that is that $\mathcal{RL}^A_{[K,X]} \neq \varnothing \ \Rightarrow\ X \cap \overline{K} \neq \varnothing$. Assume $M \in \mathcal{RL}^A_{[K,X]}$. From property 19, $\exists m \in M \setminus K,\ K \cup \{m\} \vdash \bot$. Then $\exists m \in \text{conceivable}(X), K \vdash \neg m$. Hence $\exists x \in X, K \vdash x$, *i.e.* $X \cap \overline{K} \neq \varnothing$. □

This corollary leads to the conclusion that an agent with no knowledge ($[\varnothing, X]$), or whose way of reasoning is reduced to sole consistency ($[K, \{\bot\}]$) cannot generate any lie. Note however that a relevant answer of the agent $[\varnothing, X]$ to a set A can be a lie regarding A for an agent with the same set of forbidden formulas but with an adequate knowledge base:

Example 21. Let both Φ and Ψ be two agents such that $\Phi = [\varnothing, \{\bot, a\}]$, and $\Psi = [\{a\}, \{\bot, a\}]$. With $A = \{a\}$ we get: $\mathcal{RL}^A_\Phi = \varnothing$, and $\mathcal{R}r^A_\Phi = \{\{\neg a\}\}$, but: $\mathcal{RL}^A_\Psi = \{\{\neg a\}\}$.

A lie can be generated for attacking an argument while defending another argument:

Example 22 (Lie, attack, and defense)**.** Let Φ be an agent such that: $\Phi = [\{a, a \Rightarrow (b \wedge c)\}, \{\bot, a \wedge b\}]$. An agent Ψ set an argument α such that $\{b\} \in \text{elements}(\alpha)$. Then Φ could attack α while generating a lie: $\beta = \langle\{a, \neg(a \wedge b)\}, \neg b\rangle \in \mathcal{A}rg^{\text{att}(\alpha)}_\Phi$, and $\{a, \neg(a \wedge b)\} \in \mathcal{RL}^{\{b\}}_\Phi$. But if Ψ had set an argument α' with $\{\neg b\} \in \text{elements}(\alpha')$, then Φ could have defended α' with both the same lie and the same argument β: $\beta \in \mathcal{A}rg^{\text{def}(\alpha')}_\Phi$.

Finally, suppose that an agent Φ generates an argument α constructed from a lie, for answering an argument β. If $\alpha \in \mathcal{A}rg^{\text{def}(\beta)}_\Phi$ (resp. $\alpha \in \mathcal{A}rg^{\text{att}(\beta)}_\Phi$), we cannot presume whether there exists a coherent argument (*i.e.* not a lie) of Φ attacking β (resp. defending β). In other words we cannot assume that Φ is against (resp. for) any element of β:

Example 23 (Lie and neutral attitude)**.** Let Φ be an agent such that: $\Phi = [\{a\}, \{\bot, \neg b \vee a\}]$. Now consider an argument α such that $A = \{\neg b\}$ belongs to elements (α). Then Φ is neutral about A, but she could set up a counterargument to α, as far as it is constructed from a lie: $\langle\{\neg(\neg b \vee a)\}, b\rangle$.

Coming back to example 17, conferring the commercial agent the ability to lie allows her to push further her initial argument.

Example 24 (continuation of example 17)**.** Consider a commercial agent Φ' who still wants to sell a scooter of trademark *label-Z* to a client Ψ. $\Phi' = [\{scooter, label\text{-}Z, scooter \Rightarrow edgeOut, edgeOut \Rightarrow \neg trafficJam\}, \{\bot, (\boldsymbol{scooter} \wedge \boldsymbol{\neg label\text{-}Z}) \Rightarrow \boldsymbol{\neg trafficJam}\}]$.

Φ' set up the following argument: $\langle\{scooter \Rightarrow edgeOut, edgeOut \Rightarrow \neg trafficJam\}, (scooter \wedge label\text{-}Z) \Rightarrow \neg trafficJam\rangle$. The client address the following argument to the commercial agent: $\langle\{scooter \Rightarrow edgeOut, edgeOut \Rightarrow \neg trafficJam\}, scooter \Rightarrow \neg trafficJam\rangle$. Just like Φ in the example 17, the agent Φ' is for each element of this argument, *but* she can now generate a counterargument: $\langle\{\neg((\boldsymbol{scooter} \wedge \boldsymbol{\neg label\text{-}Z}) \Rightarrow \boldsymbol{\neg trafficJam})\}, \neg(scooter \Rightarrow \neg trafficJam)\rangle$.

Lies can be advantageous in negotiation dialogues when a goal is to be achieved. But it is possible to counterbalance this. Since the work of the philosopher Hamblin (Hamblin 1970), formal dialogue systems typically establish and maintain public sets of commitments called *commitment stores* for each agent. More than one notion of commitment is present in the literature on dialogues games [3] but essentially they can bee seen as a tool of memorization for the arguments advanced by each agent. If an agent Ψ considers an argument α from an agent Φ as a diluted argument (such as for instance in examples 17 and 24), then Ψ can question Φ about one or the other of the elements of α, and hope either a clarification (in example 17 where finally Φ agree with Ψ), or detect an inconsistency in the commitment store of Φ (in example 24 where Φ' has advanced two arguments whose supports are inconsistent together).

Conclusion

X-logics allows agents to cope with singular answers, namely those ones allowing the agent to produce an argument inconsistent with her knowledge, while keeping consistency in the knowledge base. This ability relies on the possibility for the agent to use expectations (if any) that contradict her knowledge. This way, a formal notion of lie is defined that seems to match the standard intuition. Moreover, a very simplified form of commitment store allows to define a formal way for detecting such lies. Of course, many questions still deserve to be addressed. A first question may concern how our system could be embedded in standard models of dialogue. Especially, having defined how agents may cheat, we are now concerned with the notion of strategy of an argumentative exchange. In this respect, it might be interesting to see how our system could be used for the formalization of some of the strategies described in (Schopenhauer 2004). Another may be more exciting question in the same direction concerns how an agent can exchange arguments with herself, thus allowing the representation of a form of introspection. Finally, some technical questions are still unanswered or under study, such as a link between the four possible attitudes of an agent with Belnap's four valued logic, or the comparison with the three *attitudes* described in (Parsons, Wooldridge, & Amgoud 2003).

[3] For recent works on the question, see for instance (Maudet & Chaib-draa 2003; Bentahar *et al.* 2004).

Acknowledgments

The authors are grateful to the anonymous reviewer for their helpful comments.

References

Amgoud, L., and Cayrol, C. 1998. On the acceptability of arguments in preference-based argumentation. In *UAI*, 1–7.

Amgoud, L., and Parsons, S. 2002. Agent dialogues with conflicting preferences. In *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, 190–205. London, UK: Springer-Verlag.

Aubry, G., and Risch, V. 2005. Toward a logical tool for generating new arguments in an argumentation based framework. In *ICTAI'05*. Hong Kong, China: IEEE Computer Society. ISBN: 0–7695-2488–5.

Bench-Capon, T. 2003. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation* 13(3):429–448.

Bentahar, J.; Moulin, B.; Meyer, J.-J. C.; and Chaib-draa, B. 2004. A logical model for commitment and argument network for agent communication. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 792–799. Washington, DC, USA: IEEE Computer Society.

Besnard, P., and Hunter, A. 2001. A logic-based theory of deductive arguments. *Artificial Intelligence* 128(1-2):203–235.

Bochman, A. 2003. Brave nonmonotonic inference and its kinds. *Annals of Mathematics and Artificial Intelligence* 39(1–2):101–121.

Bondarenko, A.; Dung, P.; Kowalski, R.; and Toni, F. 1997. An abstract, argumentation-theoretic framework for default reasoning. *Artificial Intelligence* 93(1-2):63–101.

Dung, P. M.; Kowalski, R. A.; and Toni, F. 2006. Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence* 170(2):114–159.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–358.

Elvang-Gøransson, M.; Krause, P.; and Fox, J. 1993. Dialectic reasoning with inconsistent information. In Heckerman, D., and Mamdani, E. H., eds., *UAI '93, The Catholic University of America, Providence, Washington, DC, USA*, 114–121. Morgan Kaufmann.

Hamblin, C. L. 1970. *Fallacies*. Methuen.

Lin, F., and Shoham, Y. 1989. Argument systems: A uniform basis for nonmonotonic reasoning. In Kaufmann, M., ed., *KR'89*, 245–255.

Maudet, N., and Chaib-draa, B. 2003. Commitment-based and dialogue-game based protocols – new trends in agent communication language. *Knowledge Engineering Review* 17(2):157–179.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2003. Properties and complexity of some formal inter-agent dialogues. *J. Log. Comput.* 13(3):347–376.

Prakken, H., and Vreeswijk, G. 2002. Logical systems for defeasible argumentation. In Gabbay, D., and Guenthner, F., eds., *Handbook of Philosophical Logic*, volume 4. Dordrecht: Kluwer Academic, 2nd edition. 219–318.

Prakken, H. 2005. Coherence and flexibility in dialogue games for argumentation. *Journal of Logic and Computation* 15:1009–1040.

Rescher, N. 1977. *Dialectics, A Controversy-Oriented Approach to the Theory of Knowledge*. State University of New York Press, Albany.

Schopenhauer, A. 2004. *The Art Of Controversy*. Kessinger Publishing. translated by T. Bailey Saunders.

Siegel, P., and Forget, L. 1996. A representation theorem for preferential logics. In *KR'96*, 453–460. Morgan Kaufmann.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53(2-3):125–157.

Vreeswijk, G. 1992. Reasoning with defeasible arguments: Examples and applications. In Pearce, D., and Wagner, G., eds., *Logics in AI: Proc. of the European Workshop JELIA'92*. Berlin, Heidelberg: Springer. 189–211.

5.7 Comparing Decisions in an Argumentation-based Setting

Comparing decisions on the basis of a bipolar typology of arguments

Leila Amgoud and **Henri Prade**

Institut de Recherche en Informatique de Toulouse
118, route de Narbonne, 31062 Toulouse, France
{amgoud, prade}@irit.fr

Abstract

Arguments play two types of roles w.r.t. decision, namely helping to select an alternative, or to explain a choice. Until now, the various attempts at formalizing argument-based decision making have relied only on one type of arguments, in favor of or against alternatives.

The paper proposes a systematic typology that identifies eight types of arguments, some of them being weaker than others. First the setting emphasizes the *bipolar* nature of the evaluation of decision results by making an explicit distinction between prioritized *goals* to be pursued, and prioritized *rejections* that are stumbling blocks to be avoided. This is the basis for an argumentative framework for decision. Each decision is *supported* by arguments emphasizing its positive consequences in terms of goals certainly satisfied, goals possibly satisfied, rejections certainly avoided and rejections possibly avoided. A decision can also be *attacked* by arguments emphasizing its negative consequences in terms of certainly or possibly missed goals, or rejections certainly or possibly led to by that decision. The logical properties of this framework are studied. The richness of the proposed typology makes it possible to partition the set of alternatives into four classes, giving thus a *status* to decisions, which may be *recommended*, *discommended*, *controversial* or *neutral*. Each class may be refined into sub-classes taking advantage of the strengths of the different types of arguments. This typology is also helpful from an explanation point of view for being able to use the right type of arguments depending on the context.

The paper also presents a preliminary investigation on *decision principles* that can be used for comparing decisions. Three classes of principles can be considered: *unipolar*, *bipolar* or *non-polar* principles depending on whether i) only arguments pro or only arguments cons, or ii) both types, or iii) an aggregation of them into a meta-argument are used.

Key words: Decision making, Argumentation

Introduction

Decision making, often viewed as a form of reasoning toward action, has raised the interest of many scholars including philosophers, economists, psychologists, and computer scientists for a long time. Any decision problem amounts to select the best option(s) among different alternatives.

The decision problem has been considered from different points of view. *Classical* decision theory, as developed by economists, has focused mainly on identifying *criteria* for comparing different alternatives. The inputs of this approach are a set of *feasible* actions, and a function that assesses the value of their consequences when the actions are performed in a given state. The output is a preference relation between actions. A decision criterion, such as the classical expected utility (Savage 1954), should then be justified on the basis of a set of postulates to which the preference relation between action should obey. Note that such an approach considers a group of candidate actions as a whole rather than focusing on a candidate action individually. Moreover, the candidate actions are supposed to be feasible.

More recently, some researchers in AI have advocated the need for a new approach in which the different aspects that may be involved in a decision problem (such as the goals of the agent, the feasibility of an action, its consequences, the conflicts between goals, the alternative plans for achieving the same goal, etc) can be handled. In (Bratman 1987; Bratman, Israel, & Pollack 1988), it has been argued that this can be done by representing the cognitive states, namely agent's beliefs, desires and intentions (thus the so-called *BDI* architecture). The decision problem is then to select among the conflicting desires a consistent and feasible subset that will constitute the intentions. The above line of research takes its inspiration in the work of philosophers who have advocated practical reasoning (Raz 1978). Practical reasoning mainly deals with the adoption, filling in, and reconsideration of intentions and plans. Moreover, it allows reasoning about individual actions using for instance the well-known *practical syllogism* (Walton 1996):

- G is a goal for agent a
- Doing action A is sufficient for agent a to carry out goal G
- Then, agent a ought to do action A

In this setting, a candidate action may be rejected because it may lead to the violation of other important goals, or to other bad consequences, etc.

In this paper, we are concerned with an argumentative counterpart of classical decision theory. Humans use arguments for supporting, attacking or explaining choices. Indeed, each potential choice has usually pros and cons of various strengths. Adopting such an approach in a decision support system would have some obvious benefits. On

the one hand, not only would the user be provided with a "good" choice, but also with the reasons underlying this recommendation, in a format that is easy to grasp. On the other hand, argumentation-based decision making is expected to be more akin with the way humans deliberate and finally make or understand a choice. This argumentative view of decision has not been much considered until recently even if the idea of basing decisions on arguments pros and cons is very old and was already somewhat formally stated by Benjamin Franklin (Franklin 1887) more than two hundred years ago.

Articulating decisions on the basis of arguments is relevant for different decision problems or approaches such as decision under uncertainty, multiple criteria decisions, or rule-based decisions. For instance, in medical domains, decisions are usually to be made under incomplete or uncertain information, and the potential results of candidate decisions may be evaluated from different criteria. Moreover, there may exist some expertise under the form of decision rules that associate possible decisions with given contexts. Thus, the different types of decision problems interfere, and consequently a unified argumentation-based model may be still more worth developing.

Whatever the decision problem is, the basic idea is that candidate decisions may lead to positively or negatively assessed results. This gives birth to arguments in favor of (pros) or against (cons) a decision in a given context. Different attempts at formalizing argument-based decision making can be found in the literature (Fox & Das 2000; Fox, Krause, & Ambler 1992; Bonet & Geffner 1996; Brewka & Gordon 1994; Amgoud & Prade 2004; Dubois & Fargier 2005; Amgoud, Bonnefon, & Prade 2005). These works do not much discuss the nature of arguments in a decision analysis, and usually rely on one type of argument that may be in favor of or against alternatives.

This paper emphasizes the *bipolar* nature of the evaluation of decision results, by making an explicit distinction between *goals* having a positive flavor, and rejections, with a negative flavor, that are stumbling blocks to be avoided. This, for instance, applies to criteria scales where the positive grades (associated with positive results) are separated from the negative ones (associated with negative results) by one or several neutral values.

The paper proposes a systematic typology that identifies eight types of arguments. Some of them are weaker than others, since they rather reflect the existence of examples or counter-examples as supporting or challenging possible choices. In the proposed framework, each decision is *supported* by arguments emphasizing its positive features in terms of goals certainly satisfied, goals possibly satisfied, rejections certainly avoided and rejections possibly avoided. The possibility that a goal may be reached or that a rejection may be avoided is assessed in practice by the existence of relevant known examples. A decision can also be *attacked* by arguments emphasizing its negative features in terms of certainly or possibly missed goals, or rejections certainly or possibly led to by that decision. The richness of the proposed typology makes it possible to partition the set of alternatives into four classes, giving thus a *status* to decisions, which may be *recommended*, *discommended*, *controversial* or *neutral*. Each class may be refined into sub-classes taking advantage of the strengths of the different types of arguments.

The aim of this paper is also to present a general discussion and a first study of the different classes of argument-based decision principles. In the following, we argue that three main classes of principles can be distinguished:

1. *Unipolar principles* that focus only on one type of arguments when comparing choices (either arguments pros or arguments cons)
2. *Bipolar principles* that take into account both types of arguments but still keeping the distinction between the two types
3. *Non-polar principles* that consist of aggregating the two types of arguments into a meta-argument and compare pairs of choices on the basis of their meta-arguments.

Note that, the use of suffix "polar" here refers to the dichotomy between arguments pros and arguments cons (and not to the bipolar structure induced by goals and rejections).

A general framework for decision making

Solving a decision problem amounts to defining a pre-ordering, usually a complete one, on a set $\mathcal{D}$ of possible choices (or decisions), on the basis of the different consequences of each decision. Argumentation can be used for defining such a pre-ordering. An argumentation-based decision process can be decomposed into the following steps:

1. Constructing arguments in *favor*/*against* each decision in $\mathcal{D}$.
2. Evaluating the strength of each argument.
3. Comparing decisions on the basis of their arguments.
4. Defining a pre-ordering on $\mathcal{D}$.

In (Amgoud, Bonnefon, & Prade 2005), an argumentation-based decision framework is defined as follows:

Definition 1 (Argumentation-based decision framework) *An* argumentation-based decision framework *is a tuple* $<\mathcal{D}, \mathcal{A}, \succeq, \triangleright_{Princ}>$ *where:*

- $\mathcal{D}$ *is a set of all possible choices.*
- $\mathcal{A}$ *is a set of arguments.*
- $\succeq$ *is a (partial or complete) pre-ordering on* $\mathcal{A}$*.*
- $\triangleright_{Princ}$ *(for principle for comparing decisions), defines a (partial or complete) pre-ordering on* $\mathcal{D}$*, defined on the basis of* $\mathcal{A}$*.*

The output *of the framework is a (complete or partial) pre-ordering* $\triangleright_{Princ}$*, on* $\mathcal{D}$*.* $d_1 \triangleright_{Princ} d_2$ *means that the decision* d_1 *is* at least as preferred as *the decision* d_2 *w.r.t. the* principle $Princ$*.*

Notation: Let A, B be two arguments of $\mathcal{A}$, and $\succeq$ be a preorder (maybe partial) on $\mathcal{A}$. $A \succeq B$ means that A is at least as 'strong' as B.
$\succ$ and $\approx$ will denote respectively the strict ordering and the relation of equivalence associated with the preference between arguments, defined as follows:

- $A \succ B$ iff $A \succeq B$ and not ($B \succeq A$) (meaning that A is strictly stronger than B),
- $A \approx B$ iff $A \succeq B$ and $B \succeq A$ (meaning that A is as strong as B).

Different definitions of $\succeq$ or different definitions of $\triangleright_{Princ}$ may lead to different decision frameworks that may not return the same results.

Logical language

In what follows, let $\mathcal{L}$ be a propositional language. From $\mathcal{L}$ we can distinguish the four following sets:

1. The set $\mathcal{D}$ gathers all the possible *alternatives*, or *decisions*. These candidate actions are assumed to be feasible. Elements of $\mathcal{D}$ are supposed to be represented by literals.
2. The set $\mathcal{K}$ represents the *background knowledge* that is assumed to be consistent. Elements of $\mathcal{K}$ are formulas of $\mathcal{L}$.
3. The set $\mathcal{G}$ gathers the *goals* of an agent. A goal represents what the agent wants to achieve, and has thus a positive flavor. This base is assumed to be consistent too, i.e. an agent is not allowed to have contradictory goals. Note that a goal may be expressed in terms of a logical combination of constraints on criteria values, and does not necessarily refer to one criterion. Elements of $\mathcal{G}$ are supposed to be literals.
4. The set $\mathcal{R}$ gathers the *rejections* of an agent. A rejection represents what the agent wants to avoid. Clearly rejections express negative preferences. The set $\{\neg r | r \in \mathcal{R}\}$ is assumed to be consistent since acceptable alternatives should satisfy $\neg r$ due to the rejection of r. However, note that if r is a rejection, this does not necessarily mean that $\neg r$ is a goal. For instance, in case of choosing a medical drug, one may have as a goal the immediate availability of the drug, and as a rejection its availability only after at least two days. As it can be guessed on this example, if g is a goal only r such that $r \vdash \neg g$ can be a rejection, and conversely. This means that rejection can be more specific than the negation of goals. Moreover, recent cognitive psychology studies (Cacioppo, Gardner, & Bernston 1997) have confirmed the cognitive validity of this distinction between goals and rejections. Elements of $\mathcal{R}$ are supposed to be literals.

Definition 2 *A* decision problem *is a tuple* $\mathcal{T} = \langle \mathcal{D}, \mathcal{K}, \mathcal{G}, \mathcal{R} \rangle$.

A new typology of arguments

When solving a decision problem, there may exist several alternative solutions. Each alternative may have arguments in its favor (called PROS), and arguments against it (called CONS). In the following, an argument is associated with an alternative, and always either refers to a goal or to a rejection.

Arguments PROS point out the existence of good consequences or the absence of bad consequences for a given alternative. More precisely, we can distinguish between two types of good consequences, namely the guaranteed satisfaction of a goal when $\mathcal{K} \cup \{d\} \vdash g$, and the possible satisfaction of a goal when $\mathcal{K} \cup \{d\} \nvdash \neg g$, with $d \in \mathcal{D}$ and $g \in \mathcal{G}$. Note that this latter situation corresponds to the existence of an interpretation that satisfies $\mathcal{K}$, d, and g. This leads to the following definition:

Definition 3 (Types of positive arguments PRO) *Let $\mathcal{T}$ be a decision problem. A* positively expressed argument in favor of *an alternative d is a pair $A = \langle d, g \rangle$ such that:*

1. *$d \in \mathcal{D}$, $g \in \mathcal{G}$, $\mathcal{K} \cup \{d\}$ is consistent*
2. - *$\mathcal{K} \cup \{d\} \vdash g$ (arguments of Type SPP), or*
   - *$\mathcal{K} \cup \{d\} \nvdash \neg g$ (arguments of Type WPP)*

The consistency of $\mathcal{K} \cup \{d\}$ means that d is applicable in the context $\mathcal{K}$, in other words that we cannot prove from $\mathcal{K}$ that d is impossible. This means that impossible alternatives w.r.t. $\mathcal{K}$ have been already taken out when defining the set $\mathcal{D}$.
Note that SPP arguments are stronger than WPP ones since ($\mathcal{K} \cup \{d\} \vdash g$) entails ($\mathcal{K} \cup \{d\} \nvdash \neg g$). SPP stands for "Strong Positive PROS", whereas WPP is short for "Weak Positive PROS". Clearly, WPP arguments have interest only when the corresponding SPP do not exist. From a practical point of view, SPP arguments will be generally expressed by only exhibiting a minimal subset S of $\mathcal{K}$ such that $S \cup \{d\} \vdash g$, while a WPP argument corresponds to the existence of (at least) one known case where g was satisfied while d was applied. This means that together with $\mathcal{K}$, the agent stores a case memory of already experienced choices. Let $Arg_{SPP}(d)$ (resp. $Arg_{WPP}(d)$) be the set of all arguments of type SPP (resp. WPP) in favor of d in the sense of the above definition. The following inclusion holds:

Property 1 *Let $d \in \mathcal{D}$. Then $Arg_{SPP}(d) \subseteq Arg_{WPP}(d)$.*

Similarly, there are two forms of absence of bad consequences that lead to arguments PROS: the first one amounts to avoid a rejection for sure, i.e. $\mathcal{K} \cup \{d\} \vdash \neg r$, and the second form corresponds only to the possibility of avoiding a rejection ($\mathcal{K} \cup \{d\} \nvdash r$), which can be testified in practice by the existence of a case counter-example assuring that $\mathcal{K} \wedge d \wedge \neg r$ is consistent (with $r \in \mathcal{R}$). This leads to the following definition:

Definition 4 (Types of negative arguments PROS) *Let $\mathcal{T}$ be a decision problem. A* negatively expressed argument in favor of *an alternative is a pair $A = \langle d, r \rangle$ such that:*

1. *$d \in \mathcal{D}$, $r \in \mathcal{R}$, $\mathcal{K} \cup \{d\}$ is consistent*
2. - *$\mathcal{K} \cup \{d\} \vdash \neg r$ (arguments of Type SNP), or*
   - *$\mathcal{K} \cup \{d\} \nvdash r$ (arguments of Type WNP)*

Here again, SNP arguments are stronger than WNP ones since $\mathcal{K} \cup \{d\} \vdash \neg r$ entails $\mathcal{K} \cup \{d\} \nvdash r$. SNP stands for

"Strong Negative PROS", while WNP means "Weak Negative PROS". Let $Arg_{SNP}(d)$ (resp. $Arg_{WNP}(d)$) be the set of all arguments of type SNP (resp. WNP) in favor of d.

Property 2 *Let $d \in \mathcal{D}$. Then $Arg_{SNP}(d) \subseteq Arg_{WNP}(d)$.*

Arguments CONS highlight the existence of bad consequences for a given alternative, or the absence of good ones. As in the case of arguments PROS, there are a strong form and a weak form of both situations. Namely, negatively expressed arguments CONS are defined either by exhibiting a rejection that is necessarily satisfied, or a rejection that is possibly satisfied. Formally:

Definition 5 (Types of negative arguments CONS) *Let $\mathcal{T}$ be a decision problem. A* negatively expressed argument against *an alternative d is a pair $A = \langle d, g\rangle$ such that:*

1. *$d \in \mathcal{D}$, $r \in \mathcal{R}$, $\mathcal{K} \cup \{d\}$ is consistent*
2. • *$\mathcal{K} \cup \{d\} \vdash r$ (arguments of Type SNC), or*
   • *$\mathcal{K} \cup \{d\} \nvdash \neg r$ (arguments of Type WNC)*

Let $Arg_{SNC}(d)$ (resp. $Arg_{WNC}(d)$) be the set of all arguments of type SNC (resp. WNC) against d, where C stands for Cons.

Property 3 *Let $d \in \mathcal{D}$. Then $Arg_{SNC}(d) \subseteq Arg_{WNC}(d)$.*

Lastly, the absence of positive consequences can also be seen as an argument against (CONS) an alternative. A strong form and a weak form of positively expressed arguments against an alternative can be defined as follows:

Definition 6 (Types of arguments CONS) *Let $\mathcal{T}$ be a decision problem. A* positively expressed argument against *an alternative is a pair $A = \langle d, g\rangle$ such that:*

1. *$d \in \mathcal{D}$, $g \in \mathcal{G}$, $\mathcal{K} \cup \{d\}$ is consistent*
2. • *$\mathcal{K} \cup \{d\} \vdash \neg g$ (arguments of Type SPC), or*
   • *$\mathcal{K} \cup \{d\} \nvdash g$ (arguments of Type WPC)*

Let $Arg_{SPC}(d)$ (resp. $Arg_{WPC}(d)$) be the set of all arguments of type SPC (resp. WPC) against d.

Property 4 *Let $d \in \mathcal{D}$. Then $Arg_{SPC}(d) \subseteq Arg_{WPC}(d)$.*

Let us consider positively expressed arguments for instance. Observe that for a given alternative d and a fixed goal g, all the types of arguments cannot take place at the same time. Formally,

Property 5 *SPP and WPC (resp. SPC and WPP, and SPP and SPC) arguments are mutually exclusive.*

In the first two cases, this is due to the opposite characteristic conditions of the definitions. The last exclusion is due to the consistency of $\mathcal{K} \cup \{d\}$, and thus g and $\neg g$ cannot be obtained simultaneously. Taking into account the subsumptions between weak and strong forms of arguments, the following result holds:

Property 6 *Let $d \in \mathcal{D}$ and $g \in \mathcal{G}$. There are only three possible situations w.r.t a positively expressed argument linking d and g, namely: i) there is an SPP argument, ii) there is an SPC argument, iii) there are both an WPP and an WPC arguments.*

The above property reflects the three possible epistemic statuses of a knowledge base ($\mathcal{K} \cup \{d\}$) w.r.t a proposition (here g), which may be true, false, or having an unknown truth status. In the latter case, emphasizing either a WPP argument, or a WPC argument is a matter of optimism vs pessimism to which we come back later. Since WPP and WPC arguments are somewhat neutralizing each other, we will not consider them in the decision status classification that we introduce now.

Decision status

In the previous section, we have shown that each decision may be supported by two types of strong arguments, and attacked by two other types of strong arguments. In summary, given a decision $d \in \mathcal{D}$, we will have the following sets of arguments:

- $Arg_{SPP}(d)$ = those arguments which capture the goals that are reached when applying d
- $Arg_{SNP}(d)$ = those arguments which capture the rejections that are avoided when applying d
- $Arg_{SNC}(d)$ = those arguments which capture the rejections that are not avoided when applying d
- $Arg_{SPC}(d)$ = those arguments which capture the goals that are missed when applying d

Note that when a given set is empty, for instance $Arg_{SPP}(d) = \emptyset$, this does not mean at all that decision d cannot lead to any goal, but rather we cannot be certain that a goal is reached as some information is missing. The above types of arguments supporting or attacking a choice d give birth to four main different *statuses* for that decision: *recommended*, *discommended*, *neutral* and *controversial* (see Table 1 for the straightforward formal definitions). Recommended choices are those choices that have only arguments in favor of them and no arguments against them (whatever their type). Discommended choices are those choices that have no arguments in favor of them and only arguments against them. Regarding neutral choices, they have neither arguments in favor of them, nor arguments against. Choices that have at the same time arguments in favor of them and arguments against are said controversial. As shown in Table 1, there are 9 situations in which a choice is controversial.

Property 7 *Let $d \in \mathcal{D}$. Then d is either fully recommended, or fully discommended, or controversial or neutral.*

Note that one may give priority to SPP and SNC arguments, which directly state that a goal is reached, or a rejection is not missed respectively. SNP and SPC arguments that are in favor of or against a choice only indirectly can be used for refining the two first types of arguments. For instance, in Table 1, a choice that has both SPP and SNP arguments in favor of it is strongly recommended, whereas a choice with only SPP arguments in favor of it is only recommended.

This classification of choices is of interest from a persuasion or explanation perspective, since e.g recommended choices are more easily arguable than controversial ones. This does not mean that recommended choices are always better than any other as we shall see in the next section.

Status	Sub-status	Combination
Recommended	Strongly recom.	$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
	Recom.	$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
	Weakly recom.	$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
Discommended	Strongly discom.	$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
	Discom.	$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
	Weakly discom.	$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
Neutral		$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
Controversial		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) = \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) \neq \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) \neq \emptyset, Arg_{SNP}(d) = \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$
		$\langle Arg_{SPP}(d) = \emptyset, Arg_{SNP}(d) \neq \emptyset, Arg_{SNC}(d) = \emptyset, Arg_{SPC}(d) \neq \emptyset \rangle$

Table 1: Decision status

Principles for comparing decisions

In the following we are interested in discussing possible choices for $\rhd_{Princ}$. The relation $\succeq$ is assumed to be given. It may be either a partial or a complete preorder. This preorder may account for the certainty of the pieces of knowledge involved in the argument and/or to the importance of the goal to which the argument pertains. However, this will not be detailed in the following.

Comparing choices on the basis of the sets of PROS or CONS arguments that are associated with them is a key step in an argumentative decision process. Depending on what sets are considered and how they are handled, one can roughly distinguish between three categories of principles:

Unipolar principles: are those that only refer to either the arguments PROS or the arguments CONS.

Bipolar principles: are those that reason on both types of arguments at the same time.

Non-polar principles: are those where arguments PROS and arguments CONS a given choice are aggregated into a unique *meta-argument*. It results that the negative and positive polarities disappear in the aggregation.

Below we present the main principles that can be thought of for each category. In what follows, $Arg_{Pro}(d) = Arg_{SPP}(d) \cup Arg_{SNP}(d)$, and $Arg_{Cons}(d) = Arg_{SNC}(d) \cup Arg_{SPC}(d)$. Moreover, the function `Result` returns for a given set of arguments, all the goals/rejections involved in those arguments.

Unipolar principles

In this section we present basic criteria for comparing decisions on the basis of only arguments PROS. Note that similar ideas apply to arguments CONS.

We start by presenting those criteria that do not involve the strength of arguments, then their respective refinements when strength is taken into account.

A first natural criterion consists of preferring the decision d_1 over d_2 if for each argument $<d_2, g>$, there exists an argument $<d_1, g>$, while the reverse is not true. Formally:

Definition 7 *Let* $d_1, d_2 \in \mathcal{D}$.
$d_1 \rhd d_2$ *iff* $\texttt{Result}(Arg_{Pro}(d_2)) \subseteq \texttt{Result}(Arg_{Pro}(d_1))$.

This *partial* preorder is refined by the following *complete* preorder in terms of cardinality, i.e preferring the decision that has more arguments PROS.

Definition 8 (Counting arguments PROS) *Let* $d_1, d_2 \in \mathcal{D}$.
$d_1 \rhd d_2$ *iff* $|Arg_{Pro}(d_1)| \geq |Arg_{Pro}(d_2)|$.

When the strength of arguments is taken into account in the decision process, one may think of preferring a choice that has a dominant argument, i.e. an argument PROS that is preferred to any argument PROS the other choices.

Definition 9 *Let* $d_1, d_2 \in \mathcal{D}$.
$d_1 \rhd d_2$ *iff* $\exists\ A \in Arg_{Pro}(d_1)$ *such that* $\forall\ B \in Arg_{Pro}(d_2)$, $A \succeq B$.

The above definition relies heavily on the relation $\succeq$ that compares arguments. Thus, the properties of this criterion depends on those of $\succeq$. Namely, it can be checked that the above criterion works properly only if $\succeq$ is a complete preorder.

Property 8 *If the relation* $\succeq$ *is a complete preorder, then* $\rhd$ *is also a complete preorder.*

Note that the above relation may be found to be too restrictive, since when the strongest arguments in favor of d_1 and d_2 have equivalent strengths (in the sense of $\approx$), d_1 and d_2 are also seen as equivalent. However, we can refine the above definition by ignoring the strongest arguments with equal strengths, by means of the following *strict preorder*.

Definition 10 *Let $d_1, d_2 \in \mathcal{D}$, and $\succeq$ a complete preorder. Let ($P_1, \ldots, P_r$), ($P'_1, \ldots, P'_s$) be the vectors of arguments PROS the decisions d_1 and d_2 respectively. Each of these vectors is assumed to be decreasingly ordered w.r.t $\succeq$ (e.g. $P_1 \succeq \ldots \succeq P_r$). Let v = min(r, s).*
$d_1 \rhd d_2$ *iff:*

- $P_1 \succ P'_1$, *or*
- $\exists\, k \leq v$ *such that* $P_k \succ P'_k$ *and* $\forall\, j < k$, $P_j \approx P'_j$, *or*
- $r > v$ *and* $\forall\, j \leq v$, $P_j \approx P'_j$.

Note that in all the above criteria, the two types of arguments PROS are considered as having the same importance. Thus, reasoning with goals is as important as reasoning with rejections. However, this may be debatable, since one may prefer arguments ensuring that a goal is reached to an argument that shows that a rejection is avoided, since the latter is the least thing that can be expected. On the basis of this new source of preference between arguments, the above criteria can be further reformulated by processing separately the two sets of arguments. More precisely, we can apply the above definitions only for the set of arguments of type SPP, and only in case of ties to apply again the same definitions on SNP arguments. We may even a different criterion on the set SNP of arguments.

Another point that is worth discussing is the impact of possible dependencies between goals (or rejections) on the decision criteria. Namely, assume that two goals are, for instance, redundant, i.e they are logically equivalent giving $\mathcal{K}$. Applying the cardinality-based criterion may lead to privilege decisions reaching redundant goals. This maybe debatable although allowing for redundancy is clearly a way of stressing the importance of a goal (or a rejection). Note that not all the above criteria are sensible to redundancy, for instance the first one (Definition 7).

Till now, we have only discussed decision criteria based on arguments PROS. However, the counterpart criteria when arguments CONS are considered can also be defined. Thus, the counterpart criterion of the one defined in Definition 7 is the following partial preorder:

Definition 11 *Let $d_1, d_2 \in \mathcal{D}$. $d_1 \rhd d_2$ iff* $\texttt{Result}(Arg_{Cons}(d_1)) \subseteq \texttt{Result}(Arg_{Cons}(d_2))$.

Similarly, it refinement in terms of cardinality is given by the following complete preorder:

Definition 12 (Counting arguments CONS) *Let $d_1, d_2 \in \mathcal{D}$.*
$d_1 \rhd d_2$ *iff* $|Arg_{Cons}(d_1)| \leq |Arg_{Cons}(d_2)|$.

The criteria that take into account the strengths of arguments have also their counterparts when handling arguments CONS.

Definition 13 *Let $d_1, d_2 \in \mathcal{D}$.*
$d_1 \rhd d_2$ *iff* $\exists\ B \in Arg_{Cons}(d_2)$ *such that* $\forall\ A \in Arg_{Cons}(d_1)$, $B \succeq A$.

As in the case of arguments PROS, when the relation $\succeq$ is a complete preorder, the above relation is also a complete preorder, and can be refined into the following strict one.

Definition 14 *Let $d_1, d_2 \in \mathcal{D}$.*
Let ($C_1, \ldots, C_r$), ($C'_1, \ldots, C'_s$) be the vectors of arguments CONS the decisions d_1 and d_2. Each of these vectors is assumed to be decreasingly ordered w.r.t $\succeq$ (e.g. $C_1 \succeq \ldots \succeq C_r$). Let v = min(r, s).
$d_1 \succ d_2$ *iff:*

- $C'_1 \succ C_1$, *or*
- $\exists\, k \leq v$ *such that* $C'_k \succ C_k$ *and* $\forall\, j < k$, $C_j \approx C'_j$, *or*
- $v < s$ *and* $\forall\, j \leq v$, $C_j \approx C'_j$.

Finally, it may be also worth distinguishing between SNC and SPC arguments, and to privilege those which are SNC since they are the most striking ones. Similar ideas given in the case of arguments PROS apply.

Bipolar principles

Let's now define some principles where both types of arguments (PROS and CONS) are taken in account when comparing decisions. Generally speaking, we can conjunctively combine the criteria dealing with arguments PROS with their counterpart handling arguments CONS. For instance, the criterion given in Definition 8 can be combined with that given in Definition 12 into the following one:

Definition 15 *Let $d_1, d_2 \in \mathcal{D}$.*
$d_1 \rhd d_2$ *iff*

1. $|Arg_{Pro}(d_1)| \geq |Arg_{Pro}(d_2)|$, *and*
2. $|Arg_{Cons}(d_1)| \leq |Arg_{Cons}(d_2)|$.

However, note that unfortunately this is no longer a complete preorder. Similarly, the criteria given respectively in Definition 9 and Definition 13 can be combined into the following one:

Definition 16 *Let $d_1, d_2 \in \mathcal{D}$.*
$d_1 \rhd d_2$ *iff:*

- $\exists\ A \in Arg_{pros}(d_1)$ *such that* $\forall\ B \in Arg_{pros}(d_2)$, $A \succeq B$.
- $\nexists\ A' \in Arg_{Cons}(d_1)$ *such that* $\forall\ B' \in Arg_{Cons}(d_2)$, $A \succeq B$.

This means that one prefers a decision which has at least one supporting argument which is better than any supporting argument of the other decision, and also which has not a very strong argument against it.
Note that the above definition can be also refined in the same spirit as Definitions 10 and 14.

Another family of bipolar decision criteria applies the *Franklin principle* which is a natural extension to the bipolar case of the idea underlying Definition 10. This criterion consists, when comparing pros and CONS a decision, of ignoring pairs of arguments pros and CONS which have the same strength. After such a simplification, one can apply any of the above bipolar principles. In what follows, we will define formally the Franklin simplification.

Definition 17 (Franklin simplification) *Let $d \in \mathcal{D}$.*
Let P = ($P_1, \ldots, P_r$), ($C = C_1, \ldots, C_m$) be the vectors of the arguments PROS and CONS the decision d. Each of these vectors is assumed to be decreasingly ordered w.r.t $\succeq$

(e.g. $P_1 \succeq \ldots \succeq P_r$).
The result of the simplification *is $P' = (P_{j+1}, \ldots, P_r)$, $C' = (C_{j+1}, \ldots, C_m)$ such that:*

- *$\forall\, 1 \leq i \leq j$, $P_i \approx C_i$ and $(P_{j+1} \succ C_{j+1}$ or $C_{j+1} \succ P_{j+1})$*
- *If $j = r$ (resp. $j = m$), then $P' = \emptyset$ (resp. $C' = \emptyset$).*

Non-polar principles

In some applications, the arguments in favor of and against a decision are aggregated into a unique *meta-argument* having a unique strength. Thus, comparing two decisions amounts to compare the resulting meta-arguments. Such a view is well in agreement with current practice in multiple principles decision making, where each decision is evaluated according to different principles using the same scale (with a positive and a negative part), and an aggregation function is used to obtain a global evaluation of each decision.

Definition 18 (Aggregation criterion) *Let $d_1, d_2 \in \mathcal{D}$. Let $<P_1, \ldots, P_n>$ and $<C_1, \ldots, C_m>$ (resp. $<P_1', \ldots, P_l'>$ and $<C_1', \ldots, C_k'>$) the vectors of the arguments PROS and CONS the decision d_1 (resp. d_2).*
$d_1 \triangleright d_2$ iff $h(P_1, \ldots, P_n, C_1, \ldots, C_m) \succeq h(P_1', \ldots, P_l', C_1', \ldots, C_k')$, where h is an aggregation function.

A simple example of this aggregation attitude is computing the difference of the number of arguments PROS and CONS.

Definition 19 *Let $d_1, d_2 \in \mathcal{D}$.*
$d_1 \triangleright d_2$ iff $|Arg_{Pros}(d_1)| - |Arg_{Cons}(d_1)| \geq |Arg_{Pros}(d_2)|$ - $|Arg_{Cons}(d_2)|$.

This has the advantage to be again a complete preorder, while taking into account both PROS and CONS arguments.

Conclusion

The paper has proposed an argumentation-based framework for decision making. The framework emphasizes clearly the bipolar nature of the consequences of choices by distinguishing goals to be pursued from rejections to be avoided. This bipolar setting gives birth to two kinds of arguments for each choice: arguments in favor of that choice and arguments against it. Moreover, we have shown that there are four types of arguments PROS a choice (resp. against a choice), and some of them are stronger than others. The different types of arguments allow us to give a unique status to each choice (recommended, discommended, neutral or controversial). We have also proposed different criteria for comparing pairs of choices. The proposed approach is very general and includes as particular cases already studied argumentation-based decision principles (Fox & Das 2000; Amgoud & Prade 2004; Dubois & Fargier 2005). Besides, the richness of the different possible behaviors when arguing a decision in this framework should be compared to the actual practice of humans as studied in cognitive psychology.

Acknowledgments

This work was supported by the Commission of the European Communities under contract IST-2004-002307, ASPIC project “Argumentation Service Platform with Integrated Components”.

References

Amgoud, L., and Prade, H. 2004. Using arguments for making decisions. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 10–17.

Amgoud, L.; Bonnefon, J.-F.; and Prade, H. 2005. An argumentation-based approach to multiple criteria decision. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'2005*, 269–280.

Bonet, B., and Geffner, H. 1996. Arguing for decisions: A qualitative model of decision making. In E. Horwitz, F. J. e., ed., *Proc. 12th Con. on Uncertainty in Artificial Intelligence (UAI'96)*, 98–105.

Bratman, M.; Israel, D.; and Pollack, M. 1988. *Plans and resource bounded reasoning.*, volume 4. Computational Intelligence.

Bratman, M. 1987. *Intentions, plans, and practical reason.* Harvard University Press, Massachusetts.

Brewka, G., and Gordon, T. 1994. How to buy a Porsche: An approach to defeasible decision making. In *Working Notes of the AAAI-94 Workshop on Computational Dialectics*, 28–38.

Cacioppo, J.; Gardner, W.; and Bernston, G. 1997. Beyond bipolar conceptualizations and measures: The case of attitudes and evaluative space. *Personality and Social Psychology Review* 1, 1:3–25.

Dubois, D., and Fargier, H. 2005. On the qualitative comparison of sets of positive and negative affects. In *Proceedings of ECSQARU'05*.

Fox, J., and Das, S. 2000. *Safe and Sound. Artificial Intelligence in Hazardous Applications*. AAAI Press, The MIT Press.

Fox, J.; Krause, P.; and Ambler, S. 1992. Arguments, contradictions and practical reasoning. In *Proceedings of the 10th European Conference on AI, ECAI'92*.

Franklin, B. 1887. *Letter to J. B. Priestley, 1772, in The Complete Works, J. Bigelow, ed.* New York: Putnam.

Raz, J. 1978. Practical reasoning. *Oxford, Oxford University Press*.

Savage, L. 1954. *The Foundations of Statistics*. New York: Dover. Reprinted by Dover, 1972.

Walton, D. 1996. *Argument schemes for presumptive reasoning*, volume 29. Lawrence Erlbaum Associates, Mahwah, NJ, USA.

5.8 Defeasible Reasoning about Beliefs and Desires

Defeasible Reasoning About Beliefs and Desires

Nicolás D. Rotstein and **Alejandro J. García**
Artificial Intelligence Research and Development Laboratory,
Department of Computer Science and Engineering, Universidad Nacional del Sur,
Email: {ndr,ajg}@cs.uns.edu.ar

Abstract

In this paper we show how a deliberative agent can represent beliefs and desires and perform defeasible reasoning in order to support its derived beliefs. Strict and defeasible filtering rules can be used by the agent for selecting among its desires a proper one that fits in the particular situation it is involved. Thus, defeasible argumentation will be used for reasoning about desires. Application examples from a robotic soccer domain will be given.

Introduction

This article addresses the problem of having a deliberative intelligent agent built upon an architecture that relies on sets of Beliefs and Desires (*e.g.*, the BDI architecture (Bratman, Israel, & Pollack 1991; Rao & Georgeff 1995; Rao 1996)). The proposed approach allows the agent to perform defeasible reasoning in order to support its derived beliefs and offers a defeasible argumentation framework for selecting desires.

In our approach, the agent will represent information that it perceives directly from its environment, and in addition to these *perceived beliefs*, the agent may represent other knowledge in the form of strict and defeasible rules. Then, using a defeasible argumentation formalism it will be able to to obtain a *warrant* for its *derived beliefs*.

We will introduce a reasoning formalism for selecting those desires that are suitable to be carried out by the agent. In order to perform this selection, the agent will use its beliefs (that represent the current situation) and a defeasible logic program composed by *filtering rules*.

Warranting agent's beliefs and perception

In this approach, agent's beliefs will correspond to the semantics[1] of a *defeasible logic program* $\mathcal{P}_\mathsf{B}$ = (Π_B,Δ_B). In this kind of programs (García & Simari 2004) two different sets are distinguished: the set Δ_B for representing tentative knowledge in the form of defeasible rules; and the set Π_B for representing non-tentative, sound knowledge in the form of strict rules and facts. The information that the agent perceives directly from its environment is represented in Π_B with a subset of facts denoted Φ. Thus, in the set Π_B two disjoint subsets will be distinguished: the subset Φ of *perceived beliefs* that will be updated dynamically, and a subset Π formed with strict rules and facts that will represent static knowledge. Therefore, Π_B= $\Phi \cup \Pi$. As we will explain below, in order for program $\mathcal{P}_\mathsf{B}$ to behave correctly, some restrictions over Π_B will be imposed.

In addition to the perceived beliefs, the agent may use strict and defeasible rules from $\mathcal{P}_\mathsf{B}$ in order to obtain a *warrant* for its *derived beliefs* (see Definition 1). A brief explanation of how warrants are obtained using the defeasible argumentation formalism of DeLP, and the definitions of facts, strict and defeasible rules are included below (the interested reader is referred to (García & Simari 2004) for a detailed explanation).

Representing knowledge and reasoning with DeLP

In DeLP, knowledge is represented using facts, strict rules or defeasible rules:

- *Facts* are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$" (*e.g.*, $hasBall(opponent)$).
- *Strict Rules* are denoted $L_0 \leftarrow L_1, \ldots, L_n$, where the *head* L_0 is a ground literal and the *body* $\{L_i\}_{i>0}$ is a set of ground literals. (*e.g.*, $\sim hasBall(myTeam) \leftarrow hasBall(opponent)$).
- *Defeasible Rules* are denoted $L_0 \prec L_1, \ldots, L_n$, where the *head* L_0 is a ground literal and the *body* $\{L_i\}_{i>0}$ is a set of ground literals. (*e.g.*, $\sim pass(mate1) \prec marked(mate1)$).

Syntactically, the symbol "$\prec$" is all that distinguishes a defeasible rule from a strict one. Pragmatically, a defeasible rule is used to represent defeasible knowledge, *i.e.*, tentative information that may be used if nothing could be posed against it. A defeasible rule "$Head \prec Body$" is understood as expressing that "*reasons to believe in the antecedent* $Body$ *provide reasons to believe in the consequent* $Head$" (Simari & Loui 1992).

A Defeasible Logic Program $\mathcal{P}$ is a set of facts, strict rules and defeasible rules. When required, $\mathcal{P}$ is denoted (Π, Δ) distinguishing the subset Π of facts and strict rules, and the subset Δ of defeasible rules. Observe that strict

[1] Since the semantics of DeLP is skeptical, there is only one.

and defeasible rules are ground. However, following the usual convention (Lifschitz 1996), some examples will use "schematic rules" with variables. Given a "schematic rule" R, $Ground(R)$ stands for the set of all ground instances of R. Given a program $\mathcal{P}$ with schematic rules, we define (Lifschitz 1996): $Ground(\mathcal{P}) = \bigcup_{R \in \mathcal{P}} Ground(R)$. In order to distinguish variables, they are denoted with an initial uppercase letter.

Strong negation is allowed in the head of program rules, and hence may be used to represent contradictory knowledge. From a program (Π, Δ) contradictory literals could be derived, however, the set Π (which is used to represent non-defeasible information) must possess certain internal coherence. Therefore, Π has to be non-contradictory, *i.e.*, no pair of contradictory literals can be derived from Π. Given a literal L the complement with respect to strong negation will be denoted $\overline{L}$ (*i.e.*, $\overline{a}$=$\sim a$ and $\overline{\sim a}$=a).

DeLP incorporates an argumentation formalism for the treatment of the contradictory knowledge that can be derived from (Π, Δ). This formalism allows the identification of the pieces of knowledge that are in contradiction. A dialectical process is used for deciding which information prevails. In particular, the argumentation-based definition of the inference relation makes it possible to incorporate a treatment of preferences in an elegant way.

In DeLP a literal L is *warranted* from (Π, Δ) if there exists a non-defeated argument $\mathcal{A}$ supporting L. In short, an *argument* for a literal L, denoted $\langle \mathcal{A}, L \rangle$, is a minimal set of defeasible rules $\mathcal{A} \subseteq \Delta$, such that $\mathcal{A} \cup \Pi$ is non-contradictory and there is a derivation for L from $\mathcal{A} \cup \Pi$. In order to establish if $\langle \mathcal{A}, L \rangle$ is a non-defeated argument, *argument rebuttals* or *counter-arguments* that could be *defeaters* for $\langle \mathcal{A}, L \rangle$ are considered, *i.e.*, counter-arguments that by some criterion are preferred to $\langle \mathcal{A}, L \rangle$. Since counter-arguments are arguments, there may exist defeaters for them, and defeaters for these defeaters, and so on. Thus, a sequence of arguments called *argumentation line* is constructed, where each argument defeats its predecessor in the line (for a detailed explanation of this dialectical process see (García & Simari 2004)).

In DeLP, given a query Q there are four possible answers[2]: YES, if Q is warranted; *no*, if the complement of Q is warranted; *undecided*, if neither Q nor its complement is warranted; and *unknown*, if Q is not in the language of the program.

Perceived and Warranted Beliefs

As stated above, agent's beliefs will correspond to the semantics of a defeasible logic program (Π_{B},Δ_{B}), and the set Π_{B} will represent perceived and static information (Π_{B}= $\Phi \cup \Pi$). Since Π_{B} has to be non-contradictory, some restrictions about perception are imposed:

1. We assume that perception is correct in the sense that it will not give a pair of contradictory literals. Whenever this happens both literals will be ignored.
2. We will also require that no perceived literal in Φ can be derived directly from Π.

Thus, if Π is non-contradictory and these two restrictions are satisfied, then Π_{B} will also be non-contradictory. The next definition introduces the different types of belief that an agent will obtain from a defeasible logic program (Π_{B},Δ_{B}).

Definition 1 (belief types) *A **perceived belief** is a fact in Φ that represents information that the agent has perceived directly from its environment. A **strict belief** is a literal that is not a perceived belief, and it is derived from $\Pi_{\mathsf{B}} = \Pi \cup \Phi$ (i.e., no defeasible rules are used for its derivation). A **defeasible belief** is a warranted literal L supported by an argument $\langle \mathcal{A}, L \rangle$ that uses at least one defeasible rule (i.e., $\mathcal{A} \neq \emptyset$). Finally, a **derived belief** is a strict or a defeasible belief. We will denote with B_s the set of strict beliefs, and with B_d the set of defeasible beliefs. Therefore, in any given situation the beliefs of an agent will be $\mathsf{B} = \Phi \cup \mathsf{B}_s \cup \mathsf{B}_d$.*

Observe that the sets Φ, B_s and B_d are disjoint sets. Observe also that, although perceived beliefs are facts in Π_{B}, there can be other facts in Π_{B} that are not perceived. For instance, facts that represent agent's features, roles, etc. These facts that do not represent perceived information are persistent in the sense that they will not change with perception. For example: $myRole(defender)$, $opponent(o1)$.

In this approach we assume a perception function that provides the agent with information about its environment. This function will be invoked by the agent in order to update its perceived beliefs set Φ. When this happens the new information obtained must override the old one following some criterion. Updating a set of literals is a well-known problem and many solutions exist in the literature.

Example 1 *Consider an agent Ag that has the following program (Π_{B},Δ_{B}). Here, the set Π_{B} was divided distinguishing the subset Φ of perceived facts, and the subset Π of non-perceived information.*

$$\Phi = \left\{ \begin{array}{l} hasBall(t1) \\ marked(t1) \end{array} \right\}$$

$$\Pi = \left\{ \begin{array}{l} mate(t1) \\ opponent(o1) \\ \sim mate(X) \leftarrow opponent(X) \\ \sim receive(self) \leftarrow hasBall(self) \end{array} \right\}$$

$$\Delta_{\mathsf{B}} = \left\{ \begin{array}{l} receive(self) \prec hasBall(X), mate(X) \\ \sim receive(self) \prec marked(self) \\ \sim receive(self) \prec hasBall(X), \sim mate(X) \end{array} \right\}$$

In this example, Ag has the following perceived beliefs: $hasBall(t1)$ (the player $t1$ has the ball), and $marked(t1)$ (its teammate $t1$ is marked). Besides its perceived beliefs, it has two other facts that are strict beliefs: $mate(t1)$ ($t1$ is a teammate) and $opponent(o1)$ ($o1$ is an opponent). The set Π has also two strict rules representing that "an opponent is not a teammate" and that "Ag cannot receive the ball from itself". Observe that it can also infer the strict belief: $\sim mate(o1)$ (o1 is not a teammate).

The set of defeasible rules Δ_{B} represents that: "if a teammate has the ball, then Ag may receive a pass

[2]The implementation (interpreter) of DeLP that satisfies the semantics described in (García & Simari 2004) is currently accessible online at `http://lidia.cs.uns.edu.ar/DeLP`.

from it", "being marked is a good reason for not receiving a pass", and "if the one that has the ball is not a teammate, then there are good reasons for not receiving the ball from it". Thus, from (Π_{B},Δ_{B})*, the argument* $\{receive(self) \prec hasBall(t1), mate(t1)\}$ *has no defeaters, and therefore, there is a warrant for one defeasible belief:* $receive(self)$ *(Ag may receive a pass).*

Consider now that, upon perception, the set Φ *is updated to* Φ= $\{hasBall(t1), marked(t1), marked(self)\}$ *(i.e., the original situation has changed only in that the agent is now being marked); then, from this new program, the argument for* $receive(self)$ *has a "blocking defeater", what means that the DeLP answer for* $receive(self)$ *and* $\sim receive(self)$ *will be both* UNDECIDED*. Figure 1 shows this situation, where two incomparable arguments block each other. There, arguments are depicted as triangles containing the defeasible rules that form them. The double arrow represents that both arguments attack each other with equal strength.*

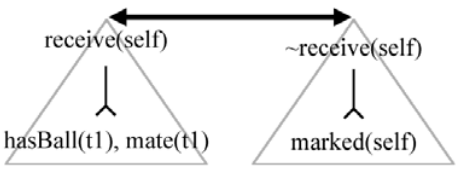


Figure 1: *Undecided situation due to blocking defeaters.*

Consider a new situation where Φ= $\{hasBall(o1)\}$*. Here, the DeLP answer for* $receive(self)$ *is* NO*, because there is a warrant for* $\sim receive(self)$ *supported by the non-defeated argument:*

$$\{\sim receive(self) \prec hasBall(o1), \sim mate(o1)\}.$$

A preference criterion between contradictory arguments is needed in order to prevent blocking situations. For a deeper discussion on this matter we refer to (Chesñevar, Maguitman, & Loui 2000; Prakken & Vreeswijk 2002).

The following propositions show that, although $\mathcal{P}_{\mathsf{B}}$ represents contradictory information, the sets of beliefs B of an agent will be non-contradictory.

Proposition 1 *The set of beliefs* B *of an agent is a set of warranted literals.*
Proof: As proved in (García & Simari 2004), empty arguments have no defeaters and therefore, if a literal can be derived from Π_{B} *then it is warranted. Thus, perceived beliefs are warranted literals, because a fact has an empty argument that supports it. Strict beliefs are also warranted because they are derived using only strict rules and facts from* Π_{B} *and therefore, supported by an empty argument. Defeasible beliefs are warranted by definition.*

Proposition 2 *The set of beliefs* B *of an agent is a non-contradictory set.*
Proof: In order to be a contradictory set, it should have two complementary beliefs (L *and* $\sim L$*). However, proposition 1 states that* B *is a set of warranted literals, and from a defeasible logic program it is not possible to obtain a warrant for a literal* L *and also a warrant for* $\sim L$*.*

Desires Filtering and Selection

In our approach, agents desires will be represented by a set of literals D. This set will contain a literal for representing each desire that the agent might want to achieve, along with its complement; that is, if $L \in \mathsf{D}$, then $\overline{L} \in \mathsf{D}$. As we will explain in detail below, we will assume that beliefs and desires are represented with separate names, *i.e.*, $\mathsf{D} \cap \mathsf{B} = \emptyset$, (see Remark 1).

Example 2 *According to our application domain, the set* D *of all possible desires for a robotic soccer agent could be:*

$$\mathsf{D} = \left\{ \begin{array}{ll} shoot & \sim shoot \\ pass(Mate) & \sim pass(Mate) \\ carry & \sim carry \\ mark(Opp) & \sim mark(Opp) \\ goto(Place) & \sim goto(Place) \end{array} \right\}$$

That is, the set of possible desires of the agent includes: to shoot on goal, to pass the ball to a teammate, to carry the ball, to mark an opponent, to go to a specific position in the field, and each corresponding complement.

The set D represents all the desires that the agent may want to achieve. However, depending on the situation in which it is involved, there could be some desires that will be impossible to be carried out. For example, consider a situation in which the agent does not have the ball and the ball is in a place p, then, the desire $shoot$ will not be possible to be carried out, whereas $goto(p)$ could be a possible option.

Therefore, agents should reason about its desires in order to select the appropriate ones. Following the spirit of the BDI model, once appropriate desires are detected, the agent may select (and commit to) a specific intention (goal), and then select appropriate actions to fulfill that intention.

In this section, we will introduce a reasoning formalism for selecting from D those desires that are suitable to be carried out. In order to perform this selection, the agent will use its beliefs (representing the current situation) and a defeasible logic program (Π_F,Δ_F) composed by *filtering rules* as defined below.

Definition 2 (Filtering rule) *Let* D *be the set of desires of an agent, a* filtering rule *is a strict or defeasible rule that has a literal* $L \in \mathsf{D}$ *in its head and a non-empty body.*

Observe that a filtering rule can be either strict or defeasible and, as we will explain below, the effect in the filtering process will be different. Note also that a filtering rule cannot be a single literal (*i.e.*, a fact). Below we will explain how to use filtering rules in order to select desires, but first we will introduce an example to provide some motivation.

Example 3 *Considering the set* D *introduced in Example 2, the following filtering rules can be defined for selecting desires:*

$$\Pi_F = \left\{ \begin{array}{l} \sim carry \leftarrow \sim hasBall \\ \sim shoot \leftarrow \sim hasBall \end{array} \right\}$$

$$\Delta_F = \left\{ \begin{array}{l} shoot \prec theirGoalieAway \\ carry \prec noOneAhead \\ \sim shoot \prec farFromGoal \\ \sim carry \prec shoot \end{array} \right\}$$

Consider a particular situation in which the agent does not have the ball (*i.e.*, $\sim hasBall$ holds as a belief), then from the agent's beliefs and the filtering rules (Π_F,Δ_F) of Example 3 there are warrants for $\sim carry$ and $\sim shoot$. In this situation the agent should not consider selecting the desires $carry$ and $shoot$ because there are warranted reasons against them.

Suppose now another situation in which the agent has the ball and the opponent goalie is away from its position but the agent is far from the goal (*i.e.*, $\{theirGoalieAway, farFromGoal\} \subset \mathsf{B}$). Then, from the agent's beliefs and the filtering rules (Π_F,Δ_F) of Example 3, there are arguments for both $shoot$ and $\sim shoot$. Since these two arguments defeat each other, a blocking situation occurs and the answer for each one is UNDECIDED. In contrast with the previous situation, here there are no strong reasons against selecting the desire $shoot$, and as we will show below, in this formalism an undecided desire will be eligible.

In Definition 3 below, we will introduce a mechanism for filtering the set of desires D in order to obtain a subset of D containing only those desires achievable in the *current* situation. In order to do that, beliefs and filtering rules should be used in combination. Hence, we need to explain how two defeasible logic programs can be properly combined.

Combining beliefs with filtering rules

In this formalism, agents will have a defeasible logic program (*de.l.p.*) (Π_B,Δ_B) containing rules and facts for deriving beliefs, and a *de.l.p.* (Π_F,Δ_F) with filtering rules for selecting desires. Observe that the union of two *de.l.p.* might not be a *de.l.p.*, because the union of two sets of consistent strict rules could be contradictory (*e.g.*, Π_F=$\{(a \leftarrow b)$, $(\sim a \leftarrow c)\}$ and Π_B=$\{b, c\}$). Therefore, in order to guarantee that $\Pi_\mathsf{B} \cup \Pi_F$ is not contradictory, a merge revision operator is needed. Therefore, instead of simply having ($\Pi_\mathsf{B} \cup \Pi_F$, $\Delta_\mathsf{B} \cup \Delta_F$) we will impose to have ($\Pi_\mathsf{B} \circ \Pi_F$, $\Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X$), where "$\circ$" is a merge revision operator (Fuhrmann 1997). The mechanism of this operator (Falappa, Kern-Isberner, & Simari 2002) "is to add Π_B to Π_F and then eliminate from the result all possible inconsistency by means of an incision function that makes a *cut* over each minimally inconsistent subset of $\Pi_\mathsf{B} \cup \Pi_F$."

Observe that in particular in our approach the set Π_F will contain only strict rules representing filtering rules, whereas the set Π_B will contain beliefs representing the agent perception of the environment. Therefore, the elements of Π_B can not be ignored or deleted. Hence, the conflicting set X of $\Pi_\mathsf{B} \circ \Pi_F$ will be defined by eliminating all the conflicting strict rules from Π_B and Π_F. Thus, if literals L and $\sim L$ can be derived from $\Pi_\mathsf{B} \cup \Pi_F$, those strict rules that have L or $\sim L$ in their heads will be in X.

The merge revision operator will remove the inconsistency from $\Pi_\mathsf{B} \cup \Pi_F$ on behalf of some criterion. This could be achieved, for example, by deleting all the rules in conflict, but the part of the knowledge represented by these rules will be lost. Another option is to convert the status of the (strict) rules involved in the conflict, turning them into defeasible rules (Falappa, Kern-Isberner, & Simari 2002). This criterion intends to keep the encoded information that would be, otherwise, lost.

Therefore, in our case, the union of two *de.l.p.* like (Π_B,Δ_B) and (Π_F,Δ_F) will be a program (Π,Δ), where Π = $\Pi_\mathsf{B} \circ \Pi_F$ and Δ = $\Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X$. The set Π is obtained using a merge revision operator $\circ$ that eliminates X, and the set Δ_X is a set of defeasible rules obtained transforming each strict rule $r \in X$ in a defeasible rule.

Example 4 *Here we show how the merge revision operator works. Consider having the following sets:*
Π_F=$\{(a \leftarrow b)$, $(\sim a \leftarrow c)\}$
Π_B=$\{b, c\}$
Note that in this case $\Pi_F \cup \Pi_\mathsf{B}$ is a contradictory set. Therefore, it cannot be part of a valid de.l.p. *As stated above, we will apply the merge revision operator instead of performing the union of both sets. Hence, we have:*
$\Pi_\mathsf{B} \circ \Pi_F$ = $\{b, c\}$
Δ_X=$\{(a \prec b)$, $(\sim a \prec c)\}$
Thus, after the application of the merge operator, no knowledge is lost (although it is certainly weakened), because it is now encoded under the form of defeasible rules.

Selecting desires

The next definition introduces a mechanism for filtering the set of desires D in order to obtain a subset containing only those desires achievable in the *current* situation.

Definition 3 (Current desires) *Let $K = (\Pi_\mathsf{B} \circ \Pi_F, \Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X)$ be a defeasible logic program where (Π_B,Δ_B) contains rules and facts for deriving beliefs, and (Π_F,Δ_F) contains filtering rules. Given a set D of desires, the set D^c of* Current Desires *will be defined as:*

$$\mathsf{D}^c = \{\delta \in \mathsf{D} \mid \textit{there is no warrant for } \overline{\delta} \textit{ from } K\}$$

Thus, the set D^c will be a subset of the set D and, at any given moment, it will contain those desires that have a chance of being achieved. Observe that a literal L will not belong to D^c when its complement $\overline{L}$ is warranted.

Example 5 *Let's consider a subset of the set of desires from example 2:*

$$\mathsf{D} = \left\{ \begin{array}{ll} shoot & \sim shoot \\ carry & \sim carry \\ goto(Place) & \sim goto(Place) \end{array} \right\}$$

Taking the filtering rules from example 3:

$$\Pi_F = \left\{ \begin{array}{l} \sim carry \leftarrow \sim hasBall \\ \sim shoot \leftarrow \sim hasBall \end{array} \right\}$$

$$\Delta_F = \left\{ \begin{array}{l} shoot \prec theirGoalieAway \\ carry \prec noOneAhead \\ \sim shoot \prec farFromGoal \\ \sim carry \prec shoot \end{array} \right\}$$

Then, if we consider the following perceived beliefs:

$$\Pi_{\mathsf{B}} = \Phi = \left\{ \begin{array}{l} theirGoalieAway \\ noOneAhead \end{array} \right\}$$

We will have these current desires:

$$\mathsf{D}^c = \left\{ \begin{array}{l} shoot \\ carry \\ \sim carry \\ goto(Place) \\ \sim goto(Place) \end{array} \right\}$$

Using the filtering rules from (Π_F,Δ_F), the argument of Figure 2 for $shoot$ can be built. This argument has no defeaters, therefore, its conclusion $shoot$ is warranted (*i.e.*, the DeLP answer is YES). Hence, $shoot$ will belong to the set D^c of current desires. An argument for desire $carry$ also exists, but it is blocked by a counter-argument holding $\sim carry$, as shown in Figure 3; then, both $carry$ and $\sim carry$ DeLP answers are UNDECIDED, and they are included into D^c. Finally, considering the desires $goto(Place)$ and $\sim goto(Place)$, we can see that there are no filtering rules regarding to them, so their DeLP answers are UNKNOWN, and both will belong to D^c.

Figure 2: *Undefeated argument for* $shoot$.

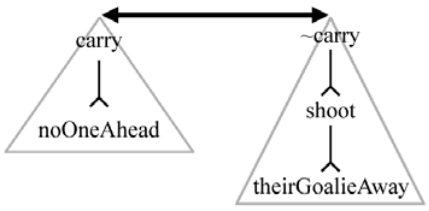


Figure 3: *Blocking situation for* $carry$ *and* $\sim carry$.

You may notice that if neither Q nor $\overline{Q}$ has a warrant built from ($\Pi_{\mathsf{B}} \circ \Pi_F$, $\Delta_{\mathsf{B}} \cup \Delta_F \cup \Delta_X$), then both will be included into the set D^c. Therefore, the agent will have to consider these two options (among the rest of the current desires) in order to choose an intention, although they are in contradiction.

In this work we will require having B and D as two separate sets. If a literal is allowed to belong to both sets, then, when joining the (Π_{B},Δ_{B}) and (Π_F,Δ_F) *de.l.p.* programs, an undesirable mix of concepts would arise. Note that this is not a strong restriction, because the fact that a literal could be both a belief and a desire brings about well-known representational issues.

Remark 1 *We will assume that no literal* $L \in \mathsf{D}$ *belongs to* B*. That is, a desire cannot be perceived or derived as a belief.*

Example 6 *Here we will show why it is important to take Remark 1 into consideration, keeping literals from* B *and* D *apart. Consider the following sets of beliefs and desires:*

$$(\Pi_{\mathsf{B}},\Delta_{\mathsf{B}}) = (\{x, y\},\{a \prec y\})$$
$$\mathsf{D} = \{a, \sim a\}$$

And we will consider these filtering rules:

$$(\Pi_F,\Delta_F) = (\{\},\{\sim a \prec x\})$$

Here, $\Pi_{\mathsf{B}} \circ \Pi_F = \{x, y\}$*; and* $\Delta_{\mathsf{B}} \cup \Delta_F \cup \Delta_X = \{a \prec y, \sim a \prec x\}$*. We have that '*$\sim a$*' has an argument based on* Δ_F*, and '*a*' has an argument built from* (Π_{B},Δ_{B})*. By Definition 3, both literals will belong to* D^c*. However, if we let this happen, we will be letting the beliefs rules decide which desires are going to be in* D^c*. Although the argumentation process involved in the selection of the current desires potentially requires some dialectical analysis to be performed upon the rules defined in* (Π_{B},Δ_{B})*, the elements of* D^c *should be determined only by the filtering rules in* (Π_F,Δ_F)*.*

A simple way of satisfying the restriction imposed by Remark 1 could be to distinguish literals in D with a particular predicate like "$desire$". For example: $desire(shoot)$, $\sim desire(shoot)$, $desire(pass(Mate))$, *etc.* Thus, assuming that no belief is represented with the predicate name "$desire$", then literals representing beliefs will be, by construction, different from the ones representing desires. Although this alternative is supported by this formalism, in the examples given in this paper we will use the convention of having different names for desires and beliefs.

The set D^c can be also defined in terms of DeLP answers. As stated in the last section about DeLP, given a literal Q there are four possible answers for the query Q: YES, NO, UNDECIDED, and UNKNOWN. Thus, given $Q \in \mathsf{D}$, Q will be in D^c if, from ($\Pi_{\mathsf{B}} \circ \Pi_F$, $\Delta_{\mathsf{B}} \cup \Delta_F \cup \Delta_X$), the answer for Q is YES, UNDECIDED or UNKNOWN.

Next, we introduce several propositions and properties that show how different characteristics of both D and (Π_F,Δ_F) determine the contents of D^c.

Proposition 3 *Given a literal* $Q \in \mathsf{D}$*, if there is no filtering rule in* (Π_F,Δ_F) *with head* Q *nor its complement* $\overline{Q}$*, then* $\{ Q, \overline{Q} \} \subseteq \mathsf{D}^c$*.*

Proof: If $Q \in \mathsf{D}$*, then* $\overline{Q} \notin \mathsf{B}$ *(remark 1), and since* $\overline{Q}$ *is not in the head of any filtering rule, then it is not possible to obtain a warrant for* $\overline{Q}$*. Therefore, by Definition 3,* Q *will be included into the set* D^c *of current desires.*

Consider the sets Π_F, Δ_F, Π_{B} and D from Example 5; we have that $goto(Place)$ and $\sim goto(Place)$ belong to D, but there are no filtering rules in (Π_F,Δ_F) with head $goto(Place)$ nor $\sim goto(Place)$. Note that both literals are in D^c, which corresponds with Proposition 3.

Observe that the proof of Proposition 3 can also be expressed in terms of DeLP answers. Since there are no rules with head Q nor its complement, and $\overline{Q} \notin \mathsf{B}$, then Q is not in the language of ($\Pi_{\mathsf{B}} \circ \Pi_F$, $\Delta_{\mathsf{B}} \cup \Delta_F \cup \Delta_X$). Therefore, from ($\Pi_{\mathsf{B}} \circ \Pi_F$, $\Delta_{\mathsf{B}} \cup \Delta_F \cup \Delta_X$) the answer for Q will be UNKNOWN (the same holds for $\overline{Q}$). Because these answers

are different from NO, Q and $\overline{Q}$ will be included into the set D^c of current desires.

Proposition 4 *Given a literal $Q \in \mathsf{D}$, if Q is warranted from $K = (\Pi_\mathsf{B} \circ \Pi_F, \Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X)$, then $Q \in \mathsf{D}^c$ and $\overline{Q} \notin \mathsf{D}^c$.*
Proof: Since Q has a warrant built from K, then there is no warrant for $\overline{Q}$, therefore, $Q \in \mathsf{D}^c$. To prove that $\overline{Q} \notin \mathsf{D}^c$, we have to show that there is a warrant for the complement of $\overline{Q}$ (i.e., Q) which is true by hypothesis.

Considering the case shown in Example 5, see that desire $shoot$ has a warrant built from $(\Pi_\mathsf{B} \circ \Pi_F, \Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X)$, and $shoot$ belongs to D^c, but $\sim shoot$ does not, which coincides with the assertion of Proposition 4.

The next proposition shows that the set D^c of current desires will not be empty if the set D of desires is not empty. This does not depend on the set of filtering rules nor the set of beliefs.

Proposition 5 *For any $K = (\Pi_\mathsf{B} \circ \Pi_F, \Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X)$, D will not be empty if, and only if, D^c is not empty.*
Proof:
($\Rightarrow$) Let L and $\overline{L}$ be two elements of D, then one of the three following cases holds:
(a) L is warranted from K, therefore, $L \in \mathsf{D}^c$.
(b) $\overline{L}$ is warranted from K, therefore, $\overline{L} \in \mathsf{D}^c$
(c) Neither L nor $\overline{L}$ is warranted from K, therefore, both L and $\overline{L}$ will belong to D^c.
Hence, in any case, D^c will not be empty.
($\Leftarrow$) Trivial from Definition 3. That is, if D^c is not empty, then D cannot be empty.

Given an agent with $K = (\Pi_\mathsf{B} \circ \Pi_F, \Delta_\mathsf{B} \cup \Delta_F \cup \Delta_X)$, and a set of desires D, the following properties hold:

1. *For every literal $L \in \mathsf{D}$, it cannot be the case that neither L nor $\overline{L}$ do not belong to D^c.*
   Suppose that L and $\overline{L}$ are not in D^c, then, the complement of each (*i.e.*, $\overline{L}$ and L) must be warranted from K, which is not allowed in DeLP. Observe that, if both literals are warranted, then both literals have to belong to D^c, which contradicts the initial supposition.
2. *If there are no filtering rules then $\mathsf{D}^c = \mathsf{D}$.*
   This is a particular case of Proposition 3. If there are no filtering rules (*i.e.*, $\Pi_F = \emptyset$ and $\Delta_F = \emptyset$) then no element of D will have its complement warranted, so every element of D will be in D^c.
3. *If there are no desires warranted from K, then $\mathsf{D}^c = \mathsf{D}$.*
   If no element of D is warranted from K, then no element of D will have its complement warranted from K ($L \in \mathsf{D}$ implies that $\overline{L} \in \mathsf{D}$). Thus, every element of D will be in D^c.
4. *If there is, at least, one element of D that is warranted from K, then D^c will be a proper subset of D.*
   If L belongs to D and is warranted from K, then $\overline{L} \notin \mathsf{D}^c$.

Application to Robotic Soccer

Robotic soccer has proven to be a system complex enough to test many of the features of any reasoning system. The robots are controlled by software agents, each of which has a set of high-level actions to perform, such as kicking the ball with a given strength or moving in a given direction. At every moment, an agent has to choose which action to do next. That choice can be made using a reasoning system, in this case, a defeasible argumentation system.

In this Section we will show how a player makes a decision based on the model proposed in this paper. The scenario has three players belonging to one team ('self', 't1' and 't2' in Figure 4) and three players from the opposite team ('o1', 'o2' and 'o3' in Figure 4). We will analyze the reasoning performed by the player named 'self'. Regarding knowledge representation, the belief predicates will be written according to the Close World Assumption, and everything that cannot be proved will be assumed false.

Let's suppose that we have a soccer-playing agent, with a desires set including the options $shoot$, $pass$ and $carry$, being in the situation depicted in Figure 4. The agent will perceive the positions of the ball and all the other players, and will reason about what to do next. The agent will have different rules, like $pass(self, t1) \prec marked(self)$, that will be a reason for passing the ball to $t1$.

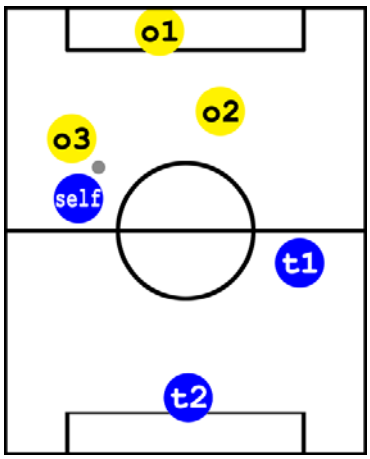


Figure 4: *'self' decides to pass the ball to 't1'*

In this situation, the agent has the following perceptions:

$$\Phi = \left\{ \begin{array}{l} marked(self), \\ oppositeField(self), \\ noOneAhead(t1), \\ hasBall(self), \\ betterPosition(self, t1) \end{array} \right\}$$

The knowledge is represented through a *de.l.p.*, defining reasons for and against every element belonging to the set D via strict and defeasible rules. In this example, the sets Δ_F and Π_F are the following:

$\Pi_F = \{$
$(\sim shoot(P) \leftarrow \sim hasBall(P))$,
$(\sim pass(Src, \_) \leftarrow \sim hasBall(Src))$,

$(\sim carry(P) \leftarrow \sim hasBall(P))$ }

Δ_F= {
$(shoot(P) -\!\prec opposite Field(P), noOneAhead(P))$,
$(shoot(P) -\!\prec opposite Field(P), not\ marked(P))$,
$(shoot(\_) -\!\prec goalieAway(opposite))$,

$(\sim shoot(P) -\!\prec pass(P, \_))$,
$(\sim shoot(P) -\!\prec carry(P))$,

$(pass(Src, \_) -\!\prec marked(Src))$,
$(pass(Src, Tgt) -\!\prec betterPosition(Tgt, Src))$,

$(\sim pass(Src, Tgt) -\!\prec playerBetween(Src, Tgt))$,
$(\sim pass(\_, Tgt) -\!\prec marked(Tgt))$,
$(\sim pass(Src, \_) -\!\prec shoot(Src))$,
$(\sim pass(Src, \_) -\!\prec carry(Src))$,

$(carry(P) -\!\prec noOneAhead(P))$,

$(\sim carry(P) -\!\prec shoot(P))$,
$(\sim carry(P) -\!\prec pass(P, \_))$ }

At the particular moment of Figure 4, the player has the following Beliefs set:

$$\mathsf{B} = \left\{ \begin{array}{l} marked(self), \\ oppositeField(self), \\ noOneAhead(t1), \\ hasBall(self), \\ betterPosition(self, t1), \\ teammate(t1), \\ myRole(forward) \end{array} \right\}$$

Observe that some of these beliefs are perceived (*e.g.*, $marked(self)$), while other are persistent (*e.g.*, $teammate(t1)$).

Once built the internal representation of the world, the agent performs DeLP queries over the elements of its set D, gathering their corresponding answers:

- *Shooting on goal:* If you consider program $\mathcal{P}$, along with the set B of Beliefs, it is clear that no argument can be built for the desire $shoot(self)$, because every rule with head $shoot(self)$ has at least one literal in its body that does not hold. As you can see in Figure 4, there are no significant common-sense reasons to shoot on goal.

  On the other hand, an argument for $\sim shoot(self)$ can be constructed from $\mathcal{P}$: the one built upon the argument of being able to perform a pass to a teammate. That counter-argument is undefeated: there are no reasons against it. Figure 4 illustrates this situation, and shows that passing the ball to player 't1' seems to be the better choice to make at that moment.

  The answer given from the interpreter is NO and the agent will no longer consider shooting on goal.

- *Passing the ball:* From program $\mathcal{P}$, one argument can be stated for $pass(self, t1)$, based on the reason that player 'self' is marked. None of the rules that could build an argument against it holds, so this one is undefeated. As explained above, from the situation we are considering, it is reasonable to think that this may be a good choice.

  The answer given from the interpreter is YES and the agent will have a strong reason to pass the ball to 't1'.

- *Carrying the ball:* there are no rules from which an argument for carrying the ball could be built. Program $\mathcal{P}$ and Figure 4 coincide in this matter: there seems to be no reason for carrying the ball. On the other hand, the aforementioned argument for passing the ball is an undefeated reason for not doing it.

  The interpreter answers NO and the agent will no longer consider this desire.

Once gathered all the DeLP answers, the set D^c of current desires can be built:

$$\mathsf{D}^c = \{\ pass(self, t1)\ \}$$

This means that, in this situation, the player has a clear choice: the selected intention must be *perform a pass to 't1'*, because it is the only current desire and it is warranted.

Related work

In a very recent paper (Rahwan & Amgoud 2006), Rahwan and Amgoud have proposed an argumentation-based approach for practical reasoning that extends (Amgoud 2003) and (Amgoud & Cayrol 2002), introducing three different instantiations of Dung's abstract argumentation framework in order to reason about beliefs, desires and plans, respectively. This work is, in our concern, the one that is most related to our approach and, as we will show next, it have many points in common with our work. Both approaches use a defeasible argumentation formalism for reasoning about beliefs and desires (in their work, they also reason about plans, but this is out of the scope of our presentation). Like us, they separate in the language those rules for reasoning about belief from those rules for reasoning about desires; and, in both approaches, it is possible to represent contradictory information about beliefs and desires. Since both approaches use a defeasible argumentation formalism, they construct arguments supporting competing desires, and these arguments are compared and evaluated in order to decide which one prevails. Their notion of *desire rule* is very similar to our notion of *filtering rule*.

Although both approaches have many things in common, they also differ in many points. In their case, two different argumentation frameworks are needed in order to reason about desires: one framework for beliefs rules and other framework for desires rules. The last one depends directly on the first one, and since there are two kinds of arguments, a policy for comparing mixed arguments is given. In our case, only one argumentation formalism is used for reasoning with both types of rules. In their object language, beliefs and desires include a certainty factor for every formula, and no explicitly mention of perceived information is given. In our case, uncertainty is represented by defeasible rules (see (García & Simari 2004)) and perceived beliefs are explicitly treated by the model. Besides, although both approaches use defeasible argumentation, the argumentative formalism used in their approach differs from ours. In their case, the comparison of arguments relies on the certainty

factor given to each formula, and they do not distinguish between proper and blocking defeaters.

The use of defeasible argumentation for reasoning in BDI architectures is not new and it was originally mention in the seminal paper (Bratman, Israel, & Pollack 1991) and in other more recent works like (Parsons, Sierra, & Jennings 1998). Other related work includes (Thomason 2000) and (Broersen *et al.* 2001), where a formalism for reasoning about beliefs and desires is given. However, these last formalisms differ from our because they do not use argumentation.

Conclusions and future work

In this paper we have shown how a deliberative agent can represent its perception and beliefs using a defeasible logic program. The information that the agent perceives directly from its environment is represented with a subset of *perceived beliefs* that is updated dynamically, and a subset Π formed with strict rules and facts represent other static knowledge of the agent. Defeasible argumentation is used in order to warrant agents (derived) beliefs.

Strict and defeasible filtering rules have been introduced to represent knowledge for selecting desires and a defeasible argumentation can be used for selecting a proper desire that fits in the particular situation the agent is involved.

With this formalism, agents can reason about its desires in order to select the appropriate ones. However, following the spirit of the BDI model, once appropriate desires are detected, the agent should select (and commit to) a specific intention (goal), and then select appropriate actions to fulfill that intention. We are currently working in extending this approach to consider the representation of agent's intentions and reasoning about them.

References

Amgoud, L., and Cayrol, C. 2002. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence* 34(1-3):197–215.

Amgoud, L. 2003. A formal framework for handling conflicting desires. In *Proceedings of the 7th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'2003*, 552–563.

Bratman, M. E.; Israel, D.; and Pollack, M. 1991. Plans and resource-bounded practical reasoning. In Cummins, R., and Pollock, J. L., eds., *Philosophy and AI: Essays at the Interface*. Cambridge, Massachusetts: The MIT Press. 1–22.

Broersen, J.; Dastani, M.; Hulstijn, J.; Huang, Z.; and van der Torre, L. 2001. The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of Fifth International Conference on Autonomous Agents (Agents2001)*. Montreal, Canada: ACM Press. 9–16.

Chesñevar, C. I.; Maguitman, A. G.; and Loui, R. P. 2000. Logical Models of Argument. *ACM Computing Surveys* 32(4):337–383.

Falappa, M. A.; Kern-Isberner, G.; and Simari, G. R. 2002. Belief revision, explanations and defeasible reasoning. *Artificial Intelligence Journal* 141:1–28.

Fuhrmann, A. 1997. *An Essay on Contraction*. Studies in Logic, Language and Information, CSLI Publications, Stanford, California.

García, A. J., and Simari, G. R. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Lifschitz, V. 1996. Foundations of logic programming. In Brewka, G., ed., *Principles of Knowledge Representation*. CSLI. 69–127.

Parsons, S.; Sierra, C.; and Jennings, N. 1998. Agents that reason and negotiate by arguing. *Journal of Logic and Computation* 8(3):261–292.

Prakken, H., and Vreeswijk, G. 2002. Logical systems for defeasible argumentation. In D.Gabbay., ed., *Handbook of Philosophical Logic, 2nd ed.* Kluwer Academic Pub.

Rahwan, I., and Amgoud, L. 2006. An argumentation-based approach for practical reasoning. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*.

Rao, A. S., and Georgeff, M. P. 1995. BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*.

Rao, A. S. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., ed., *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53(2–3):125–157.

Thomason, R. 2000. Desires and defaults: A framework for planning with inferred goals. In *Proceedings of the seventh international Confenrence on Principle of Knowledge Representation and Reasoning (KR'00)*, 702–713.

5.9 Refining SCC Decomposition in Argumentation Semantics: A First Investigation

Refining SCC decomposition in argumentation semantics: a first investigation

Pietro Baroni and **Massimiliano Giacomin**
Dipartimento di Elettronica per l'Automazione, University of Brescia
Via Branze 38
25123 Brescia Italy
{baroni,giacomin}@ing.unibs.it

Abstract

In the recently proposed SCC-recursive approach to argumentation semantics, the strongly connected components of an argumentation framework are used as the basic elements for the incremental construction of extensions. In this paper we argue that a finer decomposition, considering some suitably defined internal substructures of strongly connected components, called *autonomous fragments*, may be appropriate and support, in some cases, more intuitive results than the original approach. We cast this proposal within the SCC-recursive framework, show that it satisfies some fundamental requirements and provide some examples of its potential advantages.

Introduction

The notion of SCC-recursiveness has recently been introduced (Baroni, Giacomin, & Guida 2005; Baroni & Giacomin 2004a) as a general scheme for argumentation semantics. On one hand, SCC-recursiveness is able to encompass most significant existing proposals such as *grounded semantics* (Pollock 1992) and *preferred semantics* (Dung 1995), on the other hand, it provides a sound basis for the definition and investigation of novel semantics proposals. In particular, the SCC-recursive $CF2$ semantics (Baroni & Giacomin 2004b; Baroni, Giacomin, & Guida 2005) has been shown to produce intuitively plausible results in some cases, involving odd-length cycles, which are quite problematic for other semantics. The definition of SCC-recursiveness stands on some widely accepted basic principles which can be regarded as a common ground for any argumentation semantics: the *conflict free* principle, the *reinstatement* principle (Prakken & Vreeswijk 2001), and the *directionality* principle. In particular, the last one suggests that defeat dependencies among arguments can be taken into account following the partial order induced by the decomposition of the graph representation of an argumentation framework into *Strongly Connected Components* (SCCs). In some cases it emerges however that this decomposition is, in a sense, still "too rough" to capture some intuitively significant aspects of the defeat graph topology and a further decomposition may be appropriate. This work starts from this observation and presents a preliminary investigation about why and how such a finer decomposition can be carried out in the framework of SCC-recursive semantics. The paper is organized as follows. In the following section we recall the necessary background concepts on SCC-recursiveness, while in the next one we introduce some motivating examples for our investigation. We then introduce the notion of *autonomous fragments* within a strongly connected component and show how this notion can be exploited within the SCC-recursive scheme. After exemplifying the application of the proposed approach, we conclude the paper with some final remarks.

SCC-recursiveness

We first give an account of SCC-recursiveness as introduced in (Baroni, Giacomin, & Guida 2005). The approach lies in the frame of the general theory of abstract argumentation frameworks proposed by Dung (Dung 1995).

Definition 1 *An argumentation framework is a pair* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$*, where* $\mathcal{A}$ *is a set, and* $\rightarrow \subseteq (\mathcal{A} \times \mathcal{A})$ *is a binary relation on* $\mathcal{A}$*, called attack relation.*

In the following we will always assume that $\mathcal{A}$ is finite. An argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ can be represented as a directed graph, called *defeat graph*, where nodes are the arguments and edges correspond to the elements of the attack relation. In the following, the nodes that attack a given argument α are called *defeaters* of α and form a set which is denoted as $\mathrm{parents}_{\mathrm{AF}}(\alpha)$:

Definition 2 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and a node* $\alpha \in \mathcal{A}$*, we define* $\mathrm{parents}_{\mathrm{AF}}(\alpha) = \{\beta \in \mathcal{A} \mid \beta \rightarrow \alpha\}$*. If* $\mathrm{parents}_{\mathrm{AF}}(\alpha) = \emptyset$*, then* α *is called an* initial *node.*

Since we will frequently consider properties of sets of arguments, it is useful to extend to them the notations defined for the nodes:

Definition 3 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$*, a node* $\alpha \in \mathcal{A}$ *and two sets* $S, P \subseteq \mathcal{A}$*, we define:*

$$S \rightarrow \alpha \equiv \exists \beta \in S : \beta \rightarrow \alpha$$
$$\alpha \rightarrow S \equiv \exists \beta \in S : \alpha \rightarrow \beta$$
$$S \rightarrow P \equiv \exists \alpha \in S, \beta \in P : \alpha \rightarrow \beta$$
$$\mathrm{outparents}_{\mathrm{AF}}(S) = \{\alpha \in \mathcal{A} \mid \alpha \notin S \wedge \alpha \rightarrow S\}$$

In Dung's theory, an argumentation semantics is defined by specifying the criteria for deriving, given a generic argumentation framework, the set of all possible extensions, each

one representing a set of arguments considered to be acceptable together. Typically an argument is considered *justified* if and only if it belongs to all extensions. Given a generic argumentation semantics $\mathcal{S}$, the set of extensions prescribed by $\mathcal{S}$ for a given argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ is denoted as $\mathcal{E}_{\mathcal{S}}(\mathrm{AF})$. If it holds that $\forall \mathrm{AF}, |\mathcal{E}_{\mathcal{S}}(\mathrm{AF})| = 1$, then the semantics $\mathcal{S}$ is said to follow the *unique-status approach*, otherwise it is said to follow the *multiple-status approach* (Prakken & Vreeswijk 2001).

SCC-recursiveness is a property of the extensions which relies on the graph theoretical notion of *strongly connected components* (SCCs).

Definition 4 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$*, the binary relation of path-equivalence between nodes, denoted as* $PE_{\mathrm{AF}} \subseteq (\mathcal{A} \times \mathcal{A})$*, is defined as follows:*

- $\forall \alpha \in \mathcal{A}, (\alpha, \alpha) \in PE_{\mathrm{AF}}$
- *given two distinct nodes* $\alpha, \beta \in \mathcal{A}$*,* $(\alpha, \beta) \in PE_{\mathrm{AF}}$ *if and only if there is a path from* α *to* β *and a path from* β *to* α*.*

The *strongly connected component* of AF are the equivalence classes of nodes under the relation of path-equivalence. The set of the SCCs of AF is denoted as $\mathrm{SCCS}_{\mathrm{AF}}$. A particular case is represented by the empty argumentation framework: when $\mathrm{AF} = \langle \emptyset, \emptyset \rangle$ we assume $\mathrm{SCCS}_{\mathrm{AF}} = \{\emptyset\}$.

We extend to SCCs the notion of parents, namely the set of the other SCCs that attack a SCC S, which is denoted as $\mathrm{sccpar}_{\mathrm{AF}}(S)$, and we introduce the definition of *proper ancestors*, denoted as $\mathrm{sccanc}_{\mathrm{AF}}(S)$:

Definition 5 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and a SCC* $S \in \mathrm{SCCS}_{\mathrm{AF}}$*, we define*

$$\mathrm{sccpar}_{\mathrm{AF}}(S) = \{P \in \mathrm{SCCS}_{\mathrm{AF}} \mid P \neq S \text{ and } P \rightarrow S\}$$

and

$$\mathrm{sccanc}_{\mathrm{AF}}(S) = \mathrm{sccpar}_{\mathrm{AF}}(S) \cup \bigcup_{P \in \mathrm{sccpar}_{\mathrm{AF}}(S)} \mathrm{sccanc}_{\mathrm{AF}}(P)$$

A SCC S such that $\mathrm{sccpar}_{\mathrm{AF}}(S) = \emptyset$ is called *initial*.

It is well-known that the graph obtained by considering SCCs as single nodes is acyclic. In other words, SCCs can be partially ordered according to the relation of attack. Following the above definition, *initial* SCCs are those which are not preceded by any other one in this partial order. Of course, in any argumentation framework there is at least one initial SCC. This fact lies at the heart of the definition of SCC-recursiveness, which is based on the intuition that extensions can be built incrementally starting from initial SCCs and following the above mentioned partial order. In other words, the choices concerning extension construction carried out in an initial SCC do not depend on those concerning the other ones, while they directly affect the choices about the subsequent SCCs and so on.

While the basic underlying intuition is rather simple, the formalization of SCC-recursiveness is admittedly rather complex and involves some additional notions. Due to space limitations, we can only give here a quick account, while referring the reader to (Baroni, Giacomin, & Guida 2005) for more details and examples. First of all, the choices in the antecedent SCCs determine a partition of the nodes of a generic SCC S into three subsets:

Definition 6 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$*, a set* $E \subseteq \mathcal{A}$ *and a set* $S \subseteq \mathcal{A}$*, we define:*

- $D_{\mathrm{AF}}(S, E) = \{\alpha \in S \mid (E \cap \mathrm{outparents}_{\mathrm{AF}}(S)) \rightarrow \alpha\}$
- $P_{\mathrm{AF}}(S, E) = \{\alpha \in S \mid (E \cap \mathrm{outparents}_{\mathrm{AF}}(S)) \not\rightarrow \alpha \wedge \exists \beta \in (\mathrm{outparents}_{\mathrm{AF}}(S) \cap \mathrm{parents}_{\mathrm{AF}}(\alpha)) : E \not\rightarrow \beta \wedge \alpha \not\rightarrow \beta\}$
- $U_{\mathrm{AF}}(S, E) = S \setminus (D_{\mathrm{AF}}(S, E) \cup P_{\mathrm{AF}}(S, E)) = \{\alpha \in S \mid (E \cap \mathrm{outparents}_{\mathrm{AF}}(S)) \not\rightarrow \alpha \wedge \forall \beta \in (\mathrm{outparents}_{\mathrm{AF}}(S) \cap \mathrm{parents}_{\mathrm{AF}}(\alpha)) \; E \cup \{\alpha\} \rightarrow \beta\}$

Definition 6 is a generalized version (useful for the sequel of the paper) of the corresponding Definition 18 of (Baroni, Giacomin, & Guida 2005). In words, the set $D_{\mathrm{AF}}(S, E)$ consists of the nodes of S attacked by E from outside S, the set $U_{\mathrm{AF}}(S, E)$ includes any node α of S that is not attacked by E from outside S and is defended by E or defends itself (i.e. the defeaters of α from outside S are all attacked by E or by α itself), and $P_{\mathrm{AF}}(S, E)$ includes any node α of S that is not attacked by E from outside S and is not defended by E or by itself (i.e. at least one of the defeaters of α from outside S is not attacked by E nor by α). It is easy to verify that, when S is a SCC, as in the original Definition 18 of (Baroni, Giacomin, & Guida 2005), $D_{\mathrm{AF}}(S, E)$, $P_{\mathrm{AF}}(S, E)$ and $U_{\mathrm{AF}}(S, E)$ are determined only by the elements of E that belong to the SCCs in $\mathrm{sccanc}_{\mathrm{AF}}(S)$ and it may not be the case that a node $\alpha \in S$ defends itself against an attack coming from outside S. Regarding E as a part of an extension which is being constructed, the idea is then that arguments in $D_{\mathrm{AF}}(S, E)$, being attacked by nodes in E, cannot be chosen in the construction of the extension E (i.e. do not belong to $E \cap S$). Selection of arguments to be included in E is therefore restricted to $(S \setminus D_{\mathrm{AF}}(S, E)) = (U_{\mathrm{AF}}(S, E) \cup P_{\mathrm{AF}}(S, E))$, which, for ease of notation, will be denoted in the following as $UP_{\mathrm{AF}}(S, E)$.

To formalize this aspect, we define the *restriction* of an argumentation framework to a given subset of its nodes:

Definition 7 *Let* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *be an argumentation framework, and let* $S \subseteq \mathcal{A}$ *be a set of arguments. The* restriction *of* AF *to* S *is the argumentation framework* $\mathrm{AF}{\downarrow_S} = \langle S, \rightarrow \cap (S \times S) \rangle$.

Inspired by the reinstatement principle, we require the selection of nodes within a SCC S to be carried out on the restricted argumentation framework $\mathrm{AF}{\downarrow_{UP_{\mathrm{AF}}(S,E)}}$ without taking into account the attacks coming from $D_{\mathrm{AF}}(S, E)$.

Combining these ideas and skipping some details not strictly necessary in the context of the present paper, we can finally recall the definition of *SCC-recursiveness*:

Definition 8 *A given argumentation semantics* $\mathcal{S}$ *is* SCC-recursive *if and only if for any argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$*,* $\mathcal{E}_{\mathcal{S}}(\mathrm{AF}) = \mathcal{GF}(\mathrm{AF}, \mathcal{A})$*, where for any* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and for any set* $C \subseteq \mathcal{A}$*, the function* $\mathcal{GF}(\mathrm{AF}, C) \subseteq 2^{\mathcal{A}}$ *is defined as follows:*
for any $E \subseteq \mathcal{A}$*,* $E \in \mathcal{GF}(\mathrm{AF}, C)$ *if and only if*

- *in case* $|\mathrm{SCCS}_{\mathrm{AF}}| = 1$*,* $E \in \mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$

- *otherwise,* $\forall S \in \mathrm{SCCS}_{\mathrm{AF}}$ $(E \cap S) \in \mathcal{GF}(\mathrm{AF}{\downarrow_{UP_{\mathrm{AF}}(S,E)}}, U_{\mathrm{AF}}(S,E) \cap C)$

where $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$ is a function, called *base function*, that, given an argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ such that $|\mathrm{SCCS}_{\mathrm{AF}}| = 1$ and a set $C \subseteq \mathcal{A}$, gives a subset of $2^{\mathcal{A}}$.

Since this definition is somewhat arduous to examine in its full detail, we just give some "quick and dirty" indications which are useful for the sequel of the paper (in particular, we do not consider the meaning of the parameter C in the description, as not necessary for the comprehension of this paper). The set of extensions $\mathcal{E}_{\mathcal{S}}(\mathrm{AF})$ of an argumentation framework AF is given by $\mathcal{GF}(\mathrm{AF}, \mathcal{A})$, namely by the invocation of the function $\mathcal{GF}$ which receives as parameters an argumentation framework (in this case the whole AF) and a set of arguments (in this case the whole $\mathcal{A}$). The function $\mathcal{GF}(\mathrm{AF}, C)$ is defined recursively. The base of the recursion is reached when AF consists of a unique SCC: in this case the set of extensions is directly given by the invocation of a semantic-specific base function $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$. In the other case, for each SCC S of AF the function $\mathcal{GF}$ is invoked recursively on the restriction $\mathrm{AF}{\downarrow_{UP_{\mathrm{AF}}(S,E)}}$.

Note that the restriction concerns $UP_{\mathrm{AF}}(S,E)$, namely the part of S which "survives" the attacks of the preceding ones in the partial order. The definition also has a constructive interpretation, which suggests an effective (recursive) procedure for computing all the extensions of an argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ once a specific base function characterizing the semantics is assigned. A particular role in this context is played by the initial SCCs. In fact, for any initial SCC I, since by definition there are no outer attacks, the set of defended nodes coincides with I, i.e. $UP_{\mathrm{AF}}(I,E) = U_{\mathrm{AF}}(I,E) = I$ for any E. This gives rise to the invocation $\mathcal{GF}(\mathrm{AF}{\downarrow_I}, I)$ for any initial SCC I. Since $\mathrm{AF}{\downarrow_I}$ obviously consists of a unique SCC, according to Definition 8 the base function $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}{\downarrow_I}, I)$ is invoked, which returns the extensions of $\mathrm{AF}{\downarrow_I}$ according to the semantics $\mathcal{S}$. Therefore, the base function can be first computed on the initial SCCs, where it directly returns the extensions prescribed by the semantics. Then, the results of this computation are used to identify, within the subsequent SCCs, the restricted argumentation frameworks on which the procedure is recursively invoked.

All SCC-recursive semantics "share" this general scheme and only differ by the specific base function adopted. It has been shown that all semantics encompassed by Dung's framework are SCC-recursive and the relevant base functions have been identified. Among them, in the following we will mainly refer to grounded semantics (denoted as $\mathcal{GR}$) and preferred semantics (denoted as $\mathcal{PR}$) considered the "best" representatives of the unique-status and multiple-status approach respectively.

Moreover, defining and experimenting new SCC-recursive semantics is quite easy since it simply amounts to defining a base function operating on single-SCC argumentation frameworks. As shown in (Baroni, Giacomin, & Guida 2005), the base function has only to respect two very simple conditions in order to ensure that the resulting extensions satisfy the fundamental requirements of being conflict-free and of agreement with grounded semantics.

As to the conflict-free property, it is sufficient that the base function returns only conflict-free subsets.

Definition 9 *A semantics $\mathcal{S}$ satisfies the* conflict-free property *if and only if* $\forall \mathrm{AF}, \forall E \in \mathcal{E}_{\mathcal{S}}(\mathrm{AF})$ E *is conflict-free.*

Definition 10 *The base function of a SCC-recursive semantics $\mathcal{S}$ is* conflict-free *if and only if* $\forall \mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and* $\forall C \subseteq \mathcal{A}$ *each element of* $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$ *is conflict-free.*

Proposition 1 *(Theorem 48 of (Baroni, Giacomin, & Guida 2005)) Given a SCC-recursive semantics $\mathcal{S}$, if its base function is conflict-free then $\mathcal{S}$ satisfies the conflict-free property.*

As to the agreement with grounded semantics, it is sufficient that the base function properly deals just with the simplest case of non-empty argumentation framework (a single node not attacking itself).

Proposition 2 *(Theorem 52 of (Baroni, Giacomin, & Guida 2005)) Let $\mathcal{S}$ be a SCC-recursive semantics identified by a conflict-free base function such that*

$$\mathcal{BF}_{\mathcal{S}}(\langle \{\alpha\}, \emptyset \rangle, \{\alpha\}) = \{\{\alpha\}\}$$

For any argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$, $\forall E \in \mathcal{E}_{\mathcal{S}}(\mathrm{AF})$, *the grounded extension* $\mathrm{GE}(\mathrm{AF}) \subseteq E$.

Thanks to these properties, four original SCC-recursive semantics have been defined in (Baroni, Giacomin, & Guida 2005) in a relatively straightforward way. In particular, the SCC-recursive semantics called $CF2$ (Baroni & Giacomin 2003; Baroni, Giacomin, & Guida 2005) has been shown to provide a good behavior in several critical examples, while featuring a very simple base function: $\mathcal{BF}_{CF2}(\mathrm{AF}, C) = \mathcal{MCF}_{\mathrm{AF}}$, where $\mathcal{MCF}_{\mathrm{AF}}$ denotes the set made up of all the maximal conflict-free sets of AF (note that the parameter C plays no role at all in this case).

Motivating examples

In the SCC-recursive approach, SCCs play the role of basic decomposition elements on which the semantics-specific base function is applied. In $CF2$ semantics, as well as in the SCC-recursive formulation of grounded, stable, and preferred semantics, the base function does not take into account the "internal topology" of the SCC to which it is applied. Roughly speaking, since all elements of a SCC are mutually reachable, it has been implicitly assumed in (Baroni, Giacomin, & Guida 2005) that they can be treated as "equivalent" in the construction of extensions.

Though this hidden assumption is reasonable in most situations, there are cases where it can be regarded as questionable.

As a first example, consider the argumentation framework AF_1 represented in Figure 1. AF_1 clearly consists of a single SCC, so its extensions in a SCC-recursive semantics are directly obtained by applying the base function to the whole AF_1. In the case of $CF2$ semantics it turns out that $\mathcal{E}_{CF2}(\mathrm{AF}_1) = \mathcal{MCF}_{\mathrm{AF}_1} = \{\{\beta\}, \{\gamma\}\}$. According to both grounded and preferred semantics, the only extension in this case is the empty set: $\mathcal{E}_{\mathcal{PR}}(\mathrm{AF}_1) = \mathcal{E}_{\mathcal{GR}}(\mathrm{AF}_1) =$

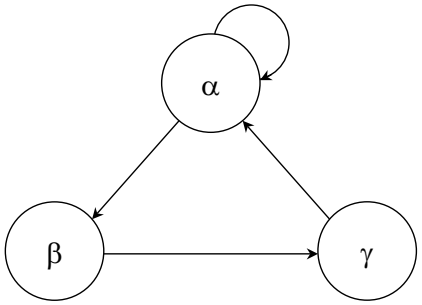


Figure 1: A self-defeating node within a three-length cycle (AF_1).

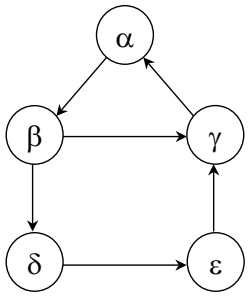


Figure 2: A three-length cycle within a five-length cycle(AF_2).

$\{\emptyset\}$. Therefore in all these semantics no argument is included in all the prescribed extensions and can be considered justified. However, this result can be questioned. In fact, one may object that the node α, being self-defeating, is intrinsically weak and should not be able to affect the justification status of the arguments it attacks. In this perspective, α should be ruled out, β, not receiving attacks anymore, should be regarded as justified and, as a consequence, γ should not be justified. While this kind of behavior is not supported by any of the above mentioned semantics, it could be obtained by a sort of ad-hoc rule or by some form of graph preprocessing devoted to suppress all self-defeating nodes.

Other examples call however for a more general approach to this kind of situation. Consider the argumentation framework AF_2 represented in Figure 2, which also consists of a unique SCC. According to $CF2$ semantics we have $\mathcal{E}_{CF2}(\mathrm{AF}_2) = \mathcal{MCF}_{\mathrm{AF}_2} = \{\{\alpha,\delta\},\{\alpha,\epsilon\},\{\gamma,\delta\},\{\beta,\epsilon\}\}$, while $\mathcal{E}_{\mathcal{PR}}(\mathrm{AF}_2) = \mathcal{E}_{\mathcal{GR}}(\mathrm{AF}_2) = \{\emptyset\}$. Again, no argument is justified according to any of the above semantics. While this may sound very reasonable, other interpretations are possible, depending on the meaning ascribed to odd-length cycles. In fact, it can be noted that arguments α, β, and γ form a three length cycle, independently of δ and ϵ. $CF2$ semantics is based on the idea that even- and odd-length cycles share the same nature and should be treated equally (Baroni & Giacomin 2003) in a "length-independent" way. However, as pointed out in (Prakken & Vreeswijk 2001), a different point view is also possible where "odd defeat loops are of an essentially different kind than even defeat loops". For instance, one might state that odd-length cycles are like paradoxes, i.e. situations where nothing can be believed, while even-length cycles are like dilemmas, i.e. situations where a choice needs to be made. According to this view, the arguments α, β, and γ should in a sense "annul" each other and lose the power of affecting other nodes, leaving then δ undefeated and, consequently, ϵ defeated. It has to be acknowledged that the choice of the "most appropriate" result is a matter of debate and may depend on case-specific considerations too (Horty 2002) and that some critical examples may be dealt with by applying some rationality postulates at the level of argument structure and construction (Caminada & Amgoud 2005). At an abstract level, it is however desirable that the alternative view presented above can be encompassed within the general SCC-recursive scheme. As it emerges from the above examples, this requires, first of all, the capability to distinguish some significant substructures (the self-defeating node in the first case, the three length-cycle in the second case) within a single SCC: this aspect is dealt with in the next section.

Decomposing a SCC into autonomous fragments

We follow the idea of identifying, within a SCC S, the subsets of nodes that can be considered "autonomously" in the incremental construction of extensions. These subsets will be called *autonomous fragments* and the set of autonomous fragments of S will be denoted as $\mathcal{AU}(S)$. A first intuitive requirement is that each fragment is strongly connected by itself and, while respecting this property, is as small as possible. Moreover, to be autonomous, fragments should not "interfere", namely should not intersect each other. To define significant minimal strongly connected fragments within a "conventional" SCC we need to modify the definition of path equivalence, substituting the clause that each node is always path-equivalent to itself with the requirement that the node is self-defeating.

Definition 11 *Given an argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and a set* $Q \subseteq \mathcal{A}$*, let* $\mathrm{AF}' = \mathrm{AF}{\downarrow_Q}$*. The binary relation of path-mutuality restricted to* Q*, denoted as* $PM_Q \subseteq (Q \times Q)$*, is defined as follows:*

- $\forall \alpha \in Q, (\alpha,\alpha) \in PM_Q$ *if and only if* $(\alpha,\alpha) \in \rightarrow$*;*
- *given two distinct nodes* $\alpha, \beta \in Q$*,* $(\alpha,\beta) \in PM_Q$ *if and only if in* AF' *there is a path from* α *to* β *and a path from* β *to* α*.*

We define the notion of *fragments* of a SCC S as follows.

Definition 12 *Given a non-empty argumentation framework* $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ *and a SCC* $S \in \mathrm{SCCS}_{\mathrm{AF}}$*, a non-empty set* $F \subseteq S$ *is called a fragment of* S *if and only if* $\forall \alpha, \beta \in F$*,*

$(\alpha, \beta) \in PM_F$. *The set of fragments of* S *is denoted as* $\mathcal{FR}(S)$.

Note that α and β are not necessarily distinct in Definition 12 and that the fragments belonging to $\mathcal{FR}(S)$ generally intersect each other. In particular, $S \in \mathcal{FR}(S)$ unless S consists of a unique non self-defeating argument, namely $S = \{\alpha\}$ and $\mathrm{AF}\downarrow_S = \langle\{\alpha\}, \emptyset\rangle$ (such a SCC will be called *monadic*). We have therefore the guarantee that $\mathcal{FR}(S) \neq \emptyset$ if S is not monadic.

The set $\mathcal{AU}(S)$ of autonomous fragments of a non monadic SCC S is derived from $\mathcal{FR}(S)$ by applying the following algorithm.

Definition of algorithm $\mathcal{AU}$

Step 1
`let` $\Sigma = \mathcal{FR}(S)$;
BEGIN MAIN LOOP
Step 2
`let` $\Sigma_{min} = \{F \in \Sigma \mid \nexists G \in \Sigma : G \subsetneq F\}$;
Step 3
`let` $\mathcal{AU}(S) = \{F \in \Sigma_{min} \mid \forall G \in \Sigma_{min} : G \neq F, F \cap G = \emptyset\}$;
Step 4
`if` $\mathcal{AU}(S) \neq \emptyset$
`then`
  EXIT;
`else`
  `let` $\Sigma = \Sigma \setminus \Sigma_{min}$;
  `goto Step 2;`
`endif`
END MAIN LOOP

In Step 1 the variable Σ is initialized to contain the set of all fragments of S. Then the algorithm enters a loop. In Step 2 the set Σ_{min} of the elements of Σ which are minimal with respect to set inclusion is identified, according to the intuition that autonomous fragments are as small as possible. In Step 3 the condition of non interference is verified, by selecting for inclusion into $\mathcal{AU}(S)$ the elements of Σ_{min} which have empty intersection with any other element of Σ_{min} (note that it may be the case that no such element exist in Σ_{min} at a given iteration of the main loop). If a non-empty $\mathcal{AU}(S)$ has been identified in Step 3, then in Step 4 the algorithm terminates, otherwise all elements of Σ_{min} are dropped from Σ and a new iteration of the main loop begins.

Proposition 3 *Let* S *be a non monadic SCC of an argumentation framework* AF*, then algorithm* $\mathcal{AU}$ *is guaranteed to terminate by producing a non-empty set* $\mathcal{AU}(S)$.

Recall that, since S is non monadic, $S \in \mathcal{FR}(S)$ and therefore $S \in \Sigma$ since the initialization of Σ in Step 1. Moreover, the finiteness of S ensures that Σ is finite. Then two situations may occur. If it holds that $\Sigma = \{S\}$, then clearly $\Sigma_{min} = \{S\}$ is assigned in Step 2 and $\mathcal{AU}(S) = \{S\}$ is assigned in Step 3, which determines algorithm termination. Otherwise $\Sigma \supsetneq \{S\}$ and $S \notin \Sigma_{min}$ since the other elements of Σ are proper subsets of S. Then two cases are possible: the algorithm terminates with a non-empty $\mathcal{AU}(S)$ or a new iteration begins after subtracting Σ_{min} from Σ. In the new iteration it still holds that $S \in \Sigma$ and we can iterate the same reasoning: then, the finiteness of Σ ensures that one of the two termination cases considered above is reached in a finite number of iterations. □

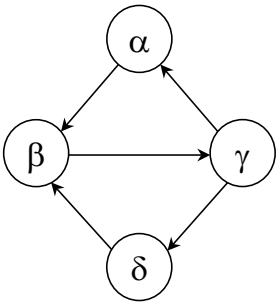


Figure 3: Two three-length cycles within a four-length cycle(AF_3).

To complete the definition of $\mathcal{AU}(S)$, the cases of a monadic SCC and of an empty SCC (which occurs only in an empty argumentation framework) have to be covered.

Definition 13 *Given an argumentation framework* $\mathrm{AF} = \langle\mathcal{A}, \rightarrow\rangle$ *and a SCC* $S \in \mathrm{SCCS}_{\mathrm{AF}}$*, the set of autonomous fragments of* S*, denoted as* $\mathcal{AU}(S)$*, is defined as follows:*

- $\mathcal{AU}(S) = \{S\}$, if S is monadic or $S = \emptyset$;
- $\mathcal{AU}(S)$ is the result of applying algorithm $\mathcal{AU}$ to S, otherwise.

By inspection of Step 3 of algorithm $\mathcal{AU}$, it can be noted that the elements of $\mathcal{AU}(S)$ are disjoint.

Let us now examine some examples of application of algorithm $\mathcal{AU}$. Every argumentation framework AF_i considered in the following consists of a single SCC S_i (i.e. $\mathrm{SCCS}_{\mathrm{AF}_i} = \{S_i\}$), which coincides with the set of all arguments of AF_i. In the case of AF_1 (Figure 1), $\mathcal{FR}(S_1) = \{\{\alpha\}, S_1\}$ and algorithm $\mathcal{AU}$ terminates in one iteration with $\mathcal{AU}(S_1) = \Sigma_{min} = \{\{\alpha\}\}$.

In the case of AF_2 (Figure 2), $\mathcal{FR}(S_2) = \{\{\alpha, \beta, \gamma\}, S_2\}$ and algorithm $\mathcal{AU}$ terminates in one iteration with $\mathcal{AU}(S_2) = \Sigma_{min} = \{\{\alpha, \beta, \gamma\}\}$.

Consider now AF_3 (Figure 3). $\mathcal{FR}(S_3) = \{\{\alpha, \beta, \gamma\}, \{\beta, \gamma, \delta\}, S_3\}$. Then in the first iteration of the main loop $\Sigma_{min} = \{\{\alpha, \beta, \gamma\}, \{\beta, \gamma, \delta\}\}$, and, since the intersection of the two elements of Σ_{min} is not empty, $\mathcal{AU}(S) = \emptyset$ in Step 3 and $\Sigma = \{S_3\}$ results in the `else` branch of Step 4. In the subsequent iteration, the algorithm terminates with $\mathcal{AU}(S_3) = \{S_3\} = \{\{\alpha, \beta, \gamma, \delta\}\}$.

In AF_4 (Figure 4), $\mathcal{FR}(S_4) = \{\{\alpha, \beta\}, \{\gamma, \delta\}, S_4\}$. Then in the first iteration of the main loop $\Sigma_{min} = \{\{\alpha, \beta\}, \{\gamma, \delta\}\}$ and, since the intersection of the two elements of Σ_{min} is empty, the algorithm terminates with $\mathcal{AU}(S_4) = \{\{\alpha, \beta\}, \{\gamma, \delta\}\}$.

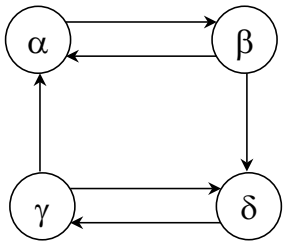


Figure 4: Two two-length cycles within a four-length cycle(AF_4).

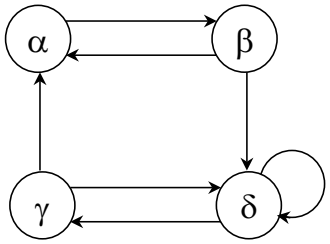


Figure 5: Two two-length cycles within a four-length cycle with a self-defeating node(AF_5).

In AF_5 (Figure 5), $\mathcal{FR}(S_5) = \{\{\delta\}, \{\alpha, \beta\}, \{\gamma, \delta\}, S_5\}$. Then in the first iteration of the main loop $\Sigma_{min} = \{\{\alpha, \beta\}, \{\delta\}\}$ and, since the intersection of the two elements of Σ_{min} is empty, the algorithm terminates with $\mathcal{AU}(S_5) = \{\{\alpha, \beta\}, \{\delta\}\}$.

Exploiting autonomous fragments in SCC-recursive semantics

Let us now continue our investigation by looking for a way to take into account the notion of autonomous fragment within the SCC-recursive scheme. Since this notion is introduced at the level of single SCCs, the most direct way is considering it within the definition of the base function, which operates at this level. Each base function $\mathcal{BF}$ considered in (Baroni, Giacomin, & Guida 2005) directly selects a set of subsets of a SCC S. An *$\mathcal{AU}$-aware* base function (denoted in the following as $\mathcal{BF}'$) should instead take into account the autonomous fragments of S. Our intuition is that each autonomous fragment represents the minimal topological unit to which some semantics-specific principle can be applied for extension construction. Therefore, we suggest that a semantics-specific *fragment function* $\mathcal{FF}$ is applied to each autonomous fragment F and returns a set of subsets of F. Each of these subsets is regarded as an elementary building block in extension construction. Moreover, if $\mathcal{AU}(S) = \{S\}$ the result should be the same as in the non *$\mathcal{AU}$-aware* case, therefore $\mathcal{FF}$ should be equal to $\mathcal{BF}$ in this case. More articulated considerations need to be applied when $|\mathcal{AU}(S)| \neq 1$ or $\mathcal{Y}(\mathcal{AU}(S)) \neq S$ (where $\mathcal{Y}(Q) \triangleq \bigcup_{P \in Q} P$, given that Q is a set of sets). Let us consider orderly these cases.

If $|\mathcal{AU}(S)| \neq 1$, several autonomous fragments are considered separately and, for each autonomous fragment F_i, a set of subsets of F_i is produced by $\mathcal{FF}$. They need then to be combined: the most direct way is considering all possible combinations of these subsets except those which infringe the conflict-free principle. In other words, all possible conflict-free combinations obtained by selecting one subset for each F_i are considered.

If $\mathcal{Y}(\mathcal{AU}(S)) = S$, i.e. the whole S has been considered since autonomous fragments are a partition of S, the above mentioned combinations represent the result of the application of $\mathcal{BF}'$ to S. Otherwise, there are some elements of S which are not included in any autonomous fragment. We follow the idea that the inclusion in the extensions of the other elements of S should be determined by the choices carried out within the autonomous fragments $\mathcal{AU}(S)$. In a sense, autonomous fragments are evaluated first, then the results of this evaluation are taken into account when the remaining elements of S (if any) are considered.

In line with the fundamental principles of SCC-recursiveness, this amounts to invoke recursively the general function $\mathcal{GF}$ on a restricted argumentation framework, derived taking into account the choices in $\mathcal{AU}(S)$.

Having provided an outline of the underlying ideas, we need to put them in formal terms.

Let $\mathcal{BF}_\mathcal{S}$ be the base function of a SCC-recursive semantics $\mathcal{S}$; the corresponding *$\mathcal{AU}$-aware* base function $\mathcal{BF}'_\mathcal{S}$ is defined as follows.

Definition 14 *Given a SCC-recursive semantics $\mathcal{S}$ with base function $\mathcal{BF}_\mathcal{S}(\mathrm{AF}, C)$, the corresponding $\mathcal{AU}$-*aware *base function $\mathcal{BF}'_\mathcal{S}(\mathrm{AF}, C)$ is a function that given an argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ such that $|\mathrm{SCCS}_{\mathrm{AF}}| = 1$ (i.e. $\mathrm{SCCS}_{\mathrm{AF}} = \{\mathcal{A}\}$) and a set $C \subseteq \mathcal{A}$, gives a subset of $2^\mathcal{A}$ as follows:*
$E \in \mathcal{BF}'_\mathcal{S}(\mathrm{AF}, C)$ if and only if E is conflict-free and $(E \cap \mathcal{Y}(\mathcal{AU}(\mathcal{A}))) \in \mathcal{UCF}_\mathcal{S}(\mathcal{AU}(\mathcal{A}), \mathrm{AF}, C)$ and if $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) \neq \emptyset$, $\forall S \in \mathrm{SCCS}_{\mathrm{AF}\downarrow_{\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A}))}}$ $(E \cap S) \in \mathcal{GF}'(\mathrm{AF}\downarrow_{UP_{\mathrm{AF}}(S,E)}, U_{\mathrm{AF}}(S, E) \cap C)$

where

- $\mathcal{UCF}_\mathcal{S}(\Sigma, \mathrm{AF}, C)$ is a function which given a set Σ of disjoint subsets of $\mathcal{A}$ returns a set of subsets of $\mathcal{Y}(\Sigma)$ as follows: $E \in \mathcal{UCF}_\mathcal{S}(\Sigma, \mathrm{AF}, C)$ if E is conflict-free and $\forall Q \in \Sigma$, $E \cap Q = \mathcal{BF}_\mathcal{S}(\mathrm{AF}\downarrow_Q, C \cap Q)$;
- $\mathcal{GF}'$ is the general recursive function in the *$\mathcal{AU}$-aware* scheme (see Definition 15 below).

Definition 14 is rather complex and not really elegant. This reflects the preliminary state of this investigation: devising a simpler formulation is one of the directions of future work. To illustrate its main features we note that:

- the base function $\mathcal{BF}$ is applied to each autonomous fragment and the resulting sets are combined in all possible conflict-free manners (through function $\mathcal{UCF}$);
- the output of $\mathcal{UCF}$ is used directly as output of $\mathcal{BF}'$ if the union of all autonomous fragments $\mathcal{Y}(\mathcal{AU}(\mathcal{A}))$ completely covers the SCC $\mathcal{A}$, since in this case $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) = \emptyset$ and therefore the second part of the definition does not apply;
- in particular, the *$\mathcal{AU}$-aware* base function $\mathcal{BF}'$ coincides with $\mathcal{BF}$ when there is only one autonomous fragment coinciding with the SCC $\mathcal{A}$ itself, since $\mathcal{UCF}_{\mathcal{S}}(\{\mathcal{A}\}, \mathrm{AF}, C)$ is invoked in this case, leading to $E \cap \mathcal{A} = \mathcal{BF}_{\mathcal{S}}(\mathrm{AF}{\downarrow_{\mathcal{A}}}, C \cap \mathcal{A}) = \mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$;
- if $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) \neq \emptyset$, the construction of the output of $\mathcal{BF}'$ proceeds recursively using the output of $\mathcal{UCF}$ as starting point and invoking the general SCC-recursive function $\mathcal{GF}'$ "as usual" on the parts of $\mathcal{A}$ not covered by $\mathcal{Y}(\mathcal{AU}(\mathcal{A}))$, i.e. on the SCCs of the restricted argumentation framework $\mathrm{AF}{\downarrow_{\mathcal{A}\setminus\mathcal{Y}(\mathcal{AU}(\mathcal{A}))}}$.

Besides complications, taking into account the internal structure of SCCs has another downside: there are cases where $\mathcal{BF}'_{\mathcal{S}}(\mathrm{AF}, C) = \emptyset$. This may happen, for instance, when $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) = \emptyset$ and $\mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(\mathcal{A}), \mathrm{AF}, C) = \emptyset$, i.e. the autonomous fragments cover the whole SCC and there are no conflict-free combinations of the subsets selected within them. This would lead to the unpleasant situation of non-existence of extensions for some argumentation frameworks. A (still not elegant) solution consists in introducing a provision for this case in the semantics definition.

Definition 15 *Given a SCC-recursive semantics $\mathcal{S}$ with base function $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}, C)$, the corresponding $\mathcal{AU}$-*aware *semantics $\mathcal{S}'$ is defined as follows: for any argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$, $\mathcal{E}_{\mathcal{S}'}(\mathrm{AF}) = \mathcal{GF}'(\mathrm{AF}, \mathcal{A})$, where for any $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ and for any set $C \subseteq \mathcal{A}$, the function $\mathcal{GF}'(\mathrm{AF}, C) \subseteq 2^{\mathcal{A}}$ is defined as follows:*
for any $E \subseteq \mathcal{A}$, $E \in \mathcal{GF}'(\mathrm{AF}, C)$ if and only if

- *in case $|\mathrm{SCCS}_{\mathrm{AF}}| = 1$, $E \in \mathcal{BF}^*_{\mathcal{S}}(\mathrm{AF}, C)$*
- *otherwise, $\forall S \in \mathrm{SCCS}_{\mathrm{AF}}$ $(E \cap S) \in \mathcal{GF}'(\mathrm{AF}{\downarrow_{UP_{\mathrm{AF}}(S,E)}}, U_{\mathrm{AF}}(S, E) \cap C)$*

where $\mathcal{BF}^*_{\mathcal{S}}(\mathrm{AF}, C) = \mathcal{BF}'_{\mathcal{S}}(\mathrm{AF}, C)$ if $\mathcal{BF}'_{\mathcal{S}}(\mathrm{AF}, C) \neq \emptyset$, $\mathcal{BF}^*_{\mathcal{S}}(\mathrm{AF}, C) = \{\emptyset\}$ otherwise.

Essentially an *$\mathcal{AU}$-aware* semantics is just a SCC-recursive semantics with a special base function $\mathcal{BF}^*$, which, apart a particular case, coincides with the *$\mathcal{AU}$-aware* base function $\mathcal{BF}'$. In turn, $\mathcal{BF}'$ exploits the original base function $\mathcal{BF}$ in the cases where a SCC does not admit significant autonomous fragments.

Proving fundamental properties of *$\mathcal{AU}$-aware* semantics turns out to be relatively easy, in virtue of its adherence to the general SCC-recursive scheme, whose properties are analyzed in (Baroni, Giacomin, & Guida 2005).

Let us consider well-foundedness of recursion first.

Proposition 4 *Recursion in Definition 14 is well-founded.*

Definition 14 involves an indirect recursion: it invokes the $\mathcal{GF}'$ function on each SCC S of $\mathrm{AF}{\downarrow_{\mathcal{A}\setminus\mathcal{Y}(\mathcal{AU}(\mathcal{A}))}}$ and such function in turn invokes the *$\mathcal{AU}$-aware* base function $\mathcal{BF}'_{\mathcal{S}}$ in the first branch of Definition 15. To verify the well-foundedness of this indirect recursion, first note that since $\mathcal{AU}(\mathcal{A}) \neq \emptyset$ (Proposition 3) and, by Definition 12, $\forall F \in \mathcal{AU}(\mathcal{A})$ $F \neq \emptyset$, it turns out that $\mathcal{Y}(\mathcal{AU}(\mathcal{A})) \neq \emptyset$. As a consequence, the restricted argumentation framework considered in the recursive branch $\mathrm{AF}{\downarrow_{\mathcal{A}\setminus\mathcal{Y}(\mathcal{AU}(\mathcal{A}))}}$ (and therefore also any of its SCCs) has a strictly lesser number of arguments than $|\mathcal{A}|$. Observe also that if $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) = \emptyset$, the recursive part of Definition 14 is not invoked. This implies that subsequent invocations (if any) of the recursive branch of $\mathcal{BF}'_{\mathcal{S}}$ (reached through $\mathcal{GF}'$) operate on progressively smaller non-empty argumentation frameworks. Due to the hypothesis of finiteness of $\mathcal{A}$, this leads to consider the case where $\mathcal{BF}'_{\mathcal{S}}$ is invoked on an argumentation framework $\mathrm{AF} = \langle \mathcal{A}, \rightarrow \rangle$ such that $|\mathcal{A}| = 1$. Such argumentation framework consists necessarily of a unique SCC: by Definition 13, in this case it holds $\mathcal{AU}(\mathcal{A}) = \{\mathcal{A}\}$, and, as a consequence, $\mathcal{A} \setminus \mathcal{Y}(\mathcal{AU}(\mathcal{A})) = \emptyset$ which represents the non-recursive case of Definition 14. □

Proposition 5 *The definition of an $\mathcal{AU}$-*aware *SCC-recursive semantics $\mathcal{S}'$ is well-founded.*

The definition of an *$\mathcal{AU}$-aware* SCC-recursive semantics is a case of the general SCC-recursive scheme with a well-defined base function (Proposition 4). Then the conclusion directly derives from the properties of the general SCC-recursive scheme shown in (Baroni, Giacomin, & Guida 2005). □

In the same line, we now show that an *$\mathcal{AU}$-aware* SCC-recursive semantics $\mathcal{S}'$ shares with its non *$\mathcal{AU}$-aware* version the fundamental properties of being conflict-free and agreeing with grounded semantics.

Proposition 6 *Any $\mathcal{AU}$-*aware *SCC-recursive semantics $\mathcal{S}'$ satisfies the conflict-free property.*

As recalled in Proposition 1, for any SCC-recursive semantics $\mathcal{T}$, if its base function $\mathcal{BF}_{\mathcal{T}}$ is conflict-free then $\mathcal{T}$ satisfies the conflict-free property. Since $\mathcal{S}'$ is a special case of SCC-recursive semantics, where $\mathcal{BF}^*_{\mathcal{S}}$ plays the role of $\mathcal{BF}_{\mathcal{T}}$, it is sufficient to show that $\mathcal{BF}^*_{\mathcal{S}}$ is conflict free, namely that all elements of $\mathcal{BF}^*_{\mathcal{S}}(\mathrm{AF}, C)$ are conflict free. By Definition 15, this in turn corresponds to require that $\mathcal{BF}'_{\mathcal{S}}(\mathrm{AF}, C)$ is conflict-free, which holds by Definition 14. □

As to agreement with grounded semantics, it has to be verified that if the sufficient condition for agreement stated in Proposition 2 is satisfied by $\mathcal{BF}$, then it is satisfied also by $\mathcal{BF}^*$.

Proposition 7 *If a SCC-recursive semantics $\mathcal{S}$ satisfies the hypothesis of Proposition 2, the corresponding $\mathcal{AU}$-*aware *semantics $\mathcal{S}'$ satisfies it as well.*

We need to show that $\mathcal{BF}^*_{\mathcal{S}}(\langle\{\alpha\}, \emptyset\rangle, \{\alpha\}) = \{\{\alpha\}\}$. This immediately follows from the fact that $\langle\{\alpha\}, \emptyset\rangle$

is an argumentation framework consisting of a single monadic SCC and therefore $\mathcal{BF}^*_{\mathcal{S}}(\langle\{\alpha\},\emptyset\rangle,\{\alpha\}) = \mathcal{BF}'_{\mathcal{S}}(\langle\{\alpha\},\emptyset\rangle,\{\alpha\}) = \mathcal{BF}_{\mathcal{S}}(\langle\{\alpha\},\emptyset\rangle,\{\alpha\}) = \{\{\alpha\}\}$. □

Putting $\mathcal{AU}$-*aware* semantics at work

Having shown that, despite its rather articulated form, our attempt to define $\mathcal{AU}$-*aware* semantics preserves the fundamental properties that are desirable for any semantics, its practical impact remains to be analyzed. First, let us remark that in all the examples discussed in (Baroni, Giacomin, & Guida 2005) every SCC S is such that $\mathcal{AU}(S) = \{S\}$ and therefore no differences emerge when considering an $\mathcal{AU}$-*aware* semantics wrt. its non $\mathcal{AU}$-*aware* version. Let us now review the motivating examples introduced above, examining the behavior of the $\mathcal{AU}$-*aware* versions of preferred and $CF2$ semantics, denoted as $\mathcal{PR}'$ and $CF2'$ respectively.

Since all examples involve an argumentation framework AF_i consisting of a single SCC S_i, Definition 15 directly leads to consider the following invocation of the $\mathcal{AU}$-*aware* base function: $\mathcal{E}_{\mathcal{S}'}(\mathrm{AF}_i) = \mathcal{BF}'_{\mathcal{S}}(\mathrm{AF}_i, S_i)$.

Example 1. Let us start with AF_1 (Figure 1), recalling that $\mathcal{AU}(S_1) = \{\{\alpha\}\}$ and, therefore, $\mathcal{Y}(\mathcal{AU}(S_1)) = \{\alpha\}$. Since $\mathcal{Y}(\mathcal{AU}(S_1)) \subsetneq S_1$ both parts of Definition 14 apply. The first part states that for any extension E, $E \cap \mathcal{Y}(\mathcal{AU}(S_1)) = \mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_1), \mathrm{AF}_1, S_1)$. Since $\mathcal{AU}(S_1)$ contains just one element, $\mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_1), \mathrm{AF}_1, S_1) = \mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_1{\downarrow}_{\{\alpha\}}, \{\alpha\})$. Since $\mathrm{AF}_1{\downarrow}_{\{\alpha\}}$ consists of a single self-defeating argument and $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_1{\downarrow}_{\{\alpha\}}, \{\alpha\}) = \{\emptyset\}$ either with $\mathcal{S} = CF2$ or $\mathcal{S} = \mathcal{PR}$, it turns out that $E \cap \{\alpha\} = \emptyset$. In words, α can not be included in any extension.

Then, according to the second part of Definition 14, $E \cap T$ is computed recursively for all $T \in \mathrm{SCCS}_{\mathrm{AF}_1{\downarrow}_{S_1\setminus\{\alpha\}}} = \{\{\beta\},\{\gamma\}\}$. Following the SCC order within $\mathrm{AF}_1{\downarrow}_{S_1\setminus\{\alpha\}}$, $\{\beta\}$ has to be considered first, yielding $E \cap \{\beta\} = \mathcal{GF}'(\mathrm{AF}_1{\downarrow}_{UP_{\mathrm{AF}_1}(\{\beta\},E)}, U_{\mathrm{AF}_1}(\{\beta\},E) \cap S_1)$.Since $E \cap \mathcal{Y}(\mathcal{AU}(S_1)) = E \cap \{\alpha\} = \emptyset$, it turns out that β is not attacked by E nor is defended by E from the attack coming from α, therefore $UP_{\mathrm{AF}_1}(\{\beta\},E) = \{\beta\}$ and $U_{\mathrm{AF}_1}(\{\beta\},E) = \emptyset$. As a consequence, $E \cap \{\beta\} = \mathcal{GF}'(\mathrm{AF}_1{\downarrow}_{\{\beta\}}, \emptyset)$, which by the first clause of Definition 14 yields $E \cap \{\beta\} = \mathcal{BF}'_{\mathcal{S}'}(\langle\{\beta\},\emptyset\rangle,\emptyset)$, resulting in $E \cap \{\beta\} = \{\beta\}$ with $\mathcal{S} = CF2$, and in $E \cap \{\beta\} = \emptyset$ with $\mathcal{S} = \mathcal{PR}$.

Turning to $\{\gamma\}$, we have $E \cap \{\gamma\} \in \mathcal{GF}'(\mathrm{AF}_1{\downarrow}_{UP_{\mathrm{AF}_1}(\{\gamma\},E)}, U_{\mathrm{AF}_1}(\{\gamma\},E) \cap S_1)$. In the case of $CF2$ semantics, since $E \cap \{\beta\} = \{\beta\}$ for any E, we have $UP_{\mathrm{AF}_1}(\{\gamma\},E) = U_{\mathrm{AF}_1}(\{\gamma\},E) = \emptyset$, which skipping some further purely formal steps leads to consider the empty argumentation framework and therefore to conclude $E \cap \{\gamma\} = \emptyset$. In the case of preferred semantics, it holds that $UP_{\mathrm{AF}_1}(\{\gamma\},E) = \{\gamma\}$, $U_{\mathrm{AF}_1}(\{\gamma\},E) = \emptyset$, which (skipping again some steps) gives $E \cap \{\gamma\} = \emptyset$.

Summing up, we obtain $\mathcal{E}_{CF2'}(\mathrm{AF}_1) = \{\{\beta\}\}$, while $\mathcal{E}_{\mathcal{PR}'}(\mathrm{AF}_1) = \emptyset$. This shows that the $\mathcal{AU}$-*aware* version of $CF2$ semantics provides a different (and intuitively more acceptable) result wrt. the non $\mathcal{AU}$-*aware* one.

Example 2. In the case of AF_2 (Figure 2), $\mathcal{AU}(S_2) = \{\{\alpha,\beta,\gamma\}\}$. As to the first part of Definition 14, for any extension E, $E \cap \mathcal{Y}(\mathcal{AU}(S_2)) = \mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_2), \mathrm{AF}_2, S_2) = \mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_2{\downarrow}_{\{\alpha,\beta,\gamma\}}, \{\alpha,\beta,\gamma\})$. Here the two semantics differ since in the case of preferred semantics $E \cap \{\alpha,\beta,\gamma\} \in \mathcal{BF}_{\mathcal{PR}}(\mathrm{AF}_2{\downarrow}_{\{\alpha,\beta,\gamma\}}, \{\alpha,\beta,\gamma\}) = \{\emptyset\}$, while in the case of $CF2$-semantics $E \cap \{\alpha,\beta,\gamma\} \in \mathcal{BF}_{CF2}(\mathrm{AF}_2{\downarrow}_{\{\alpha,\beta,\gamma\}}, \{\alpha,\beta,\gamma\}) = \{\{\alpha\},\{\beta\},\{\gamma\}\}$. In both cases, $E \cap T$ has to be computed recursively for all $T \in \mathrm{SCCS}_{\mathrm{AF}_2{\downarrow}_{S_2\setminus\{\alpha,\beta,\gamma\}}} = \{\{\delta\},\{\epsilon\}\}$, on the basis of the choices carried out for $E \cap \{\alpha,\beta,\gamma\}$.

Let us examine the case of preferred semantics first, where there is just one choice for $E \cap \{\alpha,\beta,\gamma\} = \emptyset$. Following the SCC order within $\mathrm{AF}_2{\downarrow}_{S_2\setminus\{\alpha,\beta,\gamma\}}$, $\{\delta\}$ has to be considered first. Since $\{\delta\}$ is exactly in the same situation as $\{\beta\}$ in Example 1, skipping the analogous formal steps made explicit in Example 1 we obtain $E \cap \{\delta\} = \emptyset$. Consequently, when considering the subsequent SCC of $\mathrm{AF}_2{\downarrow}_{S_2\setminus\{\alpha,\beta,\gamma\}}$, namely $\{\epsilon\}$, we are in a completely analogous situation as for $\{\gamma\}$ in Example 1 and we obtain $E \cap \{\epsilon\} = \emptyset$.

In summary, we obtain $\mathcal{E}_{\mathcal{PR}'}(\mathrm{AF}_2) = \mathcal{E}_{\mathcal{PR}}(\mathrm{AF}_2) = \{\emptyset\}$.

Let us turn to $CF2$-semantics, where there are three choices for $E \cap \{\alpha,\beta,\gamma\}$, namely $\{\alpha\}$, $\{\beta\}$, and $\{\gamma\}$, each being the starting point for the construction of one or more extensions, to be completed by possibly adding elements of $\mathrm{AF}_2{\downarrow}_{S_2\setminus\{\alpha,\beta,\gamma\}}$.

Consider first the case where $E \cap \{\alpha,\beta,\gamma\} = \{\alpha\}$. We note that in this case E defends δ by attacking β, and therefore $UP_{\mathrm{AF}_2}(\{\delta\},E) = U_{\mathrm{AF}_2}(\{\delta\},E) = \{\delta\}$. Thus δ and ϵ are in the same situation as β and γ respectively in AF_1. This leads to $E \cap \{\delta\} = \{\delta\}$ and $E \cap \{\epsilon\} = \emptyset$, obtaining a first extension $E_1 = \{\alpha,\delta\}$.

Let us now examine the case $E \cap \{\alpha,\beta,\gamma\} = \{\beta\}$. In this case E attacks δ, and therefore $UP_{\mathrm{AF}_2}(\{\delta\},E) = U_{\mathrm{AF}_2}(\{\delta\},E) = \emptyset$; skipping some purely formal steps this clearly leads to $E \cap \{\delta\} = \emptyset$. As a consequence, it turns out that $UP_{\mathrm{AF}_2}(\{\epsilon\},E) = U_{\mathrm{AF}_2}(\{\epsilon\},E) = \{\epsilon\}$, which leads to $E \cap \{\epsilon\} = \{\epsilon\}$, thus obtaining a second extension $E_2 = \{\beta,\epsilon\}$.

Finally, assume $E \cap \{\alpha,\beta,\gamma\} = \{\gamma\}$. In this case E neither attacks nor defends δ, therefore $U_{\mathrm{AF}_2}(\{\delta\},E) = \emptyset$, while $UP_{\mathrm{AF}_2}(\{\delta\},E) = \{\delta\}$. Since only $UP_{\mathrm{AF}_2}(\{\delta\},E)$ is relevant in the definition of $\mathcal{BF}_{CF2}$, this gives rise to $E \cap \{\delta\} = \{\delta\}$ and, consequently, $E \cap \{\epsilon\} = \emptyset$, obtaining a third extension $E_3 = \{\gamma,\delta\}$.

Summing up, $\mathcal{E}_{CF2'}(\mathrm{AF}_2) = \{\{\alpha,\delta\},\{\beta,\epsilon\},\{\gamma,\delta\}\} \neq \mathcal{E}_{CF2}(\mathrm{AF}_2) = \{\{\alpha,\delta\},\{\alpha,\epsilon\},\{\beta,\epsilon\},\{\gamma,\delta\}\}$. As to the justification status of arguments, the difference does not manifest itself, since, according to the $\mathcal{AU}$-*aware* version of $CF2$ semantics too, no argument is included in all extensions. It is however interesting that the extension $\{\alpha,\epsilon\}$ is not prescribed by the $\mathcal{AU}$-*aware* version of $CF2$ semantics, as not compatible with the idea of choosing first within the autonomous fragment $\{\alpha,\beta,\gamma\}$ and then propagating the effects on the rest of the argumentation framework.

Neither the $\mathcal{AU}$-*aware* version of preferred semantics nor of $CF2$ captures the intuition underlying the example that the three-length cycle $\{\alpha,\beta,\gamma\}$ could be regarded as a sort of "null element", leaving δ undefeated and γ defeated. The

search for a semantics featuring this kind of behavior remains open.

Example 3. Consider now AF_3 (Figure 3). Since $\mathcal{AU}(S_3) = \{S_3\}$ the behavior of any non *$\mathcal{AU}$-aware* semantics and of its *$\mathcal{AU}$-aware* version is the same and therefore will not be discussed here.

Example 4. In AF_4 (Figure 4), $\mathcal{AU}(S_4) = \{\{\alpha,\beta\},\{\gamma,\delta\}\}$. In this case $\mathcal{Y}(\mathcal{AU}(S_4)) = S_4$, and therefore $\mathcal{E}_{\mathcal{S}'}(\mathrm{AF}_4) = \mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_4), \mathrm{AF}_4, S_4)$. This means that first the base function $\mathcal{BF}_{\mathcal{S}}$ is evaluated separately for $\mathrm{AF}_4{\downarrow}_{\{\alpha,\beta\}}$ and $\mathrm{AF}_4{\downarrow}_{\{\gamma,\delta\}}$. Both fragments consist of a couple of rebutting arguments, a prototypical case often referred to as "Nixon diamond" where any multiple-status semantics admits two extensions, each corresponding to the choice of one of the arguments. We have therefore $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_4{\downarrow}_{\{\alpha,\beta\}}, S_4 \cap \{\alpha,\beta\}) = \{\{\alpha\},\{\beta\}\}$, and $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_4{\downarrow}_{\{\gamma,\delta\}}, S_4 \cap \{\gamma,\delta\}) = \{\{\gamma\},\{\delta\}\}$ both with $\mathcal{S} = \mathcal{PR}$ and $\mathcal{S} = CF2$. All conflict free combinations of the elements of $\{\{\alpha\},\{\beta\}\}$ and $\{\{\gamma\},\{\delta\}\}$ are then returned by function $\mathcal{UCF}$, yielding $\mathcal{E}_{\mathcal{S}'}(\mathrm{AF}_4) = \{\{\alpha,\delta\},\{\beta,\gamma\}\} = \mathcal{E}_{\mathcal{S}}(\mathrm{AF}_4)$ both with $\mathcal{S} = \mathcal{PR}$ and $\mathcal{S} = CF2$. Therefore, in this case the use of *$\mathcal{AU}$-aware* semantics does not give rise to different results.

Example 5. Differences appear instead in AF_5 (Figure 5), where $\mathcal{AU}(S_5) = \{\{\alpha,\beta\},\{\delta\}\}$.

First, $\mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_5), \mathrm{AF}_5, S_5)$ has to be evaluated, which requires in turn evaluating $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_5{\downarrow}_{Q}, S_5 \cap Q)$ for $Q \in \{\{\alpha,\beta\},\{\delta\}\}$. We have $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_5{\downarrow}_{\{\alpha,\beta\}}, S_5 \cap \{\alpha,\beta\}) = \{\{\alpha\},\{\beta\}\}$, and $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_5{\downarrow}_{\{\delta\}}, S_5 \cap \{\delta\}) = \{\emptyset\}$ both with $\mathcal{S} = \mathcal{PR}$ and $\mathcal{S} = CF2$.

Considering the conflict free combinations, we derive $\mathcal{UCF}_{\mathcal{S}}(\mathcal{AU}(S_5), \mathrm{AF}_5, S_5) = \{\{\alpha\},\{\beta\}\}$.

Therefore, with both semantics, we have two starting choices for $E \cap \mathcal{Y}(\mathcal{AU}(S_5))$, namely $\{\alpha\}$ and $\{\beta\}$. The restricted argumentation framework $\mathrm{AF}_5{\downarrow}_{\{\gamma\}}$ remains to be considered, which clearly consists of a single SCC $\{\gamma\}$. Consider first the case $E \cap \mathcal{Y}(\mathcal{AU}(S_5)) = \{\alpha\}$; since E does not attack γ and γ defends itself against the attack coming from δ, we have $UP_{\mathrm{AF}_5}(\{\gamma\}, E) = U_{\mathrm{AF}_5}(\{\gamma\}, E) = \{\gamma\}$, and therefore we obtain $\mathcal{BF}_{\mathcal{S}}(\mathrm{AF}_5{\downarrow}_{\{\gamma\}}, S_5 \cap \{\gamma\}) = \{\gamma\}$, both with $\mathcal{S} = \mathcal{PR}$ and $\mathcal{S} = CF2$. This leads to consider $E = \{\alpha,\gamma\}$ which, not being conflict free, is not compatible with Definition 14 and is discarded.

The development of the case $E \cap \mathcal{Y}(\mathcal{AU}(S_5)) = \{\beta\}$ is analogous and leads to consider $E = \{\beta,\gamma\}$ which is conflict free and compatible with Definition 14.

In summary, $\mathcal{E}_{\mathcal{S}'}(\mathrm{AF}_5) = \{\{\beta,\gamma\}\}$ both with $\mathcal{S} = \mathcal{PR}$ and $\mathcal{S} = CF2$. Note that this is the same result as for non *$\mathcal{AU}$-aware* preferred semantics, since $\mathcal{E}_{\mathcal{PR}}(\mathrm{AF}_5) = \{\{\beta,\gamma\}\}$ while a difference appears for $CF2$ semantics where $\mathcal{E}_{CF2}(\mathrm{AF}_5) = \{\{\beta,\gamma\},\{\alpha\}\}$.

Thus the *$\mathcal{AU}$-aware* version of $CF2$ semantics agrees with preferred semantics in this case (while the non *$\mathcal{AU}$-aware* version does not) and achieves a behavior which is intuitively plausible if self-defeating arguments are considered as "null elements" in an argumentation framework.

Conclusions

We have provided an initial investigation about the potential use of the novel notion of autonomous fragments within SCC-recursive argumentation semantics. The presented results are quite preliminary and further work is needed in order to improve the definition of *$\mathcal{AU}$-aware* semantics and explore more deeply its properties. Though our analysis has started from specific examples, we remark that the aim of the paper is not to achieve a "better" treatment of particular cases but rather to suggest an interesting perspective about argumentation semantics design. In fact, it emerges that different solutions are obtained by changing the semantics behavior with respect to topology, e.g. choosing between the *$\mathcal{AU}$-aware* and the non *$\mathcal{AU}$-aware* version of a semantics, without affecting the underlying notion of extension, represented by the base function $\mathcal{BF}$. This suggest that the "design" of an argumentation semantics can be conceived as composed, in a modular way, by the answers to two "orthogonal" questions: i) how to take into account the defeat graph topology in extension construction and ii) which principles rule the identification of extensions within the basic topological entities considered. Identifying alternative answers to these questions and properly combining and comparing them appears to be a very interesting research line to pursue.

References

Baroni, P., and Giacomin, M. 2003. Solving semantic problems with odd-length cycles in argumentation. In *Proc. of the 7th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2003)*, 440–451.

Baroni, P., and Giacomin, M. 2004a. A general recursive schema for argumentation semantics. In *Proc. of the 16th European Conf. on Artificial Intelligence (ECAI 2004)*, 783–787.

Baroni, P., and Giacomin, M. 2004b. A recursive approach to argumentation: motivation and perspectives. In *Proc. of the 10th Int. Workshop on Non-Monotonic Reasoning (NMR 2004)*, 50–58.

Baroni, P.; Giacomin, M.; and Guida, G. 2005. Scc-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence* 168(1-2):165–210.

Caminada, M., and Amgoud, L. 2005. An axiomatic account of formal argumentation. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 608–613.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence* 77(2):321–357.

Horty, J. F. 2002. Skepticism and floating conclusions. *Artificial Intelligence* 135(1–2):55–72.

Pollock, J. L. 1992. How to reason defeasibly. *Artificial Intelligence* 57(1):1–42.

Prakken, H., and Vreeswijk, G. A. W. 2001. Logics for defeasible argumentation. In Gabbay, D. M., and Guenthner, F., eds., *Handbook of Philosophical Logic, Second Edition*. Dordrecht: Kluwer Academic Publishers.

6 Belief Change and Updates

Belief change and nonmonotonic reasoning go hand in hand. As the area has matured, research in belief change has taken several interesting turns and seen many new developments. Moreover, it has permitted researchers to look upon other areas of research with a new perspective. As a way of capturing commonsense reasoning, for instance, belief change provides a unique perspective. Belief change has both benefitted from insights adapted from nonmonotonic reasoning while at the same time sharing its own unique view on reasoning to influence research directions in this area.

The specialized session on Belief Change and Updates is a one-day event bringing together researchers interested in the area of belief change and its relationship with nonmonotonic reasoning. The goal is to promote further development of the theory of belief change and explore relationships with other areas of research, particularly nonmonotonic reasoning but also knowledge representation and reasoning, logics of belief and knowledge, and artificial intelligence in general.

This session attracted nine excellent contributions to the field. Each paper was reviewed by at least two members of the session's program committee. All papers were deemed acceptable for publication although one contribution was considered only mildly related to the aims of the session.

Jerôme Lang's contribution *About Time, Revision and Update* challenges the often quoted view that *belief revision* deals with static environments while *belief update* deals with dynamic environments. His proposal maintains that belief revision deals with observations that provide new information about the past, present and future. On the other hand, belief update deals with new information as the result of ontic actions. However, he notes that it is not possible to capture the effects of all ontic actions when belief update is required to satisfy some of the postulates proposed by Katsuno and Mendelzon in 1992. Eduardo Ferme and Martin Krevneris' paper on *Ensconcement Based Contraction's Axiomatic: Preliminary Version* provides an axiomatic characterisation for a form of belief change based on ensconcements developed by Williams in 1992. This class of operators deals with change in belief bases. In their paper on *Elaborating Domain Descriptions* Andreas Herzig, Laurent Perrussel and Ivan Varzinczak provide a mechanism for elaborating action theories in propositional dynamic logic (PDL). Their proposal uses belief contraction to elaborate domain theories once discrepancies in the effects of actions are encountered. Richard Booth, Souhila Kaci and Leendert van der Torre's paper on *Merging Rules* investigates the idea of merging conditional statements. It achieves this by unifying approaches to belief merging. They look at notions based on consistency and on implication to show how such rules can be merged. The paper by Julien Seinturier, Pierre Drap and Odile Papini on *A Reversible Framework for Propositional Bases Merging* develops a methodology for merging belief bases using polynomials. A feature of this approach is that it allows the merging process to be reversed. Laurent Perrussel, Jean-Marc Thevenin and Thomas Meyer's contribution *A Mutual Enrichment for Agents Through Nested Belief Change: A Semantic Approach* investigates the interchange of nested beliefs among multiple agents and the subsequent revision that occurs on these beliefs. In this way, nested beliefs and agent preferences are communicated among an agent community allowing for richer forms of interaction. In a paper entitled *Getting Possibilities from the Impossible* Corinna Elsenbroich, Dov Gabbay and Odinaldo Rodrigues investigate a notion of abductive reasoning that is tolerant of inconsistencies in the initial knowledge base. Traditional abduction attempts to determine an explanation ε for an observation given a knowledge base Δ such that $\Delta \cup \varepsilon \nvdash \bot$. However, it is incapable of appropriately dealing with the case where the initial knowledge base Δ is itself inconsistent. This paper addresses this important problem by ensuring that the explanation occurs in all maximal consistent subsets of the

knowledge base. In a paper on *Rethinking Semantics of Dynamic Logic Programming* Jan Sefranek considers updating of logic programs. He suggests replacing the *Causal Rejection Principle*—in case of conflicts, reject certain of the conflicting rules—by an approach that ignores dependencies between conflicting rules. Furthermore, it applies to both ordered and non-ordered information.

Session chairs

Andreas Herzig
(Andreas.Herzig@irit.fr)

Maurice Pagnucco
(morri@cse.unsw.edu.au)

Program committee

Salem Benferhat
(benferhat@cril.univ-artois.fr)

Alexander Bochman
(bochmana@hait.ac.il)

Richard Booth
(ribooth@gmail.com)

John Cantwell
(cantwell@infra.kth.se)

Samir Chopra
(schopra@sci.brooklyn.cuny.edu)

Hans van Ditmarsch
(hans@cs.otago.ac.nz)

Wiebe van der Hoek
(WiebevanderHoek@csc.liv.ac.uk)

David Makinson
(makinson@dcs.kcl.ac.uk)

Thomas Meyer
(Thomas.Meyer@nicta.com.au)

Abhaya Nayak
(abhaya@ics.mq.edu.au)

Odile Papini
(papini@univ-tln.fr)

Pavlos Peppas
(pavlos@upatras.gr)

Henri Prade
(prade@irit.fr)

Hans Rott
(hans.rott@psk.uni-regensburg.de)

Steven Shapiro
(shapiro@informatik.uni-leipzig.de)

Renata Wassermann
(renata@ime.usp.br)

Schedule Thursday 1 June 2006 (Rydal-Elterwater Room) Session Chairs: A Herzig and M Pagnucco

- 10.30 J Lang, About time, revision, and update
- 11.00 E Ferme and M Krevneris, Ensconcement-based contraction's axiomatic
- 11.30 A Herzig, L Perrussel and I Varzinczak, Elaborating domain description
- 12.00 Lunch
- 14.00 R Booth, S Kaci, and L van der Torre, Merging rules
- 14.30 J Seinturier, P Drap, and O Papini, A reversible framework for propositional bases merging
- 15.00 L Perrussel, J-M Thevenin and T Meyer, A mutual enrichment for agents through nested belief change: A semantic approach
- 15.30 Coffee
- 16.00 C Elsenbroich, D Gabbay, and O Rodrigues, Getting possibilities from the impossible
- 16.30 J Sefranek, Rethinking semantics of dynamic logic programming

6.1 About time, revision and update

About time, revision and update

Jérôme Lang
IRIT – CNRS / Universit Paul Sabatier
31062 toulouse Cedex, France
lang@irit.fr

Abstract

In papers on belief change we often read sentences such as "belief revision has to do with static worlds, while belief update has to do with dynamic worlds". As already argued in a few papers, this is not as simple as that. In this position paper I would like to elaborate on the following question: what is the exact scope of belief revision and belief update? In particular, I will focus on belief update and argue that it has to be understood as a particular case of action progression.

Introduction

In papers on belief change we often read sentences such as "belief revision has to do with static worlds, while belief update has to do with dynamic worlds, the input formula being then a notification of the change". A few papers, including (Friedman & Halpern 1996), have already argued that this is not as simple as that. However the discussion is far from being closed. In this short note I would like to elaborate on the following question: what is the exact scope of belief revision and belief update, and more generally belief change operators?

The point is that in order to assess ths scope of the belief change operators, we need to be able to talk about the properties of the system (the world and the available acrtions) and those pertaining to the agent's state knowledge. Now, establishing a taxonomy of logical frameworks for dealing with action and change is the very topic of Sandewall's book (Sandewall 1995). Given this, it is somewhat surprising that belief change processes have never (as far as I know) been analyzed from the point of view of such a taxonomy.

A few papers, especially (Friedman & Halpern 1996), try to address the point and argue that before discussing about postulates and representation theorems we should first make it clear what kind of knowledge we are representing and processing. Still, they do not try to connect their discussion to the clear taxonomy elaborated by Sandewall, and moreover there is still much to do, partly because since then (both (Sandewall 1995) and (Friedman & Halpern 1996) are at least 10 years old), and some new classes of belief change operators have been introduced. Moreover, Sandewall's taxonomy has to be reactualized a little bit to be applied to the variety of belief change processes.

Here I write a first attempt to assess the specialities (both of the world – ontological – and of the agent's beliefs – epistemological) under which belief revision is a suitable process, and the same for belief update. I choose to write this paper in a rather non-technical way, priviledging examples and informal discussion over formal definitions and results (in particular, I will not enter deeply into the details of Sandewall's taxononomy, and I will refer to ontological and epistemological specialities in an unformal way.) Rather, this paper is intended to raise some discussions during the NMR workshop.

Background and notations

We assume throughout the paper that a single agent (often denoted, for short, by "we" or "I") is reasoning about a dynamical system on which (s)he may partially act, and that she can partly observe.

There is a number of possible assumptions that we can make about the nature of the system and its modelling (called "ontological specialities") and the nature of the agent's belief about the system (called "epistemological specialities")[1]. Rather than listing them all and relating them to Sandewall's original taxonomy (which would take long and could be the topic of another paper), we are only listing those that are of relevance as far as belief revision and belief update arer concerned.

In the rest of the paper, if PS is a finite set of propositional symbols, then L^{PS} is the language generated from PS, the usual connectives and the Boolean constants $\top$, $\bot$.

Time Time can be either continuous, or infinite discrete, or finite, or even *one-shot* (a single time point).

[1] Here however, I mix them both, because the distinction between them is not always clear, especially in the presence of sensing actions: for instance, should the fact that an observation stemming from a sensing action is always correct be considered as an epistemological assumption (property of the agent's knowledge: what he perceives is correct) or as a property of the sensing action concerned?

In the rest of the paper we assume it to be finite. The time scale is $T = \{1, \ldots, N\}$. The process is *one-shot* of $N = 1$.

Fluents For the sake of simplicity we identify fluents with state variables (a variable that does not change over time will just be encoded by a fluent together with a strong persistency law.) Fluents can be continuous or discrete. There might be dependencies between fluents (including ramifications) or not. In the rest of the paper we take a finite set V of binary fluents (a finite set of propositional variables).

For every fluent $x \in V$ and every time point $t \in \{1, \ldots, N\}$, we define the new propositional symbol x_t, meaning "x at time t" (thus, x_t is true if and only if x holds at time t.) Let $V_t = \{x_t \mid x \in V\}$. For any formula φ of L^V and any $t \leq N$, φ_t is the propositional formula of L^{V_t} obtained by replacing each variable x in φ by x_t.

States and trajectories Once N and V are fixed, we can define states and trajectories. $S = 2^V$ (respectively $S_t = 2^{V_t}$) is the set of *states* (i.e., propositional interpretations) for L^V (respectively for L^{V_t}). Elements of S (respectively S_t) are denoted by s (resp. s_t.)

Let $S_{1 \to N} = S_1 \times \ldots \times S_N = 2^{V_1 \cup \ldots \cup V_N}$. Elements of $S_{1 \to N}$ are called *trajectories* and are denoted by τ, τ' etc. For the sake of notation we write $\tau = \langle s_1, \ldots, s_N \rangle$ instead of $s_1 \cup \ldots \cup s_t$.

For any $\varphi \in L_V$, $Mod(\varphi)$ is the set of states satisfying φ. For any $X \in S$, $for(X)$ is the formula of L^V (unique up to logical equivalence) such that $Mod(for(s)) = X$. If $X = \{s\}$ the we write $for(s)$ instead of $for(\{s\})$.

Actions and events There is a finite set ACT of actions available to the agent. ACT may be empty (action-free environment).

Generally speaking, an action A has two types of effects: an *ontic (or physical) effect* and an *epistemic effect*. Taking an exemple from (Herzig *et al.* 2000), if the action consists in tossing a coin, the ontic effect is that the next value of the fluent *heads* (which is true if the current visible side of the coin is heads) may change, whereas its epistemic effect is that the new value of the fluent is observed. This distinction between ontic and epistemic effects is classical in most setting, from cognitive robotics and logics of action and belief to Partially Observable Markov Decision Processes and Kalman filtering. For the sake of simplicity we consider only *(purely) ontic* and *(purely) epistemic* actions, without loss of generality, since an arbitrary action can be decomposed into a purely ontic action followed by a purely epistemic action.

For the sake of simplicity, in this paper we identify an ontic action A with to a *transition graph* R_A on S. $R_A(s, s')$ means that s' is accessible from s after A. $R_A(s) = \{s' \mid R_A(s, s')\}$ is the set of states that can obtain after performing A in s. If $R_A(s)$ is a singleton for all s then A is *deterministic*. If $R_A(s) = \emptyset$ then A is *inexecutable* at s. A is *fully executable* iff $R_A(s) \neq \emptyset$ for every s.[2]

An epistemic action e corresponds to a set of possible *observations*, plus a *feedbackfounction* f_e from S to 2^O, where O is a finite *observation space*. $o \in f_e(s)$ means that observation o may be obtained as feedback when performing e in state s. For the sake of simplicity, we identify O with L_V, that is, we consider that observations are propositional formulas (note however that this implies a loss of generality.) The simplest possible epistemic actions are *truth tests*, and correspond to two possible observations, φ and $\neg\varphi$, for some propositional formula φ. An epistemic action e is *truthful* iff for all $s \in S$, $o \in O$, $o \in f_e(s)$ implies $s \models o$, *deterministic* iff for all $s \in S$, $f_e(s)$ is a singleton, and *fully executable* iff for all $s \in S$, $f_e(s) \neq \emptyset$.

In addition to actions there may be possible events (or exogeneous actions.) The dynamics of an event is identical to the dynamics of an ontic action (i.e., it is expressed as a transition function.) However, action are agent-trigerred (therefore the agent is always aware of action occurrences) whereas events are nature-triggered, so that the agent is not directly aware of an event's occurrence, but may only *abduce* event occurrences by observing some of its effects on the state of the world, as in (Boutilier 1998). The inertia assumption implies that events (other than the empty event) should be minimized (in this context they are caled *surprises* in (Sandewall 1995)), although this does not need to be the case.

Belief revision is no more about static worlds than logic is about birds

When it comes to position belief revision relatively to belief update, one often reads the following sentence (or something alike): "belief revision consists in incorporating some new information about a static world, while belief update consists in incorporating to a belief base about an old state of the world a piece of information notifying some change in the world"[3]

First, there is a little bit of ambiguity about the word "static" itself; stating that the world is static either

[2]Note that expressing an ontic action as a transition graph implies some loss of generality, for several reasons. First, it rules out the possibility to have graded (e.g. probabilistic) effects: second, it rules out actions with delayed effects.

[3]Reformulating what a referee pointed out to me, a more charitable reading of this static/dynamic dichotomy is that in belief revision, the *part of the world described by the set of fluents* V is static – one could talk about "domain staticity". In his/her exact formulation, the referee said something a bit different: "given input α, in some sense, in the case of revision, the assumption is that the world has not changed with respect to α (...)". Retsricting staticity only to what concerns α (which would, for instance, mean that we assume that only fluents appearing in the formula α are required not to change) is not sufficient: such a principle would imply that $a \wedge b$ revised by $\alpha = \neg b$ (or by $\alpha = b$) should be equal to b (since a has nothing to do with α in both cases).

means that the world we are referring to is a photography of the world at a given time point (and all pieces of informations refer to this single time point); or that the world its perfectly inert (times passes, but the world remains unchanged.) This is not really a problem, however.

More importantly, *nothing in the AGM theory of belief revision implies that we should restrict its application to static worlds.* Belief revision (Alchourrón, Gärdenfors, & Makinson 1985) is meant to map a belief set K (a closed propositional theory) and a new piece of information α (a consistent propositional formula) whose truth is held for sure, into a new belief set $K * \alpha$ taking account of the new piece of information without rejecting too much of the previous beliefs. The initial belief set as well as the new piece of information may talk about the state of an evolving world at different time points. As remarked in Friedman and Halpern (Friedman & Halpern 1999), what counts is not that the world is static, but that *the language used to describe the world* is static. The latter can be guaranteed simply by time-stamping the formulas in the initial belief set K as well as the "input" formula α.

In the rest of this Section I simply develop informally the above argument, mainly by illustrating it on various examples. There is nothing really original in this Section, since the argument that time-stamping variables allows for dealing with dynamic worlds in a pure belief revision framework already appears in (Friedman & Halpern 1999). However, I feel there is a need to insist on this; furthermore, I want to develop the argument in more detail than they do, and to point out connections to some related work which is posterior to (Friedman & Halpern 1999).

(Many) examples

Example 1

1. *I sent yesterday a letter with the correct address on it.*
2. *if a letter was sent yesterday with the correct address on it then today the letter is received.*

Then I learn that the address on the letter was wrong (for instance, because the recipient has moved without notifying his new address to the post service, and I used the old address.) What do I believe now?

Example 1 is clearly a case for belief revision, more precisely, this is a *revision by some new information about the past.* A way of writing it formally consists in letting $K = Cn(CorrectAddress_1, Sent_1 \leftrightarrow Received_2)$, $\psi = \neg CorrectAddress_1$. With any "reasonable" [4] revision operator, the revision of K by ψ contains the intended conclusion $\neg Received_2$.

[4] I don't want to go into the details of specific belief revision operators. In all examples I give, any change-minimizing revision operator – that is, one of the usual revision operators commonly seen in the literature, except limit cases such as full meet revision – will give the intended result.

Example 2

1. *I sent yesterday a letter with the correct address on it.*
2. *if a letter was sent yesterday with the correct address on it then today the letter is received.*

Then I learn that the recipient has not received the letter yet. What do I believe now?

Example 2 is again a case for belief revision – this time, a *revision by some new information about the present.* A way of writing it formally consists in letting K as in Example 1 and $\psi = \neg Received_1$. What the revision of K by ψ contains depends on the relative entrenchment of the different pieces of belief. Assuming that the law governing the delivery of letters is more firmly entrenchment than both facts, the revision of K by ψ contains $\neg Sent_1 \vee \neg CorrectAddress_1$ [5].

Example 3 *Today is wednesday (time 3). Time 1 is monday and time 2 is tuesday.*

1. *Yesterday I heard on the radio that the minister for nonmonotonic reasoning of some country resigned. Unfortunately, at that time I was busy with the vacuum cleaner and I could not hear well so I have now a doubt: was that in Austria or in Australia?*
2. *Until monday, the nonmon minister in Austria was Mr. Mozart whereas the nonmon minister of Australia was Dr. Darwin.*
3. *Usually, nonmon ministers tend to keep their position for quite a long time.*

Today, I learn that Mr. Tweety has just been nominated as the new nonmon minister of Australia (which means that Dr. Darwin no longer is.) What do I believe now?

Example 3 is a sequence of two belief revision steps. Using the propositional symbols M (Mozart is nonmon minister of Austria) and D (Darwin is nonmon minister of Australia), and assuming that the revision operator used minimizes change:

- on monday I believe M_1, D_1, as well as the persistency laws $X_1 \rightarrow X_2, X_2 \rightarrow X_3$ for $X \in \{M, D\}$, therefore I also believe M_2, D_2, M_3: on monday, I expect that Mozart and Darwin will remain nonmon ministers until wednesday.
- the first revision by $\neg M_2 \vee \neg D_2$ leads me to believe M_1, D_1, $M_2 \oplus D_2$, $M_2 \rightarrow M_3, D_2 \rightarrow D_3$, as well as $M_3 \oplus D_3$. This revision step lead me to give up the belief that both fluents M and D persisted from time monday to tuesday: minimization of change makes me believe that exactly one of them persisted. Moreover, on tuesday I still believe that Mozart and Darwin were nonmon ministers in their countries on monday, and I also believe that on wednesday, either Mozart or Darwin (and exactly one of them) will still be nonmon minister.

[5] Actually, if the revision operator used minimizes change, the revision of K by ψ contains either $\neg Sent_1 \oplus \neg Correct-address_1$.

- the later revision by $\neg D_3$ makes me now believe M_1, D_1, $M_2, \neg D_2$, $M_3, \neg D_3$: on wednesday, I understand that the resigning minister was Darwin, and therefore that Mozart was still minister on tuesday, and is still on wednesday.

Note that this scenario is also a case for belief extrapolation (Dupin de Saint-Cyr & Lang 2002), which is fully consistent with the above interpretation, since belief extrapolation is a particular form of revision on time-stamped beliefs.

Example 4 *Until now I believe that when something is flying, its mass does not change. Today I've just seen a penguin flying at a speed close to the speed of light and I observe that its mass changed. What do I believe now?*

This is a case where we have to revise a generic rule governing the evolution of the world by an observation that contradicts it. The problem of revising action laws by observations has been recently considered in (Eiter *et al.* 2005), who however made no mention of this connection to belief revision. My previous beliefs were $Flying_1(x), (Mass(x) = y)_1 \rightarrow (Mass(x) = y)_2$[6], which I revised by $Flying(tweety)_1 \wedge Closetolightspeed(tweety)_1 \wedge (Mass(Tweety) = m)_1 \wedge (Mass(Tweety) = m + \varepsilon)_2$. The expected outcome is a new action law that takes account of the observed limit case[7].

Discussion

The examples above all illustrated different situations where belief revision works and where the world is everything but static. Rather, in these examples we have to incorporate a new piece of information into a belief base, both of them tell about an evolving world. This argues towards arguing the scope of belief revision includes dynamic worlds, possibly incorrect observations[8], and unexpected events. Actions can even be dealt with, provided that *what we revise with (the "input formula") is an observation.* The important point is that, following the AGM postulates, revision should coincide with expansion as soon as the input formula is consistent with the initial belief base; this property, which could be called *closure by optimism*[9], assumes that *beliefs are normally correct.*

[6]Even if I make use of variables, this is just for a commodity of notation, as I do not want to step out of propositional logic. Just assume we have a fixed, finite domain and that these rules are propositionalized by instantiation on every element of the domain.

[7]Unlike previous examples, which viewed revision as incorporation as new evidence, we have here a case for revision of a generic rule by an example (see (Dubois 2006) for a discussion on this point.)

[8]Unreliable observations can be dealt with even in a standard AGM framework, satisfying the acceptance postulate, provided that unreliable observations are in the initial belief base and not in the "input formula".

[9]I'm borrowing this expression to Didier Dubois.

Belief update is a specific case of action progression

The very meaning of belief update is much less clear than that of belief revision. The literature on belief update abunds in ambiguous explanations such as "while revision deals with static worlds, update deals with dynamic, evolving worlds", "update consists in making the belief base after being notified of a change occurring in the real world"[10]. These formulations contain many ambiguities. First, the numerous examples of the previous section, all dealing with an evolving world, suggest that the opposition between revision and update relies on the possibility or not that the state of the world may evolve is not correct. Second, not all dynamic scenarios are properly handled with belief update. Therefore, claiming that belief update is the right belief change operation for dealing with evolving worlds is unsufficient. Take Example 3, for instance. With most update operators, updating $M \wedge D$ first by $\neg M \vee \neg D$ and then by $\neg D$ will lead to believe only $\neg D$ at time 3. This is due to postulate (U8) below, which is the key property of belief update, stating that possible models are updated independently (I'll come back later on this point.) This raises the following question: what is a "notification of change"? Example 3 deals with two notifications of change, still it cannot be formulated as a belief update problem.

Because the language L^V is generated by a finite number of propositional variables, we can consider, as in (Katsuno & Mendelzon 1991), belief update as transforming a propositional formulas into another propositional formula (rather than transforming a closed theory into a closed theory): an update operator is an operator from $L^V \times L^V$ to L^V. We recall here the Katsuno-Mendelzon postulates for belief update. They have been discussed and criticized in many places, including (Herzig & Rifi 1999), who argued that not all these postulates should be required. In the following we call "update operator" any operator from $L^V \times L^V$ to L^V satisfying at least (U1) and (U8). If φ and α are two propositional formulae of L^V, and $\diamond$ an update operator, then $\varphi \diamond \alpha$ is the update of φ by α.

U1 $\varphi \diamond \alpha \models \alpha$.

U2 If $\varphi \models \alpha$ then $\varphi \diamond \alpha \equiv K$.

U3 If φ and α are both satisfiable then $\varphi \diamond \alpha$ is satisfiable.

U4 If $\varphi \equiv \psi$ and $\alpha \equiv \beta$ then $\varphi \diamond \alpha \equiv \psi \diamond \beta$.

U5 $(\varphi \diamond \alpha) \wedge \beta \models \varphi \diamond (\alpha \wedge \beta)$.

U6 If $\varphi \diamond \alpha \models \beta$ and $\varphi \diamond \beta \models \alpha$ then $\varphi \diamond \alpha \equiv \varphi \diamond \beta$.

U7 If φ is complete then $(\varphi \diamond \alpha) \wedge (\varphi \diamond \beta) \models \varphi \diamond (\alpha \vee \beta)$.

U8 $(\varphi \vee \psi) \diamond \alpha \equiv (\varphi \diamond \alpha) \vee (\psi \diamond \alpha)$.

[10]These formulations appear in the original paper (Katsuno & Mendelzon 1991), which may be one of the explanations for such a long persistence of ambiguity.

The key point is postulate U8 which, by requiring that all models of the initial belief set be updated separately, forbids us to infer new beliefs about the past from later observations. This inadequacy of update to handle observations is not new (see e.g. (Boutilier 1998; Dupin de Saint-Cyr & Lang 2002)), but is not often mentioned in papers about belief update: indeed, saying that updating φ by α corresponds to incorporating an *observation* α reflecting a change in the world is incorrect (see the end of this section.)

One could try to argue that such scenarios (such as Example 3) are both a case for revision and update, depending whether the formulation of the problem uses time-stamped variables or not. This line of argumentation fails: expressing Example 3 as a belief update problem still leads to the counterintuitive results that we do not learn anything about M. Besides, several authors remarked that, unless belief bases are restricted to complete bases, a belief update operator cannot be a belief revision operator. For instance, it is shown in (Herzig 1998; Peppas *et al.* 1996) that the AGM postulates are inconsistent with U8 as soon as the language contains at least two propositional symbols.

Actually, update has to be understood as a particular form of *action progression* (for purely ontic actions). Action progression (as considered in the literature of reasoning about action and logic-based planning) consists in determining the belief base obtained from an old belief state after a given action was performed This strong connection between belief update an action progression was first mentioned in (del Val & Shoham 1994), who argue that updating an initial belief base K by a formula α corresponds to one particular action; they formalize such actions in a formal theory of actions based on circumscription, and their framework for reasoning action is then used to derive a semantics for belief update. The relationship between update and action progression appears (more or less explicitly) in several other papers, including (Liberatore 2000b), who expresses several belief update operators in a specific action language. (See also the concluding Section.) Still, the relationship between update and action progression still needs to be investigated in more detail – this is what I am trying to do next.

As said in the background section, a purely ontic action A corresponds to a transition graph R_A on S. For any formula $\varphi \in L^V$, the *progression of* φ *by* A is the propositional formula (unique up to logical equivalence) whose models are the states that can obtain after performing A in a state of $Mod(\varphi)$: $prog(\varphi, A)$ is defined by

$$prog(\varphi, A) = form\left(\bigcup_{s \models \varphi} R_A(s)\right) \quad (1)$$

The *weak* (or *deductive*) *regression* (or *preimage*[11]) of ψ by A is the formula whose models are the states from which the execution of A possibly leads to a model of ψ, i.e.

$$reg(\psi, A) = form\left(\{s, R_A(s) \cap Mod(\psi) \neq \emptyset\}\right)$$

The *strong* (or *abductive*) *regression* (or *strong preimage*) of ψ by A is the formula whose models are the states from which the execution of A certainly leads to a model of ψ, i.e.

$$Reg(\psi, A) = form\left(\{s, R_A(s) \subseteq Mod(\psi)\}\right)$$

See for instance (Lang, Lin, & Marquis 2003) on the meaning of these two notions. [12]

Clearly enough, (1) is identical to (U8). Therefore, for any update operator (and more generally any operator satisfying (U8)) and any input formula α, updating by α is an action progression operator. The next questions are: (a) Which action is this exactly? (b) What is the class of actions that correspond to a belief update? (c) If update is progression, are there belief change operators corresponding to weak and strong regression?

Question (a) first. Updating/progressing states separately as in (U8) and (1) means that the action is feedback-free. Indeed, a feedback would allow us to eliminate some states after the action has been performed, which in turn would lead us to eliminate some states before the action took place (see (Boutilier 1998; Dupin de Saint-Cyr & Lang 2002))[13]. This comes down to say that belief update assumes *unobservability*: the set of possible states after A is performed is totally determined by the set of possible states before it is performed and the transition system ciorresponding to A. In other words, *what you foresee is what you get*: once we have decided to perform A, waiting until is has actually been performed will not bring us any new information.

[11] The terminology preimage / strong preimage comes from the planning literature.

[12] Remark that the probabilistic variant of action progression is the well-known action progression operator for stochastic actions: let p is a probability distribution over S and A a stochastic action described by a stochastic matrix $p(.|., A)$, where $p(s'|s, A)$ is the probability of obtaining s' after performing A in s. Then $prog_P(p, A)$ is the probability distribution over S defined by

$$prog_P(p, A)(s') = \sum_{s \in S} p(s)p(s'|s, A)$$

Mapping each probability distribution p into the belief state $B(p) = for(\{s|p(s) > 0\})$ consisting of those states deemed possible by p, i.e.,

$$B(prog_P(p, A)) = prog(B(p), A)$$

As argued in (Dubois & Prade 1993), the probabilistic variant of belief update is Lewis' imaging (Lewis 1973).

[13] Unless the state of the world after the action is performed is totally disconnected from the state of the world before the action is performed, which only happens if $R_A(s) = S$ for all s. In this case, a feedback never allows for learning anything about the past state of the world. Clearly, this case is a very degenerated one.

Now, we know that $update(\alpha)$ is a feedback-free action. Can we describe $update(\alpha)$ in more details? Postulate (U1) means that $update(\alpha)$ has to be understood as "make α true". More precisely, due to the absence of feedback reflected by (U8), updating φ by α precisely means this, which could be seen as a dialogue between an agent and a robot: "All I know about the state of the world is that is satisfies φ. Please, go to the real world, see its real state, and whatever this state, act so as change it into a world sastisfying α, following some rules" (given that the robot will not communicate with the agent once it will be the real world.) These rules to be followed by the robot are dictated by the choice of the update operator $\diamond$. If $\diamond$ satisfies (U2), then the rules state that if the α is already true then the robot should not do anything and let the world as it is. If $\diamond$ is change-minimizing such as the PMA (Winslett 1990), then the rules are that the robot should "make α true, without changing more variables than necessary". More generally, when $\diamond$ is a Katsuno-Mendelzon operator, associated with a collection of similarity preorders (one for each world), the robot should "make α true by changing s into one of the states that are most similar to it" [14]. When $\diamond$ is a forgetting-based operation, such as in (Herzig 1996; Doherty, Łukaszewicz, & Madalińska-Bugaj 1998), then the rules state "make α true, without changing the truth values of the variables which are irrelevant to α." And so on.

Writing things more formally: given an update operator $\diamond$ and a formula α, let $update(\diamond, \alpha)$ be the ontic action defined by: for all $s, s' \in S$,

$$s' \in R_{update(\diamond,\alpha)}(s) \text{ iff } s' \models for(s) \diamond \alpha$$

Then

Proposition 1 *If $\diamond$ satisfies (U1) and (U8) then $update(\diamond, \alpha)$ is a ontic action, and $\varphi \diamond \alpha \equiv prog(\varphi, \alpha)$.*

In Proposition 1, requiring (U3) in addition to (U1) and (U8) would correspond to having $update(\diamond, \alpha)$ fully executable. We may wonder what new properties of $update(\diamond, \alpha)$ obtain when other postulates are required. (U2) is particularly interesting in this respect. For any action A, let $Inv(A)$ be the set of *invariant states* for A, that is, the set of all states s such that $R_A(s) = \{s\}$.

Proposition 2 *Let $\diamond$ satisfying (U1) and (U8). Then $\diamond$ satisfies (U2) if and only if for all $s \in S$, $R_{update(\diamond,\alpha)}(s) \subseteq Inv(update(\diamond, \alpha))$.*

Thus, the inertia postulate (U2) together with (U1) means that any state that can be reached by $update(\diamond, \alpha)$ is an invariant state. (A quick proof of the left-to-right direction: by (U1), $update(\diamond, \alpha)$ maps any state to a set of states satisfying α; then by (U2), any of these states is invariant by $update(\diamond, \alpha)$.) [15].

The other postulates will not have any direct effect on the properties of $update(\diamond, \alpha)$ considered as an isolated action, because they relate different actions of the form $update(\diamond, \alpha)$. Noticeably, requiring (U4) corresponds to the equality between $update(\diamond, \alpha)$ and $update(\diamond, \beta)$ when α and β are logically equivalent.

Let us not consider question (b). Obviously, given a *fixed* update operator $\diamond$ satisfying (U1), (U4) and (U8), there are actions that are not of the form $update(\diamond, \alpha)$ (this is obvious because there are 2^{2^n} actions of the form $update(\diamond, \alpha)$ and 2^{n+2^n} possible actions, where $n = |V|$.)[16].

Now, what happens if we allow $\diamond$ to vary? The question now is, what are the actions that can be expressed as $update(\diamond, \alpha)$, for some update operator $\diamond$ and some α? Here is a first answer:

Proposition 3 *Let A be an ontic action such that A is fully executable and $R_A(s) \subseteq Inv(A)$ for all $s \in S$. Then there exists an update operator $\diamond$ satisfying all the Katsuno-Mendelzon postulates, and a formula α such that $A = update(\diamond, \alpha)$.*

The proof is constructive: α is taken such that $Mod(\alpha) = Inv(A)$, and the collection of faithful orderings is defined by $s_1 <_s s_2$ if and only if $s = s_1 \neq s_2$ or $(s \neq s_1, s \neq s_2, s_1 \in Inv(A), s_2 \notin Inv(A))$; and $s_1 \leq_s s_2$ iff not $(s_2 <_s s_1)$.

Putting Propositions 2 and 3 together we get

Corollary 1 *Let A be an ontic action. There exists an update operator $\diamond$ satisfying all the Katsuno-Mendelzon postulates, and a formula α such that $A = update(\diamond, \alpha)$, if and only if A is fully executable and $R_A(s) \subseteq Inv(A)$ for all $s \in S$.*

Now, question (c). Is there a natural definition of reverse update which would be to action regression what update is to progression? The point is that we do not have one, but two notions of action regression, which naturally leads to two notions of reverse update.

Definition 1 *Let $\diamond$ be an update operator.*

- *the* weak reverse update *associated with $\diamond$ is defined by: for all $\psi, \alpha \in L^V$,*

$$\psi \odot \alpha = for(\{s \mid for(s) \diamond \alpha \not\models \neg\psi\})$$

[14] Note that, as argued in (Peppas *et al.* 1996), this similarity has here be understood as an *ontological* notion (s being closer to s_1 than to s_2 may, in practice, reflect that from s it is easier to go to s_1 than to s_2) and not as an epistemic notion of similarity, as it would be the case for belief revision (again see (Peppas *et al.* 1996)).

[15] Note that if $R_{update(\diamond,\alpha)}(s) \subseteq Inv(update(\diamond, \alpha))$ for all s, then $update(\diamond, \alpha)$ is involutive, i.e., $R_{update(\diamond,\alpha)} \circ R_{update(\diamond,\alpha)} = R_{update(\diamond,\alpha)}$, but the converse fails to hold.

[16] Here is another proof, more intuitive: let $V = \{p\}$, thus $S = \{p, \neg p\}$, and consider the actions $A = switch(p)$, such that $R_A(p) = \{\neg p\}$ and $R_A(\neg p) = \{p\}$. Assume there is a formula α such that $A = update(\diamond, \alpha)$; then $U1$ enforces $\alpha \equiv \top$; therefore, if $A = update(\diamond, \alpha)$ then by (U4), $A = update(A, \top)$. Now, let A' be the identity action; we also have that if A' can be expressed as an update action for $\diamond$, then $A' = update(\diamond, \top)$. Therefore, at most one of A and A' can be expressed as an update action for $\diamond$.

- *the* strong reverse update *associated with* $\diamond$ *is defined by: for all* $\psi, \alpha \in L^V$,

$$\psi \otimes \alpha = for(\{s \mid for(s) \diamond \alpha \models \psi\})$$

Intuitively, weak reverse update corresponds to (deductive) postdiction: given that the action "make α true" has been performed and that we now know that ψ holds, all we can say about the state of the world before the action was performed is that it satisfied $\psi \odot \alpha$.

As to strong reverse update, it is an abductive form of postdiction, better interpreted as follows: given that a rational agent has a goal ψ and has performed "make α true", the states of the world before the action was performed in which the action was guaranteed to succeed are those satisfying $\psi \otimes \alpha$.

Example 5 *Let* $\diamond$ *be the PMA update operator (Winslett 1990).*

- $\neg a \odot (\neg a \vee \neg b) \equiv (\neg a \vee b)$ *and* $\neg a \otimes (\neg a \vee \neg b) \equiv \neg a$.
- $\neg(a \vee b) \odot (\neg a \vee \neg b) \equiv (a \vee b) \otimes (\neg a \vee \neg b) \equiv (a \vee b)$.

Here are now some interesting properties of weak and strong reverse update. These properties will vary, of course, with the properties required for belief update. We start with the basic postulates (U1), (U4) and (U8).

Proposition 4 *Let* $\diamond$ *be an update operator satisfying (U1), (U4) and (U8).*

SW0 *if* $\psi \equiv \psi'$ *and* $\alpha \equiv \alpha'$ *then* $\psi \odot \alpha \equiv \psi' \odot \alpha'$ *and* $\psi \otimes \alpha \equiv \psi' \otimes \alpha'$.

W1 $\psi \odot \alpha \equiv (\psi \wedge \alpha) \odot \alpha$;

W2 $\psi \models (\psi \odot \alpha) \diamond \alpha$;

S1 $(\psi \wedge \alpha) \otimes \alpha \models \psi \otimes \alpha$;

S2 $(\psi \otimes \alpha) \diamond \alpha \models \psi$.

SW3 $\psi \otimes \alpha \models \psi \odot \alpha$.

Note that W1 entails: if $\psi \models \neg\alpha$ then $\psi \odot \alpha \equiv \bot$.

Example 5 shows that not only the converse implication of (SW3) fails to hold, but also that so does the converse implication of (S1): let $\psi = (a \vee b)$, $\alpha = (\neg a \vee \neg b)$, then $\psi \otimes \alpha = (a \vee b)$, and $(\psi \wedge \alpha) \otimes \alpha \equiv (a \leftrightarrow \neg b) \otimes (\neg a \vee \neg b) \equiv (a \leftrightarrow \neg b)$, which is not equivalent to $\psi \otimes \alpha$. Likewise, the converse of (W2) and (S2) fail to hold.

Other properties of reverse update are obtained when (U2) is required for $\diamond$.

Proposition 5 *Let* $\diamond$ *be an update operator satisfying (U1), (U2) and (U8).*

S4 $\psi \wedge \alpha \models \psi \otimes \alpha$;

W5 $(\psi \odot \alpha) \odot \alpha \equiv \psi \odot \alpha$.

S5 $(\psi \otimes \alpha) \otimes \alpha \equiv \psi \otimes \alpha$.

S4, together with SW3, entails (W4) $\psi \wedge \alpha \models \psi \odot \alpha$. (S4) and (W4) entail: if $\psi \models \alpha$ then $\psi \models \psi \otimes \alpha$ and $\psi \models \psi \odot \alpha$. Which entails in turn: $\alpha \models \alpha \otimes \alpha$ and $\alpha \models \alpha \odot \alpha$.

Pursuing the investigation on reverse update does not only have a theoretical interest: weak (deductive) reverse update allows for postdiction, while strong (abuctive) reverse update allows for goal regression, when the actions performed are updates. Anyway, in this paper I won't try to go further, as my goal was just to show how weak and strong reverse update can be defined. A last word: for those who would have wondered about that: reverse update has nothing to do with erasure (Katsuno & Mendelzon 1991).

In the section on revision we have seen that it is possible (and relevant) to revise some beliefs about the past, present and future by some new information about the past / the present / the link between both. Can we do the same for belief update? Is it meaningful to update by some information about the past, the future, or the way the world evolves?

Recall that we argued that updating by α corresponds to performing the action "make α true". Therefore, updating by some information the past, even if this can be written formally, is no less than weird[17]. Updating by some information about the present is no less weird. Updating by information about the next time point is exactly what usual belief update is about. Updating by some information about some remote future can be meaningful as well. For instance, updating the current belief $a_t \vee b_t$ by $\neg b_{t+2}$ means that the robot is asked to perform an action (following the rule dictated by the update operator) whose result will be that b will be false at time $t+2$. In an inert context where we have no reason to believe that b will be assigned to false earlier than at $t+2$, we expect that everything the result of the update implies at time $t+1$ is $a_{t+1} \vee b_{t+1}$.

A last point. Now that we know that $update(\alpha)$ is an action progression operator, we also know that it can be described by a propositional action theory such as those described in (Giunchiglia *et al.* 2004) etc., at least in the following obvious way: for each state s consider the causal rule

$$\mathsf{if}\ \text{for(s)}\ \mathsf{then}\ \alpha\ \mathsf{causes}\ s \diamond \alpha$$

The union of all these rules, once compiled in a propositional action theory Σ_α, will be the propositional encoding of the update operator. The question is now whether $update(\alpha)$ can be described as *simple* (that is, short) action theories. The answer depends, of course, on the update operator. Results on computational complexity, however, allow for giving a quick negative answer for many update operators. Indeed, we know (Lang, Lin, & Marquis 2003) that, given an action A specified as a set of causal rules or as a propositional action theory Σ_A, and two formulas φ, ψ, checking whether $prog(\varphi, \alpha) \models \psi$ is coNP-complete. Let now $\diamond$ an update operator such as checking whether $\varphi \diamond \alpha \models \psi$ is C-complete for some class above coNP in the complexity hierarchy.

[17] It might be argued that this happened in some political contexts that I don't have to recall here – but this, of course, was making other agents believe at time $t' > t$ that some formula, that was true at time t, was actually false – this is definitely another story...

Then, if there were a way of expressing $update(\diamond, \alpha)$ (for all α) by a polynomially large action theory, then we would have $C = \mathsf{coNP}$, which is highly unlikely. Now, most update operators are above coNP – most of them being Π_2^p-complete (Eiter & Gottlob 1992; Liberatore 2000a). This implication is actually an equivalence. For instance, let us state formally the result for the PMA:

Proposition 6 *Let $\diamond = \diamond_{PMA}$. Then 1. and 2. are equivalent:*

1. *there exists a polysize function $\Sigma : L^V \rightarrow L^{V_t \cup V_{t+1}}$ such that for all $s, s' \in S$, $(s_t, s'_{t+1}) \models \Sigma(\alpha)$ if and only if $(s, s') \in R_{update(\diamond,\alpha)}$.*
2. *the polynomial hierarchy collapses at the first level.*

Similar results would hold with most update operators of the literature, with the noticeable exceptions of the following two ones: the MPMA (Doherty, Łukasziewicz, & Madalińska-Bugaj 1998), WSS (Winslett 1990), $\text{WSS}^{\downarrow}$, $\text{WSS}^{\downarrow}_{dep}$ (Herzig 1996; Herzig & Rifi 1999), as well as the update operator proposed in (Herzig *et al.* 2001). All those update operators consists first in forgetting (in the sense of (Lin & Reiter 1994)) a set of variables (for instance, those that appear in the input formula α, or those in which α depends etc.), and then expanding by α. For these, checking whether $\varphi \diamond \alpha \models \psi$ is ' only" coNP-complete. And here is the translation, given here for WSS:

$$\Sigma(\alpha) = \alpha_{t+1} \wedge \bigwedge_{v \notin Var(\alpha)} v_t \leftrightarrow v_{t+1}$$

It is now time to summarize and discuss the scope of belief update. We have extensively argued that updating a belief base K by a formula α has to be interpreted as progressing (or projecting) K by the action *make α true*. The "input formula" α is an expected action effect, and *not* an observation: standard belief update precludes any possibility of feedback. Observations lead to filter out some possible states; an observation at time $t+1$ leads to filter out not only some possible states for time $t+1$, but also some possible states for time t, because the state of the world at time t and the state of the world at time $t+1$ are correlated, by default persistence rules or other dynamic rules[18]. The only case where belief update would be compatible with interpreting α as an observation would be the one where not the faintest correlation would exist between the state of the world at different time points – a totally degenerate and uninteresting case, as argued above. Thus, *belief update is suitable only for (some specific) ontic action progression with no feedback.*

Concluding remarks

Let us first summarize what we have said do far. Both revision and update deal with dynamic worlds, but they strongly differ in the nature of the information they are processing. Revision aims at correcting some initial beliefs about the past, the present and even the future state of the world by some newly *observed* information again about the past or the present state of the world. Update has nothing do to with observations: it is meant to project some belief state forward given the knowledge that some particular ontic action has been performed, *without any feedback being observed*. We have seen that if updating by α corresponds to progressing by a specific action (depending both on the update operator and on α), conversely, not all ontic actions can be expressed as updates, even if the update operator is allowed to vary, as soon as updates are required to satisfy the staticity postulate (U2) in addition to the basic postulates (U1), (U4) and (U8). Then, unsurprisingly, the relationship of update to action progression draws a similar relationship between action regression and "reverse update".

In complex environments, especially planning under incomplete knowledge, actions are complex and have both ontic and epistemic effects; the belief change process then is very much like the feedback loop in partially observable planning and control theory: perform an action, project its effects on the current belief state, then get the feedback, and revise the projected belief state by the feedback. Or, equivalently, if one chooses to separate the ontic and the epistemic effects of actions, by having two disjoint sets of actions (ontic and epistemic), then ontic actions lead to projection only, while epistemic actions lead to revision only (see also (Cossart & Tessier 1999)). Two recent lines of work consider both kinds of actions in a belief change framework: (Shapiro *et al.* 2000; Shapiro & Pagnucco 2004), who express revision and update in the situation calculus, and (Hunter & Delgrande 2005). Both works make it clear that update corresponds to ontic actions while revision correspond to epistemic actions and observations. However, this raises the following question: given that not all ontic actions can be expressed as updates, how restrictive is it to limit the available ontic actions to the latter ones? Why not going more general? Of course, one could give up belief update and move to causal theories of actions. But the point is that if we shall then lose the nice features of belief update, including its ability to deal with disjunctive effects. Another possibility would consist in enriching belief update by allowing for nondeterministic, conditional and concurrent updates, such as in (Herzig *et al.* 2001). This is left for further research.

Acknowledgements

I have been discussing and arguing with Andreas Herzig for ten years about the very meaning of belief update. Without these discussions I would never have had the idea of writing this position paper. The issues addressed in this paper owe a lot to discussions and/or joint works with Florence Dupin de Saint-Cyr – Bannay, Jim Delgrande, Didier Dubois, Andreas Herzig (again), Sébastien Konieczny, Gabriele Kern-Isberner, Fangzhen

[18] This very principle is the basis for generalized update (Boutilier 1998), which integrates revision and update.

Lin, and Pierre Marquis. Thanks as well to the participants of the Dagstuhl seminar "Belief change in rational agents: perspectives from Artificial Intelligence, Philosophy and Economics", August 2005.

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change : partial meet contraction and revision functions. *Journal of Symbolic Logic* 50:510–530.

Boutilier, C. 1998. A unified model of qualitative belief change: a dynamical systems perspective. *Artificial Intelligence* 1-2:281–316.

Cossart, C., and Tessier, C. 1999. Filtering vs revision and update: let us debate! In *ECSQARU'99, 5th European conference on symbolic and quantitative approaches to reasoning with uncertainty*.

del Val, A., and Shoham, Y. 1994. Deriving properties of belief update from theories of action. *J. of Logic, Language, and Information* 3:81–119.

Doherty, P.; Łukasziewicz, W.; and Madalińska-Bugaj, E. 1998. The PMA and relativizing change for action update. In *Proc. KR'98*, 258–269.

Dubois, D., and Prade, H. 1993. Revision and update in numerical formalisms. In *Proceedings of IJCAI93*.

Dubois, D. 2006. Three views on belief revision. submitted.

Dupin de Saint-Cyr, F., and Lang, J. 2002. Belief extrapolation (or how to reason about observations and unpredicted change). In *Proceedings of KR2002*, 97–508.

Eiter, T., and Gottlob, G. 1992. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence* 57:227–270.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating action domain descriptions. In *IJCAI 2005*, 418–423.

Friedman, N., and Halpern, J. 1996. Belief revision: A critique. In *Proceedings of KR'96*, 421–431.

Friedman, N., and Halpern, J. 1999. Modelling beliefs in dynamic systems. part ii: revision and update. *JAIR* 10:117–167.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Giunchiglia, F.; Lee, J.; Lifschitz, V.; Cain, N. M.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence*.

Herzig, A., and Rifi, O. 1999. Propositional belief update and minimal change. *Artificial Intelligence* 115:107–138.

Herzig, A.; Lang, J.; Longin, D.; and Polacsek, T. 2000. A logic for planning under partial observability. In *Proc. of AAAI'2000*, 768–773.

Herzig, A.; Lang, J.; Marquis, P.; and Polacsek, T. 2001. Actions, updates, and planning. In *Proceedings of IJCAI'2001*, 119–124.

Herzig, A. 1996. The PMA revisited. In *Proceedings of KR'96*, 40–50.

Herzig, A. 1998. Logics for belief base updating. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*. Kluwer Academic Press.

Hunter, A., and Delgrande, J. 2005. Iterated belief change: a transition system approach. In *Proceedings of IJCAI05*.

Katsuno, H., and Mendelzon, A. 1991. Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52:263–294.

Lang, J.; Lin, F.; and Marquis, P. 2003. Causal theories of action: a computational core. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 1073–1078.

Lewis, D. 1973. *Counterfactuals*. Harvard University Press.

Liberatore, P. 2000a. The complexity of belief update. *Artificial Intelligence* 119:141–190.

Liberatore, P. 2000b. A framework for belief update. In *Proceedings of JELIA-2000*.

Lin, F., and Reiter, R. 1994. Forget it! In *Proceedings of the AAAI Fall Symposium on Relevance*, 154–159.

Peppas, P.; Nayak, A.; Pagnucco, M.; Foo, N.; Kwok, R.; and Prokopenko, M. 1996. Revision vs. update: taking a closer look. In *Proceedings of ECAI96*.

Sandewall, E. 1995. *Features and Fluents*. Oxford University Press.

Shapiro, S., and Pagnucco, M. 2004. Iterated belief change and exogeneous actions in the situation calculus. In *Proceedings of ECAI04*.

Shapiro, S.; Pagnucco, M.; Lespé-rance, Y.; and Levesque, H. J. 2000. Iterated belief change in the situation calculus. In *Proc. of KR'00*, 527–538.

Winslett, M. 1990. *Updating Logical Databases*. Cambridge University Press.

6.2 An axiomatic characterization of ensconcement-based contraction

An axiomatic characterization of ensconcement-based contraction (Preliminary Version)

Eduardo Fermé
Departamento de Matemática e Engenharias
Universidade da Madeira
ferme@uma.pt

Martín Krevneris
Soluciones Informaticas Coop.
mkrev@2vias.com.ar *

Abstract

In this paper we propose an axiomatic characterization for *ensconcement-based contraction function*, belief base function proposed by Williams. We relate this function with other kinds of base contraction functions.

Introduction

The logic of theory change became a major subject in philosophical logic and artificial intelligence in the middle of the 1980's. The most important model, now known as the AGM model of belief change, was proposed by Alchourrón, Gärdenfors and Makinson in (Alchourrón, Gärdenfors, & Makinson 1985). The AGM model is a formal framework to characterize the dynamics and state of belief of a rational agent. The beliefs of an agent are represented by a set of sentences closed under logical consequence. The AGM model has acquired the status of a standard model, and has been characterized in at least five different equivalent ways: *Postulates, partial meet functions, epistemic entrenchment, safe/kernel contraction* and *Grove' sphere-systems*.
One of the most important variants in the AGM model is to represent the beliefs of an agent by a *belief base*, a set of sentences that is not (necessarily) closed under logical consequence.
Partial meet contraction and kernel contraction had been characterized for belief bases. Mary-Anne Williams proposed a model of contraction based in an "ensconcement", closely related with epistemic entrenchment. In this paper we provide an axiomatic characterization of *ensconcement-based contraction functions*.

Contraction functions for Belief Bases

The use of logically closed sets to represent beliefs has received much criticism. Among them we can mention: 1. Belief sets make no distinction between basic beliefs and those which were inferred from them. 2. They are computability intractable. 3. Belief Bases can distinguish between different inconsistent belief states.

*We would like to thank the audience at the Dagstuhl Seminar of Belief Change in Rational Agents for many helpful comments and stimulating suggestions. Thanks also due to the anonymous referee for his numerous perceptive remarks and criticisms; and thanks to José Castanheira for corrections concerning presentation.

Several authors propose the use of belief bases for logic of theory change. We will use the belief base theory proposed by Hansson(Hansson 1991), where a belief base is any set of sentences.

Formal preliminaries: We will assume a language $\mathcal{L}$ that is closed under truth-functional operations and a consequence operator Cn for $\mathcal{L}$. Cn satisfies the standard Tarskian properties, namely inclusion ($A \subseteq Cn(A)$), monotony (if $A \subseteq B$, then $Cn(A) \subseteq Cn(B)$), and iteration ($Cn(A) = Cn(Cn(A))$). It is supraclassical and compact, and satisfies deduction (if $\beta \in Cn(A \cup \{\alpha\})$, then $(\alpha \rightarrow \beta) \in Cn(A)$). $A \vdash \alpha$ will be used as an alternative notation for $\alpha \in Cn(A)$, $\vdash \alpha$ for $\alpha \in Cn(\emptyset)$ and $Cn(\alpha)$ for $Cn(\{\alpha\})$. Upper-case letters denote subsets of $\mathcal{L}$. Lower-case Greek letters denote elements of $\mathcal{L}$. Base expansion is simply a set union, i.e., $A + \alpha = A \cup \{\alpha\}$. A contraction of A with respect to α involves removal of a set of sentences from A so that α is no longer implied.

Partial Meet Contraction

We can construct a base contraction function using the *remainder* sets, i.e., maximal subsets of A that fail to imply α:

Definition 1 (Alchourrón & Makinson 1981) Let A be a belief base and α a sentence. The set $A \perp \alpha$ (A *remainder* α) is the set of sets such that $B \in A \perp \alpha$ if and only if:

- $B \subseteq A$
- $B \nvdash \alpha$
- There is no set B' such that $B \subset B' \subseteq A$ and $B' \nvdash \alpha$

Definition 2 (Alchourrón, Gärdenfors, & Makinson 1985; Hansson 1991) The partial meet base contraction operator on A based on a selection function γ is the operator $-_\gamma$ such that for all sentences α:

$$A -_\gamma \alpha = \cap\gamma(A \perp \alpha)$$

Hansson characterized partial meet base contraction in terms of postulates:

Theorem 3 (Hansson 1991) Let A be a belief base. An operator $-$ on A is a partial meet contraction function for A if and only if $-$ satisfies

Success If $\nvdash \alpha$, then $A - \alpha \nvdash \alpha$.

Inclusion $A{-}\alpha \subseteq A$.
Relevance If $\beta \in A$ and $\beta \notin A{-}\alpha$ then there is some set A' such that $A{-}\alpha \subseteq A' \subseteq A$ and $\alpha \notin Cn(A')$ but $\alpha \in Cn(A' \cup \{\beta\})$.
Uniformity If it holds for all subsets A' of A that $\alpha \in Cn(A')$ if and only if $\beta \in Cn(A')$, then $A - \alpha = A - \beta$.

Partial meet base contraction also satisfies other interesting properties as we will see in the following observation:

Observation 4 (Hansson 1999) Let A be a belief base and $-$ an operator on A. Then:
1. If $-$ satisfies *inclusion* and *relevance*, then it satisfies *vacuity* (If $A \not\vdash \alpha$, then $A - \alpha = A$).
2. If $-$ satisfies *uniformity*, then it satisfies *extensionality* (If $\vdash \alpha \leftrightarrow \beta$, then $A - \alpha = A - \beta$).
3. If $-$ satisfies *inclusion* and *relevance*, then it satisfies *failure* (If $\vdash \alpha$, then $A - \alpha = A$).

Kernel Contraction

In (Hansson 1994) Hansson introduced *Kernel Contraction*, a generalization of Safe Contraction(Alchourrón & Makinson 1985). It is based on a selection among the sentences of a set A that contribute effectively to imply α; and on how to use this selection in contracting by α. Formally:

Definition 5 (Hansson 1994) Let A be a set in $\mathcal{L}$ and α a sentence. Then $A\perp\!\!\!\perp\alpha$ is the set such that $B \in A\perp\!\!\!\perp\alpha$ if and only if:

$$\begin{cases} B \subseteq A \\ B \vdash \alpha \\ \text{If } B' \subset B \text{ then } B' \not\vdash \alpha \end{cases}$$

$A\perp\!\!\!\perp\alpha$ is called the *kernel set of A with respect to α* and its elements are the *α-kernels* of A.

Definition 6 (Hansson 1994) Let A be a set of sentences. Let $A\perp\!\!\!\perp\alpha$ be the kernel set of A with respect to α. An *incision function* σ for A is a function such that for all sentences α:

$$\begin{cases} \sigma(A\perp\!\!\!\perp\alpha) \subseteq \bigcup(A\perp\!\!\!\perp\alpha) \\ \emptyset \neq B \in A\perp\!\!\!\perp\alpha, \text{ then } B \cap \sigma(A\perp\!\!\!\perp\alpha) \neq \emptyset \end{cases}$$

Definition 7 (Hansson 1994) Let A be a set of sentences and σ an incision function for A. The *kernel contraction* $-_\sigma$ for A is defined as: :

$$A{-}_\sigma\alpha = A \setminus \sigma(A\perp\!\!\!\perp\alpha).$$

An operator $-$ for a set A is a kernel contraction if and only if there is an incision function σ for A such that $A{-}\alpha = A{-}_\sigma\alpha$ for all sentences α.

Hansson also provided an axiomatic characterization for kernel contraction.

Theorem 8 (Hansson 1994) The operator $-$ for a set of sentences A is a kernel contraction if and only if it satisfies *success, inclusion, uniformity* and

Core retainment If $\beta \in A$ and $\beta \notin A{-}\alpha$ then there is some set A' such that $A' \subseteq A$ and $\alpha \notin Cn(A')$ but $\alpha \in Cn(A' \cup \{\beta\})$.

Since *core retainment* is weaker than *relevance*, it follows that, for belief bases, all partial meet contractions are kernel contractions. There exist two more conservative types of kernel contraction:

A kernel contraction is *smooth* if and only if for all subsets A' of A that: if $A' \vdash \beta$ and $\beta \in \sigma(A\perp\!\!\!\perp\alpha)$ then $A' \cap \sigma(A\perp\!\!\!\perp\alpha) \neq \emptyset$ (Hansson 1994).

A kernel contraction is *relevant* if and only if for all $\beta \in \sigma(B\perp\!\!\!\perp\alpha)$, there is an X such that $(B \setminus \sigma(B\perp\!\!\!\perp\alpha)) \subseteq X \subseteq B$, $X \not\vdash \alpha$ and $X \cup \{\beta\} \vdash \alpha$ (Falappa, Fermé, & Kern-Isberner 2006).

Theorem 8 (cont.)

- (Hansson 1994) $-$ is smooth if and only if it also satisfies:
  Relative Closure $A \cap Cn(A - \alpha) \subseteq A - \alpha$.
- (Falappa, Fermé, & Kern-Isberner 2006) $-$ is relevant if and only if it also satisfies *relevance*.

Ensconcement

Mary-Anne Williams (Williams 1992; 1994b; 1995) defines an *ensconcement* relation on a belief base A as a transitive and connective relation $\preceq$ that satisfies the following three conditions:[1]

($\preceq$1) If $\beta \in A \setminus Cn(\emptyset)$, then $\{\alpha \in A : \beta \prec \alpha\} \not\vdash \beta$.
($\preceq$2) If $\not\vdash \alpha$ and $\vdash \beta$, then $\alpha \prec \beta$
($\preceq$3) If $\vdash \alpha$ and $\vdash \beta$, then $\alpha \preceq \beta$

$\preceq$1 says that the formulae that are strictly more ensconced than α do not (even conjointly) imply α. Conditions $\preceq$2 and $\preceq$3 say that tautologies are the most ensconced formulae. Given an ensconcement relation, a cut operator is defined by:

$$cut_A(\alpha) = \{\beta \in A : \{\gamma \in A : \beta \preceq \gamma\} \not\vdash \alpha\}$$

Lemma 9 (Williams 1994a)
If $\alpha \in A$, $cut_A(\alpha) = \{\beta \in A : \alpha \prec \beta\}$

The previous Lemma says that when α is an explicit belief, its cut is the subset of A such that its members are strictly more ensconced than α. Other interesting properties of *cut* are:

Lemma 10

(a) If $\alpha \prec \beta$, then $cut_A(\alpha \wedge \beta) = cut_A(\alpha)$.
(b) If $\alpha \prec \beta$, then $cut_A(\alpha) \vdash \beta$ and $cut_A(\beta) \not\vdash \alpha$.
(c) If $\beta =_\preceq \alpha$, then $cut_A(\alpha \wedge \beta) = cut_A(\alpha) = cut_A(\beta)$.

[1] $\alpha \prec \beta$ means $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$. $\alpha =_\preceq \beta$ means $\alpha \preceq \beta$ and $\beta \preceq \alpha$.

(d) If $\nvdash \alpha, cut_A(\alpha) \nvdash \alpha$.
(e) If $A \nvdash \alpha, cut_A(\alpha) = A$
(f) If $cut_A(\alpha) \vdash \beta$, then $cut_A(\alpha \wedge \beta) = cut_A(\alpha)$.
(g) If $cut_A(\alpha) \nvdash \beta$, then $cut_A(\alpha \wedge \beta) = cut_A(\beta)$.

We can define a base contraction operator $-$ using the cut operator:

Definition 11 Let A be a belief base and $\prec$ an ensconcement relation. Then $-$ is an Ensconcement-Based Contraction Function if and only if:

$\beta \in A - \alpha$ if and only if $\beta \in A$ and either (i) $\alpha \in Cn(\emptyset)$ or (ii) $cut_A(\alpha) \vdash \alpha \vee \beta$}[2]

Axiomatic for Ensconcement-Based Contraction Function

In this section we are going to investigate the postulates that characterize the Ensconcement-Based Contraction Function. The following postulates are well known in the belief revision literature:

Success If $\nvdash \alpha$, then $A - \alpha \nvdash \alpha$.
Inclusion $A - \alpha \subseteq A$.
Vacuity If $A \nvdash \alpha$, then $A \subseteq A - \alpha$.
Extensionality If $\vdash \alpha \leftrightarrow \beta$, then $A - \alpha = A - \beta$
Conjunctive Factoring $A - \alpha \wedge \beta = \begin{cases} A - \alpha \text{ or} \\ A - \beta \text{ or} \\ A - \alpha \cap A - \beta \end{cases}$

The question that arises here is what is the postulate that characterizes the notion of "minimal change" in ensconcement. We propose the following postulate:

Disjunctive Elimination If $\beta \in A$ and $\beta \notin A - \alpha$ then $A - \alpha \nvdash \alpha \vee \beta$.

It is important to note the relation between *disjunctive elimination* and other postulates:

Observation 12 Let A be a belief base and $-$ an operator on A that satisfies *disjunctive elimination*. Then $-$ satisfies:
Relative Closure $A \cap Cn(A - \alpha) \subseteq A - \alpha$.
If $-$ also satisfies *inclusion* then it satisfies:
Failure If $\vdash \alpha$ then $A - \alpha = A$.

By means of Disjunctive Elimination we can characterize Ensconcement-Based Contraction Functions:

Theorem 13 Let A be a belief base. An operator $-$ of A is an ensconcement-based contraction on A if and only if it satisfies *success, inclusion, vacuity, extensionality, disjunctive elimination* and *conjunctive factoring*.

[2]Note that condition (ii) requires that $cut_A(\alpha) \vdash \alpha \vee \beta$ instead of the more intuitive condition $cut_A(\alpha) \vdash \beta$. Mary-Anne Williams calls a contraction based in this alternative condition "Brutal Theory Base Contraction". Brutal Theory Base Contraction is closely related to Severe Withdrawal (Rott 1991), whereas Ensconcement-Based Contraction is closely related to AGM contraction.

Due to Observation 12 every ensconcement-based contraction function satisfies *failure* and *relative closure*.

Observation 14 An ensconcement-based contraction function $-$ on A does not, in general, satisfies *uniformity*.

Ensconcement and minimal change

As we showed, ensconcement-based contraction satisfies *vacuity, failure* and *relative success*. *Vacuity* guarantees the minimal change (i.e. to do nothing) when the sentence to be contracted is not implied by the original belief base. *Failure* (Fuhrmann & Hansson 1994), means that when instructed to do the impossible (to contract a tautology), the minimal change is to do nothing. *Relative closure* was introduced in (Hansson 1994) and ensures that original beliefs that are implied by the contracted set are not *gratuitously* removed.

In the previous section, we referred that *disjunctive elimination* is the postulate that characterizes the notion of minimal change in ensconcement-based contraction. In its original formulation (where the precondition is the same as in *core retainment* and *relevance*) the consequence $A - \alpha \nvdash \alpha \vee \beta$ appears as a negative condition. However if we reformulate the postulate as

If $\beta \in A$ and $A - \alpha \vdash \alpha \vee \beta$, then $\beta \in A - \alpha$.

we can see the postulate as a condition for a sentence β "to survive" the contraction process. If we reformulate *relative closure* as "If $\beta \in A$ and $A - \alpha \vdash \beta$, then $\beta \in A - \alpha$" we can see *disjunctive elimination* as a weaker version of *relative closure*, that requires that original beliefs that are implied by the contracted set should be elements of the contracted set.

In belief bases, *disjunctive elimination* is a condition weaker than *relevance*:

Observation 15 Let A be a belief base. Then *relevance* implies *disjunctive elimination*.

In (Rott 2000), Rott points out that ensconcement-based contraction gratuitously loses independent beliefs with a low priority in the belief base, as we see in the following example:

Let $A = \{\alpha, \beta, \beta \rightarrow \delta\}$ and assume, α independent of β. Let $\prec$ be an ensconcement for A such that $\beta \prec \alpha \prec \beta \rightarrow \delta$. Let $-$ be the ensconcement-based contraction for A based on $\prec$. Hence $A - \alpha = \{\beta \rightarrow \delta\}$, and β was loosed.

This happens because, in general, *core-retainment* fails for ensconcement-based contractions. We note also that even if an ensconcement-based contraction satisfies core retainment it may not satisfy *relevance*:

Observation 16 Let A be a belief base and $-$ and operator on A. Then *core retainment* and *disjunctive elimination* do not imply *relevance*.

The difference between *relevance* and *disjunctive elimination* disappears if A is a belief set.

Observation 17 Let A be a belief set and $-$ and operator on A that satisfies *inclusion* and *vacuity*. Then *disjunctive elimination* implies *relevance*.

Maps between different base contraction functions

Mary-Anne Williams demonstrated that an ensconcement based contraction can be related with an AGM contraction function:

Observation 18 (Williams 1994a)
For every ensconcement-based contraction $-$ on a belief base base A there is an AGM contraction $\div$ on the corresponding belief set $Cn(A)$ (that satisfies the basic and supplementary postulates for belief set contraction) such that for all sentences α:

$$A - \alpha = Cn(A) \div \alpha \cap A$$

We can generalize this result just for the basic postulates:

Observation 19 Let A be a belief base. Let $-$ be an operator on A. Then $-$ satisfies *success, inclusion, vacuity, extensionality* and *disjunctive elimination* iff there exists some basic AGM contraction $\div$ for $Cn(A)$ such that

$$A - \alpha = Cn(A) \div \alpha \cap A$$

We will call $-$ a *basic related-AGM base contraction*.

Due to Observations 4 and 15 it follows that every partial meet contraction is a basic related-AGM base contraction. With this remark we can construct the actual map of the base contraction functions as follows:

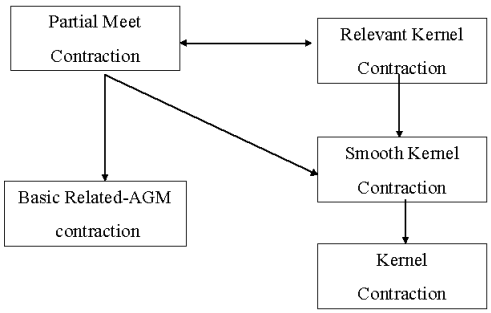


Figure 1: Map among different kinds of base contraction functions

Conclusions and Future Works

We found an axiomatic for ensconcement-based contraction and we defined a new contraction operator (basic related-AGM). With these results we extended the maps among different kinds of base contraction functions. In the near future we intend to analyze the following open questions:

- How to use the relationship $A - \alpha = Cn(A) \div \alpha \cap A$ for other contraction functions like Levi contraction, Semi-Contraction or Severe Withdrawal (for an overview of these functions see (Fermé & Rodriguez 1998a; Fermé & Rodríguez 1998b; Rott & Pagnucco 1991)).
- How to extend the results for belief bases of partial meet contraction and kernel contraction for supplementary postulates.
- How to elucidate the relation among different kinds of supplementary postulates for belief bases.

Appendix: Proofs

Proof of Theorem 13
From Ensconcement-based Contraction to Postulates:

Success Let $\not\vdash \alpha$ and assume by *reductio* that $A - \alpha \vdash \alpha$. Then it follows by compactness that there exists a finite subset of $A - \alpha$, $A' = \{\beta_1, ..., \beta_k\}$ and such that $A' \vdash \alpha$. Then it follows by definition of $-$ that $cut_A(\alpha) \vdash \alpha \vee \beta_i$, $i = 1...k$. Then $cut_A(\alpha) \vdash \alpha \vee (\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_k) \vdash \alpha$. Hence $cut_A(\alpha) \vdash \alpha$. Contradiction.
Inclusion Trivial.
Vacuity Let $A \not\vdash \alpha$ and let $\beta \in A$. By lemma 10(e) it follows that $cut_A(\alpha) = A$, from which it follows that $cut_A(\alpha) \vdash \alpha \vee \beta$, hence by definition of $-$, $\beta \in A - \alpha$.
Extensionality Let $\vdash \alpha \leftrightarrow \beta$. Then $cut_A(\alpha) = cut_A(\beta)$, and the rest follows trivially.
Disjunctive Elimination Let $\beta \in A$ and $\beta \notin A - \alpha$. Then it follows from definition of $-$ that $cut_A(\alpha) \not\vdash \alpha \vee \beta$. Assume by *reductio* that $A - \alpha \vdash \alpha \vee \beta$. Then compactness yields that there exists a finite subset of $A - \alpha$, $A' = \{\beta_1, ..., \beta_k\}$ and such that $A' \vdash \alpha \vee \beta$. It follows by definition of $-$ that $cut_A(\alpha) \vdash \alpha \vee \beta_i$, $i = 1...k$. Then $cut_A(\alpha) \vdash \alpha \vee (\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_k) \vdash \alpha \vee \beta$. Hence $cut_A(\alpha) \vdash \alpha \vee \beta$. Contradiction.
Conjunctive Factoring If $\vdash \alpha \wedge \beta$ trivial by definition of $-$. Let $\not\vdash \alpha \wedge \beta$. We will prove by cases:
1. $cut_A(\alpha) \vdash \beta$: Then by lemma 10 (f) $cut_A(\alpha \wedge \beta) = cut_A(\alpha)$. We will prove by double inclusion that $A - \alpha \wedge \beta = A - \alpha$. Let $\gamma \in A - \alpha \wedge \beta$. It follows by definition of $-$ that $\gamma \in A$ and $cut_A(\alpha \wedge \beta) \vdash (\alpha \wedge \beta) \vee \gamma$, then $cut_A(\alpha \wedge \beta) \vdash \alpha \vee \gamma$. Hence $cut_A(\alpha) \vdash \alpha \vee \gamma$, from which we can conclude that $\gamma \in A - \alpha$.
For the other hand, let $\gamma \in A - \alpha$. Then it follows by definition of $-$ that $\gamma \in A$ and $cut_A(\alpha) \vdash \alpha \vee \gamma$. Then $cut_A(\alpha \wedge \beta) \vdash \alpha \vee \gamma$. $cut_A(\alpha) \vdash \beta$ yields $cut_A(\alpha) \vdash \beta \vee \gamma$, then $cut_A(\alpha \wedge \beta) \vdash \beta \vee \gamma$. Hence $cut_A(\alpha \wedge \beta) \vdash (\alpha \wedge \beta) \vee \gamma$. From definition we can conclude that $\gamma \in A - \alpha \wedge \beta$.
2. $cut_A(\beta) \vdash \alpha$: Due to the symmetry of the case, it follows that $A - \alpha \wedge \beta = A - \beta$.
3. $cut_A(\alpha) \not\vdash \beta$ and $cut_A(\beta) \not\vdash \alpha$: It follows by lemma 10 (g) that $cut_A(\alpha \wedge \beta) = cut_A(\alpha) = cut_A(\beta)$. Let $\gamma \in A - \alpha \wedge \beta$. By definition $\gamma \in A$ and $cut_A(\alpha \wedge \beta) \vdash (\alpha \wedge \beta) \vee \gamma$ iff $cut_A(\alpha \wedge \beta) \vdash \alpha \vee \gamma$ and $cut_A(\alpha \wedge \beta) \vdash \beta \vee \gamma$ iff $cut_A(\alpha) \vdash \alpha \vee \gamma$ and $cut_A(\beta) \vdash \beta \vee \gamma$ iff $\gamma \in A - \alpha$ and $\gamma \in A - \beta$.

From Postulates to Ensconcement-based Contraction:

Let $-$ be an operator to A that satisfies *success, inclusion, vacuity, extensionality, disjunctive elimination* and *conjunctive factoring*. In order to prove that $-$ is an ensconcement-based contraction we must prove that there

exists a relationship $\preceq$ such that $\prec$ satisfies $\preceq$**1 -** $\preceq$**3** and such that

$$\textbf{ebc}\ A - \alpha = \begin{cases} \{\beta \in A : cut_A(\alpha) \vdash \alpha \vee \beta\} & \text{if } \nvdash \alpha \\ A & \text{otherwise} \end{cases}$$

where cut is defined in terms of $\preceq$.

Let $\preceq$ defined as follows:

$$\alpha \preceq \beta \text{ iff } \begin{cases} A - \alpha \wedge \beta \nvdash \alpha \\ or \\ \vdash \alpha \wedge \beta \end{cases}$$

$\preceq$**1** Let $\gamma \in A \setminus Cn(\emptyset)$ and assume by *reductio* that $H = \{\alpha \in A : \gamma \prec \alpha\} \vdash \gamma$. Using our construction $\gamma \prec \alpha$ means that $A - \alpha \wedge \gamma \vdash \alpha$, $\nvdash \alpha \wedge \gamma$ and $A - \alpha \wedge \gamma \nvdash \gamma$. $\nvdash \alpha \wedge \gamma$ is redundant, then $H = \{\alpha \in A : A - \alpha \wedge \gamma \vdash \alpha$ and $A - \alpha \wedge \gamma \nvdash \gamma\}$. It follows by *conjunctive factoring* that $A - \alpha_i \wedge \gamma = A - \gamma$ for each $\alpha_i \in H$. Then $H = \{\alpha \in A : A - \gamma \vdash \alpha$ and $A - \gamma \nvdash \gamma\}$. Hence by *success* and *relative closure* $H = A - \gamma$. *Success* contradicts $H \vdash \gamma$, since $A - \gamma \nvdash \gamma$.

$\preceq$**2** Let $\nvdash \alpha$ and $\vdash \beta$. Due to *success* it follows that $A - \alpha \wedge \beta \nvdash \alpha \wedge \beta$, then $A - \alpha \wedge \beta \nvdash \alpha$. Then $\alpha \preceq \beta$. Since $\vdash \beta$, then $A - \alpha \wedge \beta \vdash \beta$ and due to $\nvdash \alpha \wedge \beta$ it follows that $\beta \not\preceq \alpha$. Hence $\alpha \prec \beta$.

$\preceq$**3** Let $\alpha,\ \beta \in A$. Let $\vdash \alpha$ and $\vdash \beta$. Then $\vdash \alpha \wedge \beta$. Hence $\alpha \preceq \beta$.

ebc We will prove by cases:

1. $\vdash \alpha$. Follows trivial by *failure*.

2. $\nvdash \alpha$

2.1 $A \nvdash \alpha$. *Vacuity* yields $A - \alpha = A$. On the other hand $cut_A(\alpha) = A$, from which it follows that $\{\beta \in A : cut_A(\alpha) \vdash \alpha \vee \beta\} = A$.

2.2 $A \vdash \alpha$.

2.2.1 $\alpha \in A$. We will first replace $\prec$ in cut_A by our construction:

Due to lemma 9

$$cut_A(\alpha) = \{\delta \in A : \alpha \prec \delta\}$$
$$cut_A(\alpha) = \{\delta \in A : \alpha \preceq \delta \text{ and } \delta \not\preceq \alpha\}$$
$$cut_A(\alpha) = \{\delta \in A : A - \alpha \wedge \delta \nvdash \alpha \text{ and } A - \alpha \wedge \delta \vdash \delta\}$$

it follows by *conjunctive factoring* that $A - \alpha \wedge \delta = A - \alpha$. Then

$$cut_A(\alpha) = \{\delta \in A : A - \alpha \vdash \delta\},$$

hence by *relative closure* and *inclusion*,

$$cut_A(\alpha) = A - \alpha$$

Replacing in **ebc**:

$$\beta \in A - \alpha \text{ iff } \beta \in A \text{ and } A - \alpha \vdash \alpha \vee \beta,$$

that trivially follows from *inclusion* and *disjunctive elimination*.

2.2.2 $\alpha \notin A$. First we will construct $A' = A \cup \{\alpha\}$ and let $\preceq'$ such that for all β, γ in A, $\beta \preceq' \gamma$ iff $\beta \preceq \gamma$, $\alpha \prec' \beta$ if $cut_A(\alpha \wedge \beta) \neq cut_A(\beta)$, $\beta \prec' \alpha$ if $cut_A(\alpha \wedge \beta) \neq cut_A(\alpha)$ and $\alpha =_{\preceq} \beta$ if $cut_A(\alpha \wedge \beta) = cut_A(\alpha) = cut_A(\beta)$. Let $cut'_{A'}(\alpha)$ defined in terms of A' and $\preceq'$. Due to previous proof 2.2.1, we will prove this case proving that $\{\beta \in A : cut_A(\alpha) \vdash \alpha \vee \beta\} = \{\beta \in A' : cut'_{A'}(\alpha) \vdash \alpha \vee \beta\}$. To do that we must prove (a) $\preceq'$ is an ensconcement and (b) $Cn(cut_A(\beta)) = Cn(cut'_{A'}(\beta))$. Part (a) is trivial. For part (b) we have that $cut'_{A'}(\beta) = \{\delta \in A' : \{\gamma \in A' : \delta \preceq' \gamma\} \nvdash \beta\} = \{\delta \in A \cup \{\alpha\} : \{\gamma \in A \cup \{\alpha\} : \delta \preceq \gamma\} \nvdash \beta\}$. If $\alpha \notin cut'_{A'}(\beta)$ it follows that $cut'_{A'}(\beta) = cut_A(\beta)$. If $\alpha \in cut'_{A'}(\beta)$ it follows that $\{\gamma \in A' : \alpha \preceq \gamma\} \nvdash \beta\}$, from which it follows that $\alpha \not\preceq \beta$. Then $\beta \prec \alpha$, hence (by lemma 10 (b)) $cut_A(\beta) \vdash \alpha$, from which it follows that $Cn(cut_A(\beta)) = Cn(cut'_{A'}(\beta))$.

Proof of observation 14 Counterexample: Let $A = \{\alpha \wedge \beta, \alpha \vee \gamma, \gamma\}$, such that $\gamma \prec \alpha \wedge \beta \prec \alpha \vee \gamma$. Then $cut_A(\alpha) = cut_A(\beta) = \{\alpha \vee \gamma\}$. $\gamma \in A - \alpha$, since $cut_A(\alpha) \vdash \alpha \vee \gamma$, but $\gamma \notin A - \beta$, since $cut_A(\beta) \nvdash \alpha \vee \gamma$. Hence *uniformity* fails.

Proof of observation 15 Let $\beta \in A$ and $\beta \notin A - \alpha$. Then by *relevance* there is some set A' such that $A - \alpha \subseteq A' \subseteq A$ and $\alpha \notin Cn(A')$ but $\alpha \in Cn(A' \cup \{\beta\})$, from which it follows by deduction that $\beta \rightarrow \alpha \in Cn(A')$. Hence (due to $\alpha \notin Cn(A')$), $\alpha \vee \beta \notin Cn(A')$.

Proof of observation 16 Counterexample: (Assume that *core retainment* and *disjunctive elimination* are satisfied for all $\delta \neq \alpha$). Let $A = \{\beta, \gamma \rightarrow \alpha, \gamma \wedge (\beta \rightarrow \alpha)\}$. Let $A - \alpha = \{\gamma \rightarrow \alpha\}$. *Disjunctive elimination* is satisfied since $A - \alpha \nvdash \alpha \vee \beta$ and $A - \alpha \nvdash \alpha \vee (\gamma \wedge (\beta \rightarrow \alpha))$. To show that *core retainment* is satisfied for β let $H = \{\gamma \wedge (\beta \rightarrow \alpha)\}$ and for $\gamma \wedge (\beta \rightarrow \alpha)$ let $H = \{\gamma \rightarrow \alpha\}$. However, *relevance* fails for β.

Proof of observation 17 Let $\beta \in A$ and $\beta \notin A - \alpha$. Then it follows by *disjunctive elimination* that $A - \alpha \nvdash \alpha \vee \beta$. By deduction theorem $(A - \alpha) + \neg\beta \nvdash \alpha$, from which it follows that $(A - \alpha) + \neg\beta \vee \alpha \nvdash \alpha$. Let $B = (A - \alpha) + \neg\beta \vee \alpha$. Due to $\beta \in A$ and $\beta \notin A - \alpha$ *vacuity* yields that $A \vdash \alpha$. From inclusion $A - \alpha \subseteq A$. Hence (by monotony) $B \subseteq A + \neg\beta \vee \alpha = A$. $B \nvdash \alpha$, and $B + \beta \vdash \alpha$. Hence $-$ satisfies *relevance*.

Proof of observation 19

(For this proof we assume that the reader is familiar with the AGM contraction functions).

$\Leftarrow$. Let $\div$ an operator on $Cn(A)$ that satisfies the basic AGM postulates (*closure, success, vacuity, inclusion, recovery* and, *extensionality*), and let $-$ such that $A - \alpha = Cn(A) \div \alpha \cap A$.

Success, inclusion, vacuity and *extensionality* trivially follow trivially from definition. For *disjunctive elimination* let $\beta \in A$ and $\beta \notin A - \alpha$. Then (by *closure*) $Cn(A \div \alpha) \nvdash \beta$, from which it follows by *recovery* that $Cn(A \div \alpha) \vdash \neg\alpha \vee \beta$. Then it follows by *success* and *vacuity* that $Cn(A \div \alpha) \nvdash \alpha \vee \beta$. Hence $A - \alpha \nvdash \alpha \vee \beta$.

$\Rightarrow$ Let $-$ be an operator on A that satisfies *success, inclusion, vacuity, extensionality* and *disjunctive elimination* and let $\div$ defined as follows:

$$Cn(A) \div \alpha = Cn(A - \alpha \cup (\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i))$$

We must prove: (i) that $\div$ is a basic AGM contraction function, (ii) that $\div$ satisfies $A - \alpha = Cn(A) \div \alpha \cap A$.

(i) *Closure* follows trivially from definition. *Success* and *extensionality* follows trivially from definition and $-$*success* and $-$*extensionality* respectively. *Vacuity* follows from $-$*vacuity* and from $(\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i) \subseteq Cn(A)$. For *re-*

covery let $\gamma \in Cn(A)$ and $\gamma \notin Cn(A) \div \alpha$, we must prove that $\alpha \rightarrow \gamma \in Cn(A) \div \alpha$ that follows trivially from our definition of $\div$.
(ii) We must prove that $A - \alpha = Cn(A - \alpha \cup (\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i)) \cap A$. We will prove by double inclusion. Let γ be a sentence. If $\vdash \gamma \rightarrow \alpha$, trivial by success. Let $\nvdash \gamma \rightarrow \alpha$. For one side let $\gamma \in A - \alpha$. Then by *inclusion* $\gamma \in A$ and the rest follows trivially. For the other side let $\gamma \in Cn(A - \alpha \cup (\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i)) \cap A$. Then $\gamma \in A$ and $\gamma \in Cn(A - \alpha \cup (\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i))$ from which it follows (due to $\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i \vdash \neg\alpha \vee \gamma$ and $\bigcup_{\beta_i \in A} \alpha \rightarrow \beta_i \nvdash \alpha \vee \gamma$) that $A - \alpha \vdash \alpha \vee \gamma$. Hence by *disjunctive elimination* $\gamma \in A - \alpha$.

References

Alchourrón, C., and Makinson, D. 1981. Hierarchies of regulations and their logic. In Hilpinen, R., ed., *New Studies in Deontic Logic: Norms, Actions, and the Foundations of Ethics*, 125–148.

Alchourrón, C., and Makinson, D. 1985. On the logic of theory change: Safe contraction. *Studia Logica* 44:405–422.

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic* 50:510–530.

Falappa, M.; Fermé, E.; and Kern-Isberner, G. 2006. On the logic of theory change: Relations between incision and selection functions. In *Proceedings 17th European Conference on Artificial Intelligence, ECAI06*. (to appear).

Fermé, E., and Rodriguez, R. 1998a. A brief note about the Rott contraction. *Logic Journal of the IGPL* 6(6):835–842.

Fermé, E., and Rodríguez, R. 1998b. Semi-contraction: Axioms and construction. *Notre Dome Journal of Formal Logic* 39(3):332–345.

Fuhrmann, A., and Hansson, S. O. 1994. A survey of multiple contraction. *Journal of Logic, Language and Information* 3:39–74.

Hansson, S. O. 1991. *Belief Base Dynamics*. Ph.D. Dissertation, Uppsala University.

Hansson, S. O. 1994. Kernel contraction. *Journal of Symbolic Logic* 59:845–859.

Hansson, S. O. 1999. A survey of non-prioritized belief revision. *Erkenntnis* 50:413–427.

Rott, H., and Pagnucco, M. 1991. Severe withdrawal (and recovery). *Journal of Philosophical Logic* 28:501–547.

Rott, H. 1991. Two methods of constructing contractions and revisions of knowledge systems. *Journal of Philosophical Logic* 20:149–173.

Rott, H. 2000. "Just because". Taking belief bases seriously. In Buss, S.; Hajek, P.; and Pudlak, P., eds., *Logic Colloquium '98 - Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic. Lecture Notes in Logic*, volume 13. Prague: Association for Symbolic Logic.

Williams, M.-A. 1992. Two operators for theory bases. In *Proc. Australian Joint Artificial Intelligence Conference*, 259–265. World Scientific.

Williams, M.-A. 1994a. On the logic of theory base change. In MacNish., ed., *Logics in Artificial Intelligence*, number 835 in Lecture Notes Series in Computer Science. Springer Verlag.

Williams, M.-A. 1994b. Transmutations of knowledge systems. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Proceedings of the fourth International Conference on Principles of Knowledge Representation and Reasoning*. Bonn, Germany: Morgan Kaufmann. 619–629.

Williams, M.-A. 1995. Iterated theory base change: A computational model. In *Proc. of the 14th IJCAI*, 1541–1547.

6.3 Elaboraing domain descriptions

Elaborating domain descriptions

Andreas Herzig **Laurent Perrussel** **Ivan Varzinczak***
IRIT – 118 route de Narbonne
31062 Toulouse Cedex – France
e-mail: {herzig,perrusse,ivan}@irit.fr

Abstract

In this work we address the problem of *elaborating* domain descriptions (alias action theories), in particular those that are expressed in dynamic logic. We define a general method based on contraction of formulas in a version of propositional dynamic logic with an incorporated solution to the frame problem. We present the semantics of our theory change and define syntactical operators for contracting a domain description. We establish soundness and completeness of the operators w.r.t. the semantics for descriptions that satisfy a principle of modularity that we have proposed elsewhere. We also investigate an example of changing non-modular domain descriptions.

Introduction

Suppose a situation where an agent has always believed that if the light switch is up, then there is light in the room. Suppose now that someday, she observes that even if the switch is in the upper position, the light is off. In such a case, the agent must change her beliefs about the relation between the propositions "the switch is up" and "the light is on". This example is an instance of the problem of changing propositional belief bases and is largely addressed in the literature about belief change (Gärdenfors 1988) and belief update (Katsuno & Mendelzon 1992).

Next, let our agent believe that whenever the switch is down, after toggling it, there is light in the room. This means that if the light is off, in every state of the world that follows the execution of toggling the switch, the room is lit up. Then, during a blackout, the agent toggles the switch and surprisingly the room is still dark.

Imagine now that the agent never worried about the relation between toggling the switch and the material it is made of, in the sense that she ever believed that just toggling the switch does not break it. Nevertheless, in a stressful day, she toggles the switch and then observes that she had broken it.

Completing the wayside cross our agent experiments in discovering the world's behavior, suppose she has believed that it is always possible to toggle the switch, provided some conditions like being close enough to it, having a free hand, the switch is not broken, etc, are satisfied. However, in a beautiful April fool's day, the agent discovers that someone has glued the switch and, consequently, it is no longer possible to toggle it.

The last three examples illustrate situations where changing the beliefs about the behavior of the action of toggling the switch is mandatory. In the first one, toggling the switch, once believed to be deterministic, has now to be seen as nondeterministic, or alternatively to have a different outcome in a specific context (e.g. if the power station is overloaded). In the second example, toggling the switch is known to have side-effects (ramifications) one was not aware of. In the last example, the executability of the action under concern is questioned in the light of new information showing a context that was not known to preclude its execution. Carrying out modifications is what we here call *elaborating* a domain description, which has to do with the principle of *elaboration tolerance* (McCarthy 1988).

Such cases of theory change are very important when one deals with logical descriptions of dynamic domains: it may always happen that one discovers that an action actually has a behavior that is different from that one has always believed it had.

Up to now, theory change has been studied mainly for knowledge bases in classical logics, both in terms of revision and update. Only in a few recent works it has been considered in the realm of modal logics, viz. in epistemic logic (Hansson 1999), and in action languages (Eiter *et al.* 2005). Recently, several works (Shapiro *et al.* 2000; Jin & Thielscher 2005) have investigated revision of beliefs about facts of the world. In our examples, this would concern e.g. the current status of the switch: the agent believes it is up, but is wrong about this and might subsequently be forced to revise his beliefs about the current state of affairs. Such belief revision operations do not modify the agent's beliefs about the action laws. In opposition to that, here we are interested exactly in such modifications. The aim of this work is to make a step toward that issue and propose a framework that deals with the contraction of action theories.

Dynamic logic, more specifically propositional dynamic logic (PDL (Harel 1984)), has been extensively used in reasoning about actions in the last years (Castilho, Gasquet, & Herzig 1999; Castilho, Herzig, & Varzinczak 2002; Zhang & Foo 2001; Foo & Zhang 2002; Zhang, Chopra, &

*Supported by a fellowship from the government of the FEDERATIVE REPUBLIC OF BRAZIL. Grant: CAPES BEX 1389/01-7.

Foo 2002). It has shown to be a viable alternative to situation calculus approaches because of its simplicity and existence of proof procedures for it. In this work we investigate the elaboration of domain descriptions encoded in a simplified version of such a logical formalism, viz. the multimodal logic K_n. We show how a theory expressed in terms of static laws, effect laws and executability laws is elaborated: usually, a law has to be changed due to its generality, i.e., the law is too strong and has to be weakened. It follows that elaborating an action theory means contracting it by static, effect or executability laws, before expanding the theory with more specific laws.

The present text is organized as follows: in the next section we define the logical framework we use throughout this work and show how action theories are encoded. Then we present our semantics of theory change and its syntactical counterpart. After that we establish soundness and completeness of our change operators w.r.t. the semantics, where completeness is conditioned by a notion of modularity that we have proposed in previous work. We then analyse an example of correcting a non-modular theory. Before concluding, we address related work on the field and discuss on how elaboration tolerant the framework here proposed is.

Background

Following the tradition in the reasoning about actions community, action theories are going to be collections of statements that have the particular form: "if *context*, then *effect* after *every execution* of *action*" (effect laws); and "if *precondition*, then *action executable*" (executability laws). Statements mentioning no action at all represent laws about the world (static laws). Besides that, statements of the form "if *context*, then *effect* after *some execution* of *action*" will be used as a causal notion to solve the frame and the ramification problems.

Logical preliminaries

Let $\mathfrak{Act} = \{a_1, a_2, \ldots\}$ be the set of all *atomic action constants* of a given domain. An example of atomic action is *toggle*. To each atomic action a there is associated a modal operator $[a]$.

$\mathfrak{Prop} = \{p_1, p_2, \ldots\}$ denotes the set of all *propositional constants*, also called *fluents* or *atoms*. Examples of those are *light* ("the light is on") and *up* ("the switch is up"). The set of all literals is $\mathfrak{Lit} = \{l_1, l_2, \ldots\}$, where each l_i is either p or $\neg p$, for some $p \in \mathfrak{Prop}$. If $l = \neg p$, then we identify $\neg l$ with p.

We use small Greek letters $\varphi, \psi, \ldots$ to denote *classical formulas*. They are recursively defined in the usual way:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi$$

$\mathfrak{Fml}$ is the set of all classical formulas. An example of a classical formula is $up \rightarrow light$. By $val(\varphi)$ we denote the set of valuations making φ true. We view a valuation as a maximally-consistent set of literals. For $\mathfrak{Prop} = \{light, up\}$, there are four valuations: $\{light, up\}$, $\{light, \neg up\}$, $\{\neg light, up\}$ and $\{\neg light, \neg up\}$. Given a set of formulas Σ, by $lit(\Sigma)$ we denote the set of all literals appearing in formulas of Σ.

We denote complex formulas (with modal operators) by $\Phi, \Psi, \ldots$ They are recursively defined in the following way:

$$\Phi ::= \varphi \mid [a]\Phi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \rightarrow \Phi \mid \Phi \leftrightarrow \Phi$$

$\langle a \rangle$ is the dual operator of $[a]$, defined as $\langle a \rangle \Phi =_{\text{def}} \neg[a]\neg\Phi$. An example of a complex formula is $\neg up \rightarrow [toggle]up$.

The semantics is that of multimodal logic K (Popkorn 1994).

Definition 1 A K_n*-model* is a tuple $\mathscr{M} = \langle W, R \rangle$ where W is a set of valuations, and R a function mapping action constants a to accessibility relations $R_a \subseteq W \times W$.

Definition 2 Given a K_n-model $\mathscr{M} = \langle W, R \rangle$,

- $\models^{\mathscr{M}}_{w} p$ (p is true at world w of model $\mathscr{M}$) if $p \in w$;
- $\models^{\mathscr{M}}_{w} [a]\Phi$ if for every w' such that wR_aw', $\models^{\mathscr{M}}_{w'} \Phi$;
- truth conditions for the other connectives are as usual.

Definition 3 $\mathscr{M}$ is a model of Φ (noted $\models^{\mathscr{M}} \Phi$) if and only if for all $w \in W$, $\models^{\mathscr{M}}_{w} \Phi$. $\mathscr{M}$ is a model of a set of formulas Σ (noted $\models^{\mathscr{M}} \Sigma$) if and only if $\models^{\mathscr{M}} \Phi$ for every $\Phi \in \Sigma$. A formula Φ is a *consequence of the set of global axioms* Γ in the class of all K_n-models (noted $\Gamma \models_{\mathsf{K}_n} \Phi$) if and only if for every K_n-model $\mathscr{M}$, if $\models^{\mathscr{M}} \Gamma$, then $\models^{\mathscr{M}} \Phi$.

Describing the behavior of actions in K_n

Given a domain, we are interested in theories whose statements describe the behavior of actions. K_n allows for the representation of such statements, that we call *action laws*. Here we distinguish several types of them. The first kind of statement represents the *static laws*, which are formulas that must hold in every possible state of the world.

Definition 4 A *static law* is a formula $\varphi \in \mathfrak{Fml}$.

An example of a static law is $up \rightarrow light$, saying that if the switch is up, then the light is on. The set of all static laws of a domain is denoted by $\mathcal{S} \subseteq \mathfrak{Fml}$.

The second kind of action law we consider is given by the *effect laws*. These are formulas relating an action to its effects, which can be conditional.

Definition 5 An *effect law for action a* is of the form $\varphi \rightarrow [a]\psi$, where $\varphi, \psi \in \mathfrak{Fml}$.

The consequent ψ is the effect which always obtains when action a is executed in a state where the antecedent φ holds. If a is a nondeterministic action, then the consequent ψ is typically a disjunction. The set of effect laws of a domain is denoted by $\mathcal{E}$. An example of an effect law is $\neg up \rightarrow [toggle]light$, saying that whenever the switch is down, after toggling it, the room is lit up. If ψ is inconsistent, we have a special kind of effect law that we call an *inexecutability law*. For example, $broken \rightarrow [toggle]\bot$ expresses that *toggle* cannot be executed if the switch is broken.

Finally, we also define *executability laws*, which stipulate the context where an action is guaranteed to be executable. In K_n, the operator $\langle a \rangle$ is used to express executability. $\langle a \rangle \top$ thus reads "the execution of a is possible".

Definition 6 An *executability law for action a* is of the form $\varphi \rightarrow \langle a \rangle \top$, where $\varphi \in \mathfrak{Fml}$.

For instance, $\neg broken \rightarrow \langle toggle \rangle \top$ says that toggling can be executed whenever the switch is not broken. The set of all executability laws of a given domain is denoted by $\mathcal{X}$.

The rest of this work is devoted to the elaboration of action models and theories.

Models of contraction

When an action theory has to be changed, the basic operation is that of *contraction*. (In belief-base update (Winslett 1988; Katsuno & Mendelzon 1992) it has also been called *erasure*.) In this section we define its semantics.

In general we might contract by any formula Φ. Here we focus on contraction by one of the three kinds of laws. We therefore suppose that Φ is either φ, where φ is classical, or $\varphi \rightarrow [a]\psi$, or $\varphi \rightarrow \langle a \rangle \top$.

For the case of contracting static laws we resort to existing approaches in order to change the set of static laws. In the following, we consider any belief change operator such as Forbus' update method (Forbus 1989), or the possible models approach (Winslett 1988; 1995), or WSS (Herzig & Rifi 1999) or MPMA (Doherty, Łukaszewicz, & Madalinska-Bugaj 1998).

Contraction by φ corresponds to adding new possible worlds to W. Let $\ominus$ be a contraction operator for classical logic.

Definition 7 Let $\langle W, R \rangle$ be a K_n-model and φ a classical formula. The set of models resulting from contracting by φ is the singleton $\langle W, R \rangle^-_\varphi = \{\langle W', R \rangle\}$ such that $W' = W \ominus val(\varphi)$.

Observe that R should, a priori, change as well, otherwise contracting a classical formula may conflict with $\mathcal{X}$.[1] For instance, if $\neg\varphi \rightarrow \langle a \rangle \top \in \mathcal{X}$ and we contract by φ, the result may make $\mathcal{X}$ untrue. However, given the amount of information we have at hand, we think that whatever we do with R (adding or removing edges), we will always be able to find a counter-example to the intuitiveness of the operation, since it is domain dependent. For instance, adding edges for a deterministic action may render it nondeterministic. Deciding on what changes to carry out on R when contracting static laws depends on the user's intuition, and unfortunately this information cannot be generalized and established once for all. We opt for a priori doing nothing with R and postponing correction of executability laws.

Action theories being defined in terms of effect and executability laws, elaborating an action theory will mainly involve changes in these two sets of laws. Let us consider now both these cases.

Suppose the knowledge engineer acquires new information regarding the effect of action a. Then it means that the law under consideration is probably too strong, i.e., the expected effect may not occur and thus the law has to be weakened. Consider e.g. $\neg up \rightarrow [toggle]light$, and suppose it has to be weakened to the more specific $(\neg up \wedge \neg blackout) \rightarrow [toggle]light$.[2] In order to carry out such a weakening, first the designer has to contract the set of effect laws and second to expand the resulting set with the weakened law.

Contraction by $\varphi \rightarrow [a]\psi$ amounts to adding some 'counterexample' arrows from φ-worlds to $\neg\psi$-worlds. To ease such a task, we need a definition. Let $PI(\varphi)$ denote the set of prime implicates of φ.

Definition 8 Let $\varphi_1, \varphi_2 \in \mathfrak{Fml}$. $NewCons_{\varphi_1}(\varphi_2) = PI(\varphi_1 \wedge \varphi_2) \setminus PI(\varphi_1)$ computes the *new consequences* of φ_2 w.r.t. φ_1: the set of strongest clauses that follow from $\varphi_1 \wedge \varphi_2$, but do not follow from φ_1 alone (cf. e.g. (Inoue 1992)).

For example, the set of prime implicates of p_1 is just $\{p_1\}$, that of the formula $p_1 \wedge (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee p_3 \vee p_4)$ is $\{p_1, p_2, p_3 \vee p_4\}$, hence we have that $NewCons_{p_1}((\neg p_1 \vee p_2) \wedge (\neg p_1 \vee p_3 \vee p_4)) = \{p_2, p_3 \vee p_4\}$.

Definition 9 Let $\langle W, R \rangle$ be a K_n-model and $\varphi \rightarrow [a]\psi$ an effect law. The models resulting from contracting by $\varphi \rightarrow [a]\psi$ is $\langle W, R \rangle^-_{\varphi \rightarrow [a]\psi} = \{\langle W, R \cup R'_a \rangle : R'_a \subseteq \{(w, w') : \models^{\langle W,R \rangle}_w \varphi, \models^{\langle W,R \rangle}_{w'} \neg\psi \text{ and } w' \setminus w \subseteq lit(NewCons_{\mathcal{S}}(\neg\psi))\}\}$.

In our context, $lit(NewCons_{\mathcal{S}}(\neg\psi))$ corresponds to all the ramifications that action a can produce.

Suppose now the knowledge engineer learns new information about the executability of a. This usually occurs when there are executability laws that are too strong, i.e., the condition in the theory guaranteeing the executability of a is too weak and has to be made more restrictive. Let e.g. $\langle toggle \rangle \top$ be the law to be contracted, and suppose it has to be weakened to the more specific $\neg broken \rightarrow \langle toggle \rangle \top$. To implement such a weakening, the designer has to first contract the set of executability laws and then to expand the resulting set with the weakened law.

Contraction by $\varphi \rightarrow \langle a \rangle \top$ corresponds to removing some arrows leaving worlds where φ holds. Removing such arrows has as consequence that a is no longer always executable in context φ.

Definition 10 Let $\langle W, R \rangle$ be a K_n-model and $\varphi \rightarrow \langle a \rangle \top$ an executability law. The set of models that result from the contraction by $\varphi \rightarrow \langle a \rangle \top$ is $\langle W, R \rangle^-_{\varphi \rightarrow \langle a \rangle \top} = \{\langle W, R' \rangle : R' = R \setminus R''_a, R''_a \subseteq \{(w, w') : wR_aw' \text{ and } \models^{\langle W,R \rangle}_w \varphi\}\}$.

In the next section we make a step toward syntactical operators that reflect the semantic foundations for contraction.

[1] We are indebted to the anonymous referees for pointing this out to us.

[2] The other possibility of weakening the law, i.e., replacing it by $\neg up \rightarrow [toggle](light \vee \neg light)$ looks silly. We were not able to find examples where changing the consequent could give a more intuitive result. In this sense, we prefer to always weaken a given law by strengthening its antecedent.

Contracting an action theory

Having established the semantics of action theory contraction, we can turn to its syntactical counterpart. Nevertheless, before doing that we have to consider an important issue. As the reader might have expected, the logical formalism of K_n alone does not solve the frame problem. For instance,

$$\left\{ \begin{array}{c} up \rightarrow light, \\ \neg up \rightarrow [toggle]up, \\ up \rightarrow [toggle]\neg up, \\ \langle toggle\rangle\top \end{array} \right\} \not\models_{\mathsf{K}_n} broken \rightarrow [toggle]broken.$$

Thus, we need a consequence relation powerful enough to deal with the frame and ramification problems. This means that the deductive power of K_n has to be augmented in order to ensure that the relevant frame axioms follow from the theory. Following the logical framework developed in (Castilho, Gasquet, & Herzig 1999), we consider metalogical information given in the form of a dependence relation:

Definition 11 A *dependence relation* is a binary relation $\rightsquigarrow \subseteq \mathfrak{Act} \times \mathfrak{Lit}$.

The expression $a \rightsquigarrow l$ denotes that the execution of action *a may* change the truth value of the literal *l*. On the other hand, $\langle a, l\rangle \notin \rightsquigarrow$ (written $a \not\rightsquigarrow l$) means that *l* can never be caused by *a*. In our example we have *toggle* $\rightsquigarrow$ *light* and *toggle* $\rightsquigarrow \neg$*light*, which means that action *toggle* may cause a change in literals *light* and $\neg$*light*. We do not have *toggle* $\rightsquigarrow \neg$*broken*, for toggling the switch never repairs it.

We assume $\rightsquigarrow$ is finite.

Definition 12 A *model of a dependence relation* $\rightsquigarrow$ is a K_n-model $\mathscr{M}$ such that $\models^{\mathscr{M}} \{\neg l \rightarrow [a]\neg l \ : \ a \not\rightsquigarrow l\}$.

Given a dependence relation $\rightsquigarrow$, the associated consequence relation in the set of models for $\rightsquigarrow$ is noted $\models_{\rightsquigarrow}$. For our example we obtain

$$\left\{ \begin{array}{c} up \rightarrow light, \\ \neg up \rightarrow [toggle]up, \\ up \rightarrow [toggle]\neg up, \\ \langle toggle\rangle\top \end{array} \right\} \models_{\rightsquigarrow} broken \rightarrow [toggle]broken.$$

We have *toggle* $\not\rightsquigarrow \neg$*broken*, i.e., $\neg$*broken* is never caused by *toggle*. Therefore in all contexts where *broken* is true, after every execution of *toggle*, *broken* still remains true. The consequence of this independence is that the frame axiom *broken* $\rightarrow [$*toggle*$]$*broken* is valid in the models of $\rightsquigarrow$.

Such a dependence-based approach has been shown (Demolombe, Herzig, & Varzinczak 2003) to subsume Reiter's solution to the frame problem (Reiter 1991) and moreover treats the ramification problem, even when actions with both indeterminate and indirect effects are involved (Castilho, Herzig, & Varzinczak 2002; Herzig & Varzinczak 2004a).

Definition 13 An *action theory* is a tuple of the form $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle$.

In our example, the corresponding action theory is

$$\mathcal{S} = \{up \rightarrow light\},\ \mathcal{E} = \left\{ \begin{array}{c} \neg up \rightarrow [toggle]up, \\ up \rightarrow [toggle]\neg up \end{array} \right\}$$

$$\mathcal{X} = \{\langle toggle\rangle\top\},\ \rightsquigarrow = \left\{ \begin{array}{c} \langle toggle, light\rangle, \\ \langle toggle, \neg light\rangle, \\ \langle toggle, up\rangle, \\ \langle toggle, \neg up\rangle \end{array} \right\}$$

And we have $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \neg up \rightarrow [toggle]light$. (For parsimony's sake, we write $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \Phi$ instead of $\mathcal{S} \cup \mathcal{E} \cup \mathcal{X} \models_{\rightsquigarrow} \Phi$.)

Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle$ be an action theory and Φ a K_n-formula. $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\Phi}$ is the action theory resulting from the contraction of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle$ by Φ.

Contracting a theory by a static law φ amounts to using any existing contraction operator for classical logic. Let $\ominus$ be such an operator. Moreover, based on (Herzig & Varzinczak 2005b), we also need to guarantee that φ does not follow from $\mathcal{E}$, $\mathcal{X}$ and $\rightsquigarrow$. We define contraction of a domain description by a static law as follows:

Definition 14 $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\varphi} = \langle \mathcal{S}^{-}, \mathcal{E}, \mathcal{X}^{-}, \rightsquigarrow\rangle$, where $\mathcal{S}^{-} = \mathcal{S} \ominus \varphi$ and $\mathcal{X}^{-} = \{(\varphi_i \wedge \varphi) \rightarrow \langle a\rangle\top \ : \ \varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}\}$.

We now consider the case of contracting an action theory by an executability law $\varphi \rightarrow \langle a\rangle\top$. For every executability in $\mathcal{X}$, we ensure that action *a* is executable only in contexts where $\neg\varphi$ is the case. The following operator does the job.

Definition 15 $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\varphi \rightarrow \langle a\rangle\top} = \langle \mathcal{S}, \mathcal{E}, \mathcal{X}^{-}, \rightsquigarrow\rangle$, where $\mathcal{X}^{-} = \{(\varphi_i \wedge \neg\varphi) \rightarrow \langle a\rangle\top \ : \ \varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}\}$.

For instance, contracting *glued* $\rightarrow \langle$*toggle*$\rangle\top$ in our example would give us $\mathcal{X}^{-} = \{\neg$*glued* $\rightarrow \langle$*toggle*$\rangle\top\}$.

Finally, to contract a theory by $\varphi \rightarrow [a]\psi$, for every effect law in $\mathcal{E}$, we first ensure that *a* still has effect ψ whenever φ does not hold, second we enforce that *a* has no effect in context $\neg\varphi$ except on those literals that are consequences of $\neg\psi$. Combining this with the new dependence relation also linking *a* to literals involved by $\neg\psi$, we have that *a* may now produce $\neg\psi$ as outcome. In other words, the effect law has been contracted. The operator below formalizes this:

Definition 16 $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\varphi \rightarrow [a]\psi} = \langle \mathcal{S}, \mathcal{E}^{-}, \mathcal{X}, \rightsquigarrow^{-}\rangle$, with $\rightsquigarrow^{-} = \rightsquigarrow \cup \{\langle a, l\rangle \ : \ l \in lit(NewCons_{\mathcal{S}}(\neg\psi))\}$ and $\mathcal{E}^{-} = \{(\varphi_i \wedge \neg\varphi) \rightarrow [a]\psi \ : \ \varphi_i \rightarrow [a]\psi \in \mathcal{E}\} \cup \{(\neg\varphi \wedge \neg l) \rightarrow [a]\neg l \ : \ \langle a, l\rangle \in (\rightsquigarrow^{-} \setminus \rightsquigarrow)\}$.

For instance, contracting the law *blackout* $\rightarrow [$*toggle*$]$*light* from our theory would give us $\mathcal{E}^{-} = \{(\neg up \wedge \neg$*blackout*$) \rightarrow [$*toggle*$]up, (up \wedge \neg$*blackout*$) \rightarrow [$*toggle*$]\neg up\}$.

Results

In this section we present the main results that follow from our framework. These require the action theory under consideration to be modular (Herzig & Varzinczak 2005b). In our framework, an action theory is said to be modular if a formula of a given type entailed by the whole theory can also be derived solely from its respective module (the set of formulas of the same type) together with the static laws $\mathcal{S}$. As shown in (Herzig & Varzinczak 2005b), to make a domain description satisfy such a property it is enough to guarantee

that there is no classical formula entailed by the theory that is not entailed by the static laws alone.

Definition 17 $\varphi \in \mathfrak{Fml}$ is an *implicit static law* of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ if and only if $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \varphi$ and $\mathcal{S} \not\models \varphi$.

A theory is *modular* if it has no implicit static laws. Our concept of modularity of theories was originally defined in (Herzig & Varzinczak 2004b; 2005b), but similar notions have also been addressed in the literature (Cholvy 1999; Amir 2000a; Zhang, Chopra, & Foo 2002; Lang, Lin, & Marquis 2003; Herzig & Varzinczak 2005a). A modularity-based approach for narrative reasoning about actions is given in (Kakas, Michael, & Miller 2005).

To witness how implicit static laws can show up, consider the quite simple action theory below, depicting the walking turkey scenario (Thielscher 1995):

$$\mathcal{S} = \{walking \rightarrow alive\},\ \mathcal{E} = \left\{ \begin{array}{c} [tease]walking, \\ loaded \rightarrow [shoot]\neg alive \end{array} \right\}$$

$$\mathcal{X} = \{\ \langle tease\rangle\top, \langle shoot\rangle\top\ \},$$

$$\rightsquigarrow = \left\{ \begin{array}{c} \langle shoot, \neg loaded\rangle, \langle shoot, \neg alive\rangle, \\ \langle shoot, \neg walking\rangle, \langle tease, walking\rangle \end{array} \right\}$$

With this domain description we have $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} alive$: first, $\{walking \rightarrow alive, [tease]walking\} \models_{\rightsquigarrow} [tease]alive$, second $\models_{\rightsquigarrow} \neg alive \rightarrow [tease]\neg alive$ (from the independence $tease \not\rightsquigarrow alive$), and then $\mathcal{S}, \mathcal{E} \models_{\rightsquigarrow} \neg alive \rightarrow [tease]\bot$. As long as $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \langle tease\rangle\top$, we must have $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} alive$. As $\mathcal{S} \not\models alive$, the formula *alive* is an implicit static law of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$.

Modular theories have several advantages (Herzig & Varzinczak 2004b;). For example, consistency of a modular action theory can be checked by just checking consistency of $\mathcal{S}$: if $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ is modular, then $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \bot$ if and only if $\mathcal{S} \models \bot$. Deduction of an effect of a sequence of actions $a_1; \ldots; a_n$ (prediction) does not need to take into account the effect laws for actions other than $a_1, \ldots, a_n$. This applies in particular to plan validation when deciding whether $\langle a_1; \ldots; a_n\rangle\varphi$ is the case.

Throughout this work we have used multimodal logic K_n. For an assessment of the modularity principle in the Situation Calculus, see (Herzig & Varzinczak 2005a).

Here we establish that our operators are correct w.r.t. the semantics. Our first theorem establishes that the semantical contraction of the models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ by Φ produces models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$.

Theorem 1 Let $\langle W, R\rangle$ be a model of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$, and let Φ be a formula that has the form of one of the three laws. For all models $\mathscr{M}$, if $\mathscr{M} \in \langle W, R\rangle^-_\Phi$, then $\mathscr{M}$ is a model of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$.

It remains to prove that the other way round, the models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$ result from the semantical contraction of models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ by Φ. This does not hold in general, as shown by the following example: suppose there is only one atom p and one action a, and consider the theory $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ such that $\mathcal{S} = \emptyset$, $\mathcal{E} = \{p \rightarrow [a]\bot\}$, $\mathcal{X} = \{\langle a\rangle\top\}$, and $\rightsquigarrow = \emptyset$. The only model of that action theory is $\mathscr{M} = \langle\{\{\neg p\}\}, \{(\{\neg p\}, \{\neg p\})\}\rangle$. By definition, $\mathscr{M}^-_{p\rightarrow\langle a\rangle\top} = \{\mathscr{M}\}$. On the other hand, $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_{p\rightarrow\langle a\rangle\top} = \langle\emptyset, \{p \rightarrow [a]\bot\}, \{\neg p \rightarrow \langle a\rangle\top\}, \emptyset\rangle$. The contracted theory has *two* models: $\mathscr{M}$ and $\mathscr{M}' = \langle\{\{p\}, \{\neg p\}\}, (\{\neg p\}, \{\neg p\})\rangle$. While $\neg p$ is valid in the contraction of the models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$, it is invalid in the models of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_{p\rightarrow\langle a\rangle\top}$.

Fortunately, we can establish a result for those action theories that are modular. The proof requires three lemmas. The first one says that for a modular theory we can restrict our attention to its 'big' models.

Lemma 1 Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ be modular. Then $\mathcal{S}, \mathcal{E}, \mathcal{X} \models_{\rightsquigarrow} \Phi$ if and only if $\models^{\langle W,R\rangle} \Phi$ for every model $\langle W, R\rangle$ of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ such that $W = val(\mathcal{S})$.

Note that the lemma does not hold for non-modular theories, as $\{\langle W, R\rangle : \langle W, R\rangle$ is a model of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ and $W = val(\mathcal{S})\}$ is empty then.

The second lemma says that modularity is preserved under contraction.

Lemma 2 Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ be modular, and let Φ be a formula of the form of one of the three laws. Then $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$ is modular.

The third one establishes the required link between the contraction operators and contraction of 'big' models.

Lemma 3 Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ be modular, and let Φ be a formula of the form of one of the three laws. If $\mathscr{M}' = \langle val(\mathcal{S}), R'\rangle$ is a model of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$, then there is a model $\mathscr{M}$ of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ such that $\mathscr{M}' \in \mathscr{M}^-_\Phi$.

Putting the three above lemmas together we get:

Theorem 2 Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ be modular, Φ be a formula of the form of one of the three laws, and $\langle \mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^-, \rightsquigarrow^- \rangle$ be $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$. If it holds that $\mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^- \models_{\rightsquigarrow^-} \Psi$, then for every model $\mathscr{M}$ of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ and every $\mathscr{M}' \in \mathscr{M}^-_\Phi$ it holds that $\models^{\mathscr{M}'} \Psi$.

Our two theorems together establish correctness of the operators:

Corollary 1 Let $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ be modular, Φ be a formula of the form of one of the three laws, and $\langle \mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^-, \rightsquigarrow^- \rangle$ be $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$. Then $\mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^- \models_{\rightsquigarrow^-} \Psi$ if and only if for every model $\mathscr{M}$ of $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ and every $\mathscr{M}' \in \mathscr{M}^-_\Phi$ it holds that $\models^{\mathscr{M}'} \Psi$.

We give a necessary condition for success of contraction:

Theorem 3 Let Φ be an effect or an executability law such that $\mathcal{S} \not\models_{\mathsf{K}_n} \Phi$. Let $\langle \mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^-, \rightsquigarrow^- \rangle$ be $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_\Phi$. If $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ is modular, then $\mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^- \not\models_{\rightsquigarrow^-} \Phi$.

Contracting implicit static laws

There can be many reasons why a theory should be changed. Following (Herzig & Varzinczak 2004b; 2005b;), here we focus on the case where it has some classical consequence φ the designer is not aware of.

If φ is taken as intuitive, then, normally, no change has to be done at all, unless we want to keep abide on the modularity principle and thus make φ explicit by adding it to $\mathcal{S}$. In the scenario example of last section, if the knowledge engineer's universe has immortal turkeys, then she would add the static law *alive* to $\mathcal{S}$.

The other way round, if φ is not intuitive, as long as φ is entailed by $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$, the goal is to avoid such an entailment, i.e., what we want is $\mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^- \not\models_{\rightsquigarrow^-} \varphi$, where $\langle \mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^-, \rightsquigarrow^- \rangle$ is $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle^-_{\Phi}$. In the mentioned scenario, the knowledge engineer considers that having immortal turkeys is not reasonable and thus decides to change the domain description to $\langle \mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^-, \rightsquigarrow^- \rangle$ so that $\mathcal{S}^-, \mathcal{E}^-, \mathcal{X}^- \not\models_{\rightsquigarrow^-}$ *alive*.

This means that action theories that are not modular need to be changed, too. Such a changing process is driven by the problematic part of the theory detected by the algorithms defined in (Herzig & Varzinczak 2004b) and improved in (Herzig & Varzinczak).

The algorithm works as follows: for each executability law $\varphi \rightarrow \langle a \rangle \top$ in the theory, construct from $\mathcal{E}$ and $\rightsquigarrow$ a set of inexecutabilities $\{\varphi_1 \rightarrow [a]\bot, \ldots, \varphi_n \rightarrow [a]\bot\}$ that potentially conflict with $\varphi \rightarrow \langle a \rangle \top$. For each i, $1 \leq i \leq n$, if $\varphi \wedge \varphi_i$ is satisfiable w.r.t. $\mathcal{S}$, mark $\neg(\varphi \wedge \varphi_i)$ as an implicit static law. Incrementally repeat this procedure (adding all the $\neg(\varphi \wedge \varphi_i)$ that were caught to $\mathcal{S}$) until no implicit static law is obtained.

For an example of the execution of the algorithm, consider $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow \rangle$ as above. For the action *tease*, we have the executability $\langle \textit{tease} \rangle \top$. Now, from $\mathcal{E}$, and $\rightsquigarrow$ we try to build an inexecutability for *tease*. We take $[\textit{tease}]\textit{walking}$ and compute then all indirect effects of *tease* w.r.t. $\mathcal{S}$. From *walking* $\rightarrow$ *alive*, we get that *alive* is an indirect effect of *tease*, giving us $[\textit{tease}]\textit{alive}$. But $\langle \textit{tease}, \textit{alive} \rangle \notin \rightsquigarrow$, which means the frame axiom $\neg\textit{alive} \rightarrow [\textit{tease}]\neg\textit{alive}$ holds. Together with $[\textit{tease}]\textit{alive}$, this gives us the inexecutability $\neg\textit{alive} \rightarrow [\textit{tease}]\bot$. As $\mathcal{S} \cup \{\top, \neg\textit{alive}\}$ is satisfiable ($\top$ is the antecedent of the executability $\langle \textit{tease} \rangle \top$), we get the implicit static law *alive*. For this example no other inexecutability for *tease* can be derived, so the computation stops.

It seems that in general implicit static laws are not intuitive. Therefore their contraction is more likely to happen than their addition.[3] In the example above, the action theory has to be contracted by *alive*.[4] In order to contract the action theory, the designer has several choices:

1) Contract the set $\mathcal{S}$. (In this case, such an operation is not enough, since *alive* is a consequence of the rest of the theory.)

2) Weaken the effect law $[\textit{tease}]\textit{walking}$ to $\textit{alive} \rightarrow [\textit{tease}]\textit{walking}$, since the original effect law is too strong. This means that in a first stage the designer has to contract the theory and in a second one expand the effect laws with the weaker law. The designer will usually choose this option if she focuses on the preconditions of the effects of actions.

3) Weaken the executability law $\langle \textit{tease} \rangle \top$ by rephrasing it as $\textit{alive} \rightarrow \langle \textit{tease} \rangle \top$: first the executability is contracted and then the weaker one is added to the resulting set of executability laws. The designer will choose this option if she focuses on preconditions for action execution.

The analysis of this example shows that the choice of what change has to be carried out is up to the knowledge engineer. Such a task can get more complicated when ramifications are involved. To witness, suppose our scenario has been formalized as follows: $\mathcal{S} = \{\textit{walking} \rightarrow \textit{alive}\}$, $\mathcal{E} = \{[\textit{shoot}]\neg\textit{alive}\}$, $\mathcal{X} = \{\langle \textit{shoot} \rangle \top\}$, and $\rightsquigarrow = \{\langle \textit{shoot}, \neg\textit{alive} \rangle\}$. From this action theory we can derive the inexecutability $\textit{walking} \rightarrow [\textit{shoot}]\bot$ and thus the implicit static law $\neg\textit{walking}$. In this case we have to change the theory by contracting the frame axiom $\textit{walking} \rightarrow [\textit{shoot}]\textit{walking}$ (which amounts to adding the missing indirect dependence $\textit{shoot} \rightsquigarrow \neg\textit{walking}$).

Elaboration tolerance

The principle of elaboration tolerance has been proposed by McCarthy (McCarthy 1988). Roughly, it states that the effort required to add new information to a given representation (new laws or entities) should be proportional to the complexity of the information being added, i.e., it should not require the complete reconstruction of the old theory (Shanahan 1997).

Since then many formalisms in the reasoning about actions field claim, in a more or less tacit way, to satisfy such a principle. However, for all this time there has been a lack of good formal criteria allowing for the evaluation of theory change difficulty and, consequently, comparisons between different frameworks are carried out in a subjective way.

The proposal by Amir (Amir 2000b) made the first steps in formally answering what difficulty of changing a theory means by formalizing one aspect of elaboration tolerance. The basic idea is as follows: let $\mathcal{T}_0$ be the original theory and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two equivalent (and different) theories such that each one results from $\mathcal{T}_0$ by the application of some sequence of operations (additions and/or deletions of formulas). The resulting theory whose transformation from $\mathcal{T}_0$ has the shortest length (number of operations) is taken as the most elaboration tolerant.

Nevertheless, in the referred work only addition/deletion of *axioms* is considered, i.e., changes in the logical language or contraction of consequences of the theory not explicitly stated in the original set of axioms are not taken into account. This means that even the formal setting given in (Amir 2000b) is not enough to evaluate the complexity of

[3] In all the examples in which we have found implicit static laws that are intuitive they are so evident that the only explanation for not having them explicitly stated is that they have been forgotten by the theory's designer.

[4] Here the change operation is a revision-based operation rather than an update-based operation since we mainly "fix" the theory.

theory change in a broad sense. Hence the community still needs formal criteria that allow for the comparison between more complex changes carried out by frameworks like ours, for example.

Of course, how elaboration tolerant a given update/revision method is strongly depends on its underlying formalism for reasoning about actions, i.e., its logical background, the solution to the frame problem it implements, the hypothesis it relies on, etc. In what follows we discuss how the dependence-based approach here used behaves when expansion is considered. Most of the comments concerning consequences of expansion can also be stated for contraction. We do that with respect to some of the qualitative criteria given in (McCarthy 1998). In all that follows we suppose that the resulting theory is consistent.

Adding effect laws In the dependence-based framework, adding the new effect law $\varphi \rightarrow [a]\psi$ to the theory demands a change in the dependence module $\rightsquigarrow$. In that case, the maximum number of statements added to $\rightsquigarrow$ is $|\{l : l \in lit(NewCons_{\mathcal{S}}(\psi))\}|$ (dependences for all indirect effects have to be stated, too). This is due to the explanation closure nature of the reasoning behind dependence (for more details, see (Castilho, Gasquet, & Herzig 1999)). Because of this, according to Shanahan (Shanahan 1997), explanation closure approaches are not elaboration tolerant when dealing with the ramification problem. In order to achieve that, the framework should have a mechanism behaving like circumscription that automatically deals with ramifications. This raises the question: "if we had an automatic (or even semi-automatic) procedure to do the job of generating the indirect dependences, could we say the framework is elaboration tolerant?". We think we can answer positively to such a question, and, supported by Reiter (Reiter 2001), we are working on a semi-automatic procedure for generating the dependence relation from a set of effect laws.

Adding executability laws Such a task demands only a change in the set $\mathcal{X}$ of executabilities, possibly introducing implicit static laws as a side effect.

Adding static laws Besides expanding the set $\mathcal{S}$, adding new (indirect) dependences may be required (see above).

Adding frame axioms If the frame axiom $\neg l \rightarrow [a]\neg l$ has to be valid in the resulting theory, expunging the dependence $a \rightsquigarrow l$ should do the job.

Adding a new action name Without loss of generality we can assume the action in question was already in the language. In that case, we expect just to add effect or executability laws for it. For the former, at most $|\mathfrak{Lit}|$ dependences will be added to $\rightsquigarrow$. (We point out nevertheless that the requirement made in (McCarthy 1998) that the addition of an action irrelevant for a given plan in the old theory should not preclude it in the resulting theory is too strong. Indeed, it is not difficult to imagine a new action forcing an implicit static law from which an inexecutability for some action in the plan can be derived. The same holds for the item below.)

Adding a new fluent name In the same way, we can suppose the fluent was already in the language. Such a task amounts thus to one or more of the above expansions. There will be at most $2 \times |\mathfrak{Act}|$ new elements added to $\rightsquigarrow$.

Related work

Following (Li & Pereira 1996; Liberatore 2000), Eiter *et al.* (Eiter *et al.* 2005) have investigated update of action domain descriptions. They define a version of action theory update in an action language and give complexity results showing how hard such a task can be.

Update of action descriptions in their sense is always relative to some conditions (interpreted as knowledge possibly obtained from earlier observations and that should be kept). This characterizes a constraint-based update. In the example they give, change must be carried out preserving the assumption that pushing the button of the remote control is always executable. Actually, the method is more subtle, as new effect laws are added constrained by *the addition* of viz. an executability law for the new action under concern. In the example, the constraint (executability of push) was not in the original action description and must figure in the updated theory.

They describe domains of actions in a fragment of the action language $\mathcal{C}$ (Gelfond & Lifschitz 1998). However they do not specify which fragment, so it is not clear whether the claimed advantages $\mathcal{C}$ has over $\mathcal{A}$ really transfer to their framework. At one hand, their approach deals with indirect effects, but they do not talk about updating a theory by a law with a nondeterministic action. Anyway, except for concurrency, their account can be translated into ours, as shown in (Castilho, Gasquet, & Herzig 1999).

Eiter *et al.* consider an action theory $\mathcal{T}$ as comprising two main components: $\mathcal{T}_u$, the part of the theory that must remain unchanged, and $\mathcal{T}_m$, the part concerning the statements that are allowed to change. The crucial information to the associated solution to the frame problem is always in $\mathcal{T}_u$.

Given an action theory $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_m$, $((\mathcal{T}_u \cup \mathcal{T}_m), \mathcal{T}', C)$ is the problem of updating $\mathcal{T}$ by $\mathcal{T}' \subseteq \mathcal{S} \cup \mathcal{E}$ warranting the result satisfies all constraints in $C \subseteq \mathcal{S} \cup \mathcal{X}$.

Even though they do not explicitly state postulates for their kind of theory update, they establish conditions for the update operator to be successful. Basically, they claim for consistency of the resulting theory; maintenance of the new knowledge and the invariable part of the description; satisfaction of the constraints in C; and minimal change.

In some examples that they develop, the illustrated "partial solution" does not satisfy C due to the existence of implicit laws (cf. Example 1, where there is an implicit inexecutability law). To achieve a solution, while keeping C, some other laws must be dropped (in the example, the agent gives up a static law).[5]

Just to see the link between update by subsumed laws and addition of implicit static laws, we note that Proposition 1 in the referred work is the same as Theorem 14 in (Herzig & Varzinczak 2005b): every implicit static law in Herzig and Varzinczak's sense is trivially a subsumed law in Eiter *et al.*'s sense.

[5]This does not mean however that the updated theory will necessarily contain no implicit law.

With their method we can also contract by a static and an effect law. Contraction of executabilities are not explicitly addressed, and weakening (replacing a law by a weaker one) is left as future work.

A main difference between the approach in (Eiter *et al.* 2005) and ours is that we do not need to add new fluents at every elaboration stage: we still work on the same set of fluents, refining their behavior w.r.t. an action a. In Eiter *et al.*'s proposal an update forces changing all the variable rules appearing in the action theory by adding to each one a new update fluent. This is a constraint when elaborating action theories.

Concluding remarks

In this work we have presented a general method for changing a domain description (alias action theory) given any formula we want to contract.

We have defined a semantics for theory contraction and also presented its syntactical counterpart through contraction operators. Soundness and completeness of such operators with respect to the semantics have been established (Corollary 1).

We have also shown that modularity is a necessary condition for a contraction to be successful (Theorem 3). This gives further evidence that our modularity notion is fruitful.

We have analysed an example of contraction of a non-modular theory by an implicit static law that is unintended.

Because of forcing formulas to be explicitly stated in their respective modules (and thus possibly making them inferable in independently different ways), intuitively modularity could be seen to diminish elaboration tolerance. For instance, when contracting a classical formula φ from a non-modular theory, it seems reasonable to expect not to change the set of static laws $\mathcal{S}$, while the theory being modular surely forces changing such a module. However it is not difficult to conceive non-modular theories in which contraction of a formula φ may demand a change in $\mathcal{S}$ as well. To witness, suppose $\mathcal{S} = \{\varphi_1 \rightarrow \varphi_2\}$ in an action theory from whose dynamic part we (implicitly) infer $\neg\varphi_2$. In this case, a contraction of $\neg\varphi_1$ keeping $\neg\varphi_2$ would necessarily ask for a change in $\mathcal{S}$. We point out nevertheless that in both cases (modular and non-modular) the extra work in changing other modules stays in the mechanical level, i.e., in the machinery that carries out the modification, and does not augment in a significant way the amount of work the knowledge engineer is expected to do.

What is the status of the AGM-postulates for contraction in our framework? First, contraction of static laws satisfies all the postulates, as soon as the underlying classical contraction operation $\ominus$ satisfies all of them.

In the general case, however, our constructions do not satisfy the central postulate of preservation $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\Phi} = \langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle$ if $\mathcal{S}, \mathcal{E}, \mathcal{X} \not\models_{\rightsquigarrow} \Phi$. Indeed, suppose we have a language with only one atom p, and a model $\mathscr{M}$ with two worlds $w = \{p\}$ and $w' = \{\neg p\}$ such that wR_aw', $w'R_aw$, and $w'R_aw'$. Then $\models^{\mathscr{M}} p \rightarrow [a]\neg p$ and $\not\models^{\mathscr{M}} [a]\neg p$, i.e., $\mathscr{M}$ is a model of the effect law $p \rightarrow [a]\neg p$, but not of $[a]\neg p$. Now the contraction $\mathscr{M}^{-}_{[a]\neg p}$ yields the model $\mathscr{M}'$ such that $R_a = W \times W$. Then $\not\models^{\mathscr{M}'} p \rightarrow [a]\neg p$, i.e., the effect law $p \rightarrow [a]\neg p$ is not preserved. Our contraction operation thus behaves rather like an update operation.

Now let us focus on the other postulates. Since our operator has a behavior which is close to the update postulate, we focus on the following basic erasure postulates introduced in (Katsuno & Mendelzon 1991). Let $Cn(\mathcal{T})$ be the set of all logical consequences of a theory $\mathcal{T}$.

KM1 $Cn(\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\Phi}) \subseteq Cn(\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle)$

Postulate **KM1** does not always hold because it is possible to make the formula $\varphi \rightarrow [a]\bot$ valid in the resulting theory by removing elements of R_a (cf. Definition 10).

KM2 $\Phi \notin Cn(\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle^{-}_{\Phi})$

Under the condition that $\langle \mathcal{S}, \mathcal{E}, \mathcal{X}, \rightsquigarrow\rangle$ is modular, Postulate **KM2** is satisfied (cf. Theorem 3).

KM3 If $Cn(\langle \mathcal{S}_1, \mathcal{E}_1, \mathcal{X}_1, \rightsquigarrow\rangle) = Cn(\langle \mathcal{S}_2, \mathcal{E}_2, \mathcal{X}_2, \rightsquigarrow\rangle)$ and $\models_{\mathsf{K}_n} \Phi_1 \leftrightarrow \Phi_2$, then $Cn(\langle \mathcal{S}_1, \mathcal{E}_1, \mathcal{X}_1, \rightsquigarrow\rangle^{-}_{\Phi_2}) = Cn(\langle \mathcal{S}_2, \mathcal{E}_2, \mathcal{X}_2, \rightsquigarrow\rangle^{-}_{\Phi_1})$.

Theorem 4 If $\langle \mathcal{S}_1, \mathcal{E}_1, \mathcal{X}_1, \rightsquigarrow\rangle$ and $\langle \mathcal{S}_2, \mathcal{E}_2, \mathcal{X}_2, \rightsquigarrow\rangle$ are modular and the propositional contraction operator $\ominus$ satisfies Postulate **KM3**, then Postulate **KM3** is satisfied for every $\Phi_1, \Phi_2 \in \mathfrak{Fml}$.

Here we have presented the case for contraction, but our definitions can be extended to revision, too. Our results can also be generalized to the case where learning new actions or fluents is involved. This means in general that more than one simple formula should be added to the belief base and must fit together with the rest of the theory with as little side-effects as possible. We are currently defining algorithms based on our operators to achieve that.

Acknowledgments

We are grateful to the anonymous referees for many useful comments on an earlier version of this paper.

References

Amir, E. 2000a. (De)composition of situation calculus theories. In *Proc. AAAI'2000*, 456–463. AAAI Press/MIT Press.

Amir, E. 2000b. Toward a formalization of elaboration tolerance: Adding and deleting axioms. In *Frontiers of Belief Revision*. Kluwer.

Castilho, M. A.; Gasquet, O.; and Herzig, A. 1999. Formalizing action and change in modal logic I: the frame problem. *J. of Logic and Computation* 9(5):701–735.

Castilho, M. A.; Herzig, A.; and Varzinczak, I. J. 2002. It depends on the context! a decidable logic of actions and plans based on a ternary dependence relation. In *NMR'02*, 343–348.

Cholvy, L. 1999. Checking regulation consistency by using SOL-resolution. In *Proc. Int. Conf. on AI and Law*, 73–79.

Demolombe, R.; Herzig, A.; and Varzinczak, I. 2003. Regression in modal logic. *J. of Applied Non-Classical Logics (JANCL)* 13(2):165–185.

Doherty, P.; Łukaszewicz, W.; and Madalinska-Bugaj, E. 1998. The PMA and relativizing change for action update. In *Proc. KR'98*, 258–269. Morgan Kaufmann.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating action domain descriptions. In *Proc. IJCAI'05*, 418–423. Morgan Kaufmann.

Foo, N. Y., and Zhang, D. 2002. Dealing with the ramification problem in the extended propositional dynamic logic. In *Advances in Modal Logic*, volume 3. World Scientific. 173–191.

Forbus, K. D. 1989. Introducing actions into qualitative simulation. In *Proc. IJCAI'89*, 1273–1278. Morgan Kaufmann.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *ETAI* 2(3–4):193–210.

Hansson, S. O. 1999. *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Kluwer.

Harel, D. 1984. Dynamic logic. In *Handbook of Philosophical Logic*, volume II. D. Reidel, Dordrecht. 497–604.

Herzig, A., and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence* 115(1):107–138.

Herzig, A., and Varzinczak, I. Metatheory of actions: beyond consistency. To appear.

Herzig, A., and Varzinczak, I. 2004a. An assessment of actions with indeterminate and indirect effects in some causal approaches. Technical Report 2004–08–R, Institut de recherche en informatique de Toulouse (IRIT), Université Paul Sabatier.

Herzig, A., and Varzinczak, I. 2004b. Domain descriptions should be modular. In *Proc. ECAI'04*, 348–352. IOS Press.

Herzig, A., and Varzinczak, I. 2005a. Cohesion, coupling and the meta-theory of actions. In *Proc. IJCAI'05*, 442–447. Morgan Kaufmann.

Herzig, A., and Varzinczak, I. 2005b. On the modularity of theories. In *Advances in Modal Logic*, volume 5. King's College Publications. 93–109.

Inoue, K. 1992. Linear resolution for consequence finding. *Artificial Intelligence* 56(2–3):301–353.

Jin, Y., and Thielscher, M. 2005. Iterated belief revision, revised. In *Proc. IJCAI'05*, 478–483. Morgan Kaufmann.

Kakas, A.; Michael, L.; and Miller, R. 2005. $\mathcal{M}$odular-$\mathcal{E}$: an elaboration tolerant approach to the ramification and qualification problems. In *Proc. 8th Intl. Conf. Logic Programming and Nonmonotonic Reasoning*, 211–226. Springer-Verlag.

Katsuno, H., and Mendelzon, A. O. 1991. Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52(3):263–294.

Katsuno, H., and Mendelzon, A. O. 1992. On the difference between updating a knowledge base and revising it. In Gärdenfors, P., ed., *Belief revision*. Cambridge University Press. 183–203.

Lang, J.; Lin, F.; and Marquis, P. 2003. Causal theories of action – a computational core. In *Proc. IJCAI'03*, 1073–1078. Morgan Kaufmann.

Li, R., and Pereira, L. 1996. What is believed is what is explained. In *Proc. AAAI'96*, 550–555. AAAI Press/MIT Press.

Liberatore, P. 2000. A framework for belief update. In *Proc. JELIA'2000*, 361–375.

McCarthy, J. 1988. *Mathematical logic in artificial intelligence*. Daedalus.

McCarthy, J. 1998. Elaboration tolerance. In *Proc. Common Sense'98*.

Popkorn, S. 1994. *First Steps in Modal Logic*. Cambridge University Press.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press. 359–380.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Shanahan, M. 1997. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. Cambridge, MA: MIT Press.

Shapiro, S.; Pagnucco, M.; Lespérance, Y.; and Levesque, H. J. 2000. Iterated belief change in the situation calculus. In *Proc. KR'2000*, 527–538. Morgan Kaufmann.

Thielscher, M. 1995. Computing ramifications by post-processing. In *Proc. IJCAI'95*, 1994–2000. Morgan Kaufmann.

Winslett, M.-A. 1988. Reasoning about action using a possible models approach. In *Proc. AAAI'88*, 89–93. Morgan Kaufmann.

Winslett, M.-A. 1995. Updating logical databases. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4. Oxford University Press. 133–174.

Zhang, D., and Foo, N. Y. 2001. EPDL: A logic for causal reasoning. In *Proc. IJCAI'01*, 131–138. Morgan Kaufmann.

Zhang, D.; Chopra, S.; and Foo, N. Y. 2002. Consistency of action descriptions. In *PRICAI'02, Topics in Artificial Intelligence*. Springer-Verlag.

6.4 Merging Rules

Merging Rules: Preliminary Version

Richard Booth
Faculty of Informatics
Mahasarakham University
Mahasarakham 44150, Thailand
richard.b@msu.ac.th

Souhila Kaci
CRIL
Rue de l'Université SP 16
62307 Lens Cedex, France
kaci@cril.univ-artois.fr

Leendert van der Torre
ILIAS
University of Luxembourg
Luxembourg
leon.vandertorre@uni.lu

Abstract

In this paper we consider the merging of rules or conditionals. In contrast to other approaches, we do not invent a new approach from scratch, for one particular kind of rule, but we are interested in ways to generalize existing revision and merging operators from belief merging to rule merging. First, we study ways to merge rules based on only a notion of consistency of a set of rules, and illustrate this approach using a consolidation operator of Booth and Richter. Second, we consider ways to merge rules based on a notion of implication among rules, and we illustrate this approach using so-called $\min$ and $\max$ merging operators defined using possibilistic logic.

Introduction

We are interested in the merging or fusion of rules or conditionals. When there are several sources of rules that are in some sense conflicting, incoherent or inconsistent, then a rule merging operator returns a weaker non-conflicting set of rules. Such merging operators can be used in many areas of artificial intelligence, for example when merging regulations in electronic institutions, merging conditional default rules, or merging conditional goals in social agent theory.

In general, there are two approaches to develop operators and algorithms to merge rules. One approach starts from a particular kind of rule, and develops a merging operator for a particular application domain. The other approach tries to generalize existing operators from belief merging, which have been developed as a generalization of belief revision operators. In this paper we follow in the latter approach, and we address the following research questions:

1. How to define a general framework to study rule merging and develop rule merging operators?
2. Given a merging operator for belief merging, how can we use it for rule merging?
3. Defeasible rules can be stratified into a prioritized rule base. How can we use this stratification in rule merging?

Though many notions of rules have been defined, they are typically represented as syntactic objects $\phi \rightarrow \psi$ in a meta-language, expressing a conditional statement "if ϕ then ψ", where ϕ and ψ are formulas of an object language, for example propositional or first-order logic. Given a set of such rules R expressed as pairs of formulas of a language L, we can apply the rules to a context S, consisting of formulas of L, which results again in a set of sentences of L. In this paper, following conventions from input/output logic (Makinson & van der Torre 2000), we write $out(R, S)$ for the result of applying the rules of R to S.

A crucial ingredient of belief merging operators is a notion of inconsistency. However, since rules are typically represented in the meta-language, there is no obvious choice of rule inconsistency which can be used. To use a merging operator for rule merging, we have to define when a set of rules is conflicting or contradictory. We discuss various ways to define the inconsistency of a set of rules, and illustrate how a merging operator for belief merging can be used for rule merging using a generalization of the so-called AGM partial meet approach (Alchourrón, Gärdenfors, & Makinson 1985; Hansson 1999) due to (Booth & Richter 2005).

Moreover, a notion of consistency is sufficient to define selection operators, but in general we need also a notion of implication among rules. For example, when we interpret the arrow $\rightarrow$ as an implication in conditional logic, or as the material implication of propositional logic, then two rules $\phi \rightarrow \psi$ and $\xi \rightarrow \varphi$ imply the rule $\phi \wedge \xi \rightarrow \psi \vee \varphi$. Moreover, if we merge the former two rules, the latter one may be the result. We illustrate this using merging operators from possibilistic logic (Dubois, Lang, & Prade 1994), a logic that associates with a formula a numerical value between 0 and 1. One interpretation of this value is that it represents a stratification of the formulas in the knowledge base in formulas with higher and lower priority. A particular kind of conditionals has been defined, and these conditionals have been stratified using a stratification algorithm.

The layout of this paper is as follows. We first discuss the inconsistency of a set of rules, and illustrate it on the merging operator of Booth and Richter. Then we discuss rule implication, and illustrate it on merging operators defined using possibilistic logic.

Preliminaries: Unless otherwise indicated, our background logic in this paper will be a propositional logic L containing the usual propositional connectives, including material implication which we denote by $\supset$. For any set of formulas S, $Cn(S)$ is the set of logical consequences of S. We will say S is Cn-consistent if S is classically consistent. Ω is the set of all propositional interpretations relative to L.

Rules, alias conditionals, will be of the form $\phi \to \psi$ where $\phi, \psi \in L$. Thus L^2 is the set of all rules.

Rule consistency

Applying rules

In this paper we make only minimal assumptions on $out(R, S)$ in general. We assume $out(R, S)$ is a logically closed set of formulas of L. We also assume that a rule can be applied when the context is precisely the body of the rule, and that a set of rules cannot imply more than the materialization of the rules, that is, then assuming that the set of rules are formulas of L by interpreting the condition as a material implication. There are many additional properties one may want to impose on the application of rules.

Let R be a set of pairs from a logic L, let S be a set of formulas of L, $out(R, S) \subseteq L$ is assumed to satisfy the following conditions:

1. $out(R, S) = Cn(out(R, S))$
2. If $\phi \to \psi \in R$, then $\psi \in out(R, \{\phi\})$;
3. $out(R, S) \subseteq Cn(S \cup \{\phi \supset \psi \mid \phi \to \psi \in R\})$

Seven kinds of such rules have been studied in the input/output logic framework (Makinson & van der Torre 2000). One example, called *simple-minded output* in (Makinson & van der Torre 2000), is

$$out_1(R, S) = Cn(\{\psi \in L \mid (\phi \to \psi) \in R \text{ and } \phi \in Cn(S)\}).$$

We will refer to this operation again later in this section. Many other examples can be defined. They satisfy additional properties, such as the monotonicity property that the output $out(R, S)$ increases if either R or S increases.

Consistency of output

Since rules are defined as pairs of formulas of L, we can define the consistency of a set of rules in terms of Cn-consistency. In input/output logic, the following two notions of consistency have been defined for a set of rules relative to a given context S (Makinson & van der Torre 2001):

Output constraint A set of rules R satisfies the output constraint when $out(R, S)$ is Cn-consistent.

Input/output constraint A set of rules R satisfies the input/output constraint when $S \cup out(R, S)$ is Cn-consistent.

When the application of a set of rules $out(R, S)$ always contains the input S, then these two kinds of constraints obviously coincide. However, there are several intuitive notions of rules, such as norms, which do not have this property, and where the two constraints are distinct.

Rule consistency

We consider a weak and a strong notion of consistency of a set of rules. A set is weakly consistent when it does not lead to Cn-inconsistent output for the inputs of the available rules, and it is strongly consistent when it does not lead to Cn-inconsistency for any consistent context. Strong consistency is sometimes used, for example, when developing institutional regulations.

Weak consistency For all $\phi \to \psi \in R$, we have $out(R, \{\phi\})$ is Cn-consistent.

Strong consistency For any Cn-consistent $S \subseteq L$, we have that $out(R, S)$ is Cn-consistent.

Example: A consolidation operator

(Booth & Richter 2005) assume a very abstract framework based on the abstract framework for fuzzy logic due to (Gerla 2001). They just need three ingredients:

(i) a set L_0 of formulas,

(ii) a set W of abstract *degrees* which can be assigned to the formulas in L_0 to create *fuzzy belief bases*, and

(iii) a special subset $\mathbf{Con}$ of these fuzzy belief bases which specifies those fuzzy bases which are meant to be *consistent*, in whatever sense.[1]

For the set L_0 they assume no particular structure – it is just an arbitrary set, while the only thing assumed about W in general is that it is a complete distributive lattice. Formally, a fuzzy belief base is a function $u : L_0 \to W$. $\mathcal{F}(L_0)$ denotes the set of all fuzzy bases. If $u(\varphi) = a$, then this is interpreted as the information that the degree of φ is *at least* a, i.e., it *could* in actual fact be bigger than a, but the information contained in u doesn't allow us to be more precise. The partial order over W is denoted by $\leq_W$. The "fuzzy" subset relation $\sqsubseteq$ between fuzzy bases is defined by

$$u \sqsubseteq v \text{ iff } u(\varphi) \leq_W v(\varphi) \text{ for all } \varphi \in L_0.$$

So $\sqsubseteq$ is an "information ordering": $u \sqsubseteq v$ iff the information contained in v is more "precise" than u. Under these definitions $(\mathcal{F}(L_0), \sqsubseteq)$ itself forms a complete distributive lattice. Given any set $X \subseteq \mathcal{F}(L_0)$ of fuzzy bases the infimum and supremum of X under $\sqsubseteq$ are denoted by $\bigsqcap X$ and $\bigsqcup X$ respectively, with $u \sqcup v$ being written rather than $\bigsqcup\{u, v\}$, etc. In the simplest case we can take $W = \{0, 1\}$ with $0, 1$ standing for true and false respectively. In this case the fuzzy bases just reduce to (characteristic functions of) crisp belief bases and we can write $\varphi \in u$ rather than $u(\varphi) = 1$, while $\sqsubseteq, \bigsqcup, \bigsqcap$ reduce to the usual $\subseteq, \bigcup, \bigcap$.

The set $\mathbf{Con} \subseteq \mathcal{F}(L_0)$ is required to satisfy two conditions. First, it is assumed to be *downwards closed* in the lattice $\mathcal{F}(L_0)$:

If $v \in \mathbf{Con}$ and $u \sqsubseteq v$ then $u \in \mathbf{Con}$.

The second condition is slightly more involved, and corresponds to a type of compactness condition:

Definition 1 $\mathbf{Con}$ *is* logically compact *iff* $\bigsqcup X \in \mathbf{Con}$ *for any* $X \subseteq \mathbf{Con}$ *such that* $u, v \in X$ *implies there exists* $w \in X$ *such that* $u \sqcup v \sqsubseteq w$.

[1] Actually for *(iii)* they start off assuming a *deduction* operator D which for each fuzzy base returns a new fuzzy base denoting its fuzzy *consequences*. However, as they point out, only the plain notion of consistency is required for their formal results. (See Section 7 of (Booth & Richter 2005).)

In other words, the supremum of every directed family of consistent fuzzy bases is itself consistent.

Given all this, we can make the following definitions, assuming some fixed $u \in \mathcal{F}(L_0)$:

Definition 2 *$u\perp$ is defined to be the set of maximally consistent fuzzy subsets of u, i.e., $v \in u\perp$ iff*
(i). $v \sqsubseteq u$.
(ii). $v \in \mathbf{Con}$.
(iii). *If $v \sqsubset w \sqsubseteq u$ then $w \notin \mathbf{Con}$.*
A selection function (for u) *is a function γ such that $\emptyset \neq \gamma(u\perp) \subseteq u\perp$.*

From a selection function γ we define a *consolidation* operator $!_\gamma$ for u by setting

$$u!_\gamma = \bigsqcap \gamma(u\perp).$$

Definition 3 *$!$ is a* partial meet fuzzy base consolidation *operator (for u) if $! = !_\gamma$ for some selection function γ for u.*

Partial meet fuzzy base consolidation can be thought of as a special case of a more general operation of partial meet fuzzy base *revision*. In fact consolidation amounts to a revision by a "vacuous" revision input $(\varphi/0_W)$ representing the new information that the degree of φ is at least 0_W, where 0_W is the minimal element of the lattice W. This more general operation was studied and axiomatically characterized in (Booth & Richter 2005). The following characterization of partial meet fuzzy consolidation does not appear in (Booth & Richter 2005), though it can be proved using similar methods.

Theorem 1 *$!$ is a partial meet fuzzy consolidation operator iff $!$ satisfies the following three conditions:*

1. $u! \in \mathbf{Con}$.
2. $u! \sqsubseteq u$
3. *For all $\phi \in L_0$, $b \in W$, if $b \not\leq_W u!(\phi)$ and $b \leq_W u(\phi)$ then there exists $u' \in \mathbf{Con}$ such that $u! \sqsubseteq u' \sqsubseteq u$ and $u' \sqcup (\phi/b) \notin \mathbf{Con}$.*

In 3, $u!(\phi)$ is the degree assigned to ϕ by the fuzzy base $u!$, while (ϕ/b) denotes that fuzzy base which assigns b to ϕ and 0_W to every other formula.

Application

Given an arbitrary set $R \subseteq L^2$ (possibly infinite) of rules, we need to formally define when R is consistent. For now, we use the earlier-defined notion of strong consistency for out_1, which we will refer to as consistent$_1$.

Definition 4 *Let $R \subseteq L^2$ be a set of rules. We say R is* consistent$_1$ *iff $out_1(R, \phi)$ is Cn-consistent for all Cn-consistent $\phi \in L$.*

Using results of (Makinson & van der Torre 2000), we get an alternative characterization of consistent$_1$:

Proposition 1 *The following are equivalent:*
(i). *R is consistent$_1$.*
(ii). *For all Cn-consistent $\phi \in L$, $\phi \to \perp$* cannot *be derived from R using the rule-set* Rules$_1$ *that contains SI: derive $(\phi \wedge \xi) \to \psi$ from $\phi \to \psi$, WO: derive $\phi \to (\psi \vee \varphi)$ from $\phi \to \psi$, AND: derive $\phi \to (\psi \wedge \varphi)$ from $\phi \to \psi$ and $\phi \to \varphi$.*

To help define merging operators for rules, our aim now is to define an operation which takes an arbitrary set of rules R and returns a new rule set $R!$ which is consistent$_1$. We set up the following definitions:

Definition 5 *$R\perp$ is defined to be the set of maximally consistent$_1$ subsets of R, i.e., $X \in R\perp$ iff*
(i). *$X \subseteq R$.*
(ii). *X is consistent$_1$.*
(iii). *If $X \subset Y \subseteq R$ then Y is inconsistent$_1$.*
A selection function (for R) *is a function γ such that $\emptyset \neq \gamma(R\perp) \subseteq R\perp$.*

From a given selection function γ we then define a consolidation operator for R by setting

$$R!_\gamma = \bigcap \gamma(R\perp).$$

Definition 6 *$!$ is a* partial meet consolidation *operator (for R) if $! = !_\gamma$ for some selection function γ for R.*

What are the properties of this family of consolidation operators? It turns out the following is a sound and complete set of properties for partial meet consolidation.

1. $R!$ is consistent$_1$.
2. $R! \subseteq R$.
3. If $\phi \to \psi \in R \setminus R!$ then there exists X such that $R! \subseteq X \subseteq R$, X is consistent$_1$, and $X \cup \{\phi \to \psi\}$ is inconsistent$_1$.

Theorem 2 *$!$ is a partial meet consolidation operator iff $!$ satisfies 1–3 above.*

Now, by considering the special case $L_0 = L^2, W = \{0, 1\}$, and $\mathbf{Con} =$ consistent$_1$ we obtain Theorem 2 as just an instance of Theorem 1. However, to be able to do this we need to check that consistent$_1$ satisfies the conditions required of it:

Theorem 3 *consistent$_1$ is downwards closed and logically compact.*

Proof: The easiest way to show these is by considering the proof-theoretical characterization of consistent$_1$ from Proposition 1*(ii)*.

To show consistent$_1$ is downwards closed in this case means to show that if R is consistent$_1$ and $R' \subseteq R$ then R' is consistent$_1$. But if R' is inconsistent$_1$ then $\phi \to \perp$ is derivable from R' using *Rules*$_1$, for some Cn-consistent ϕ. If $R' \subseteq R$ then obviously any derivation from R' is a derivation from R. Hence this implies R is inconsistent$_1$.

To show consistent$_1$ is logically compact means to show that $\bigcup X$ is consistent$_1$ for any set X of consistent$_1$ rule bases such that $R, R' \in X$ implies there exists $R'' \in X$ such that $R \cup R' \subseteq R''$. But suppose for contradiction consistent$_1$ was *not* logically compact. Then there is some set X of consistent$_1$ rule bases satisfying the above condition and such that $\bigcup X$ is inconsistent$_1$. This means for some Cn-consistent ϕ there is a derivation of $\phi \to \perp$ from $\bigcup X$ using *Rules*$_1$. Let $A_1, \ldots, A_n$ be the elements of $\bigcup X$ used in this derivation, and let $R_1, \ldots, R_n$ be rule bases in X such that $A_i \in R_i$. By repeated application of the above

condition on X we know there exists $R'' \in X$ such that $R_1 \cup \ldots \cup R_n \subseteq R''$. Hence our derivation of $\phi \rightarrow \bot$ is also a derivation from R''. Thus we have found an element of X (namely R'') which is inconsistent$_1$ – contradicting the assumption that X contains only consistent$_1$ rule bases. Thus consistent$_1$ is logically compact. ■

Remark
The proof above clearly goes through independently of the actual rules which belong to *Rules*$_1$. We could just as easily substitute *Rules*$_2$ = *Rules*$_1 \cup \{$OR$\}$: derive $\phi \vee \xi \rightarrow \psi$ from $\phi \rightarrow \psi$ and $\xi \rightarrow \psi$, or *Rules*$_3$ = *Rules*$_1 \cup \{$CT$\}$: derive $\phi \rightarrow \varphi$ from $\phi \rightarrow \psi$ and $\phi \wedge \psi \rightarrow \varphi$, or *Rules*$_4$ = *Rules*$_1 \cup \{$OR, CT$\}$. This means Theorem 3 also holds if we replace consistent$_1$ with consistent$_i$ for any $i \in \{1, 2, 3, 4\}$, where we define R is consistent$_i$ iff $out_i(R, \phi)$ is Cn-consistent for all Cn-consistent $\phi \in L$. This follows from results in (Makinson & van der Torre 2000) which state $\psi \in out_i(R, \phi)$ iff $\phi \rightarrow \psi$ is derivable from R using *Rules*$_i$.

Rule implication

Merging operators may merge $\phi \wedge \psi$ and $\neg\phi$ into ψ, which illustrates that merging operators not necessarily select a subset of the formulas from the conflicting sources, like the consolidation operators discussed in the previous section, but they may also contain a formula implied by one of the formulas of the sources. When we want to adapt such an operator for rule merging, we have to define not only the consistency of a set of rules, but also when rules imply other rules.

There are many notions of rule implication in the literature. For example, consider the material implication in propositional logic, thus $\phi \rightarrow \psi = \phi \supset \psi$. We have for example that $\phi \supset \psi$ implies $(\phi \wedge \xi) \supset \psi$ and $\phi \supset (\psi \vee \xi)$, or that $\phi \supset \psi$ together with $\psi \supset \xi$ implies $\phi \supset \xi$. Such properties have been studied more systematically traditionally in conditional logic, or more recently in input/output logic (Makinson & van der Torre 2000; Bochman 2005). But these are just examples, and do not directly provide a general solution. In particular, it does not solve the question how to use merging operators for rules defined in a meta-language, in which case we only have an operation $out(R, S)$ defining how to apply a set of rules.

For the general case we propose the following definition of implication among rules. Following the convention in input/output logic, we overload the operator 'out' to refer to this operation too (the two kinds of operations are distinguished by the number of their arguments).

$$out(R) = \{\phi \rightarrow \psi \mid \phi = \wedge S, \psi \in out(R, S)\}.$$

Example: merging in possibilistic logic

Prioritized information is represented in possibilistic logic (Dubois, Lang, & Prade 1994) by means of a set of weighted formulas of the form $B = \{(\phi_i, a_i) : i = 1, \cdots, n\}$. The pair (ϕ_i, a_i) means that the certainty (or priority) degree of ϕ_i is at least a_i which belongs to the unit interval $[0, 1]$. A possibility distribution is associated to a possibilistic base as follows: $\forall\omega \in \Omega$,

$$\pi_B(\omega) = \begin{cases} 1 & \text{if } \forall(\phi_i, a_i) \in B, \omega \models \phi_i \\ 1 - max\{a_i : (\phi_i, a_i) \in B \text{ and } \omega \not\models \phi_i\} & \\ & \text{otherwise.} \end{cases}$$

When $\pi(\omega) > \pi(\omega')$ we say that ω is preferred to (or more satisfactory than) ω'. For the rest of this section we simplify by assuming our language L is generated by only *finitely* many propositional variables.

A possibilistic base $B = \{(\phi_i, a_i) : i = 1, \cdots, n\}$ is consistent iff the set of propositional formulas $\{\phi_i : (\phi_i, a_i) \in B\}$ associated with B is Cn-consistent.

$\oplus : [0, 1]^k \rightarrow [0, 1]$ is a k merging operator when it satisfies the following two conditions. The first condition says that if an alternative is fully satisfactory for all agents then it will be also fully satisfactory w.r.t. the result of merging. The second condition is the monotonicity property.

(i). $\oplus(1, \cdots, 1) = 1$
(ii). $\oplus(a_1, \cdots, a_n) \geq \oplus(b_1, \cdots, b_n)$ if $a_i \geq b_i$ for all $i = 1, \cdots, n$.

For example, let $B_1 = \{(\phi_i, a_i) : i = 1, \cdots, n\}$ and $B_2 = \{(\psi_j, b_j) : j = 1, \cdots, m\}$ be two possibilistic bases. Using $\oplus$, the result of merging B_1 and B_2, written as $\mathcal{B}_\oplus$, is defined as follows (Benferhat *et al.* 1999):

$$\begin{aligned} \mathcal{B}_\oplus = \quad & \{(\phi_i, 1 - \oplus(1 - a_i, 1)) : (\phi_i, a_i) \in B_1\} \\ & \cup \{(\psi_j, 1 - \oplus(1, 1 - b_j)) : (\psi_j, b_j) \in B_2\} \\ & \cup \{(\phi_i \vee \psi_j, 1 - \oplus(1 - a_i, 1 - b_j))\}. \end{aligned} \quad (1)$$

We suppose that the bases (possibilistic bases in this section and rule bases in the following sections) are *individually consistent*. Inconsistency occurs after their merging.

Given $\mathcal{B}_\oplus$, the *useful* result of merging – from which inferences are drawn – is defined as a subset of $\mathcal{B}_\oplus$ composed of the consistent most prioritized formulas of $\mathcal{B}_\oplus$, as far as possible. Formally we have:

Definition 7 (Useful result of merging) *Let $\mathcal{B}_\oplus$ be the result of merging $B_1, \cdots, B_n$ using $\oplus$. Let $\mathcal{B}_{\oplus_{\geq a}} = \{\phi_i : (\phi_i, a_i) \in \mathcal{B}_\oplus, a_i \geq a\}$ and $Inc(\mathcal{B}_\oplus) = max\{a_i : (\phi_i, a_i) \in \mathcal{B}_\oplus, \mathcal{B}_{\oplus_{\geq a_i}}$ is Cn-inconsistent$\}$. The useful part of $\mathcal{B}_\oplus$ is:*

$$\rho(\mathcal{B}_\oplus) = \{(\phi_i, a_i) : (\phi_i, a_i) \in \mathcal{B}_\oplus, a_i > Inc(\mathcal{B}_\oplus)\}.$$

Another more qualitative representation of a possibilistic base has been studied in possibilistic logic, based on a well ordered partition of formulas (so without explicit weights!) $\mathcal{B} = B_1; \cdots; B_n$, where formulas of B_i are prioritized over formulas of B_j for $i < j$.

Let $\mathcal{B} = B_1; \cdots; B_n$ and $\mathcal{B}' = B'_1; \cdots; B'_m$. The useful result of merging $\mathcal{B}$ and $\mathcal{B}'$ using the $\min$ operator, written as $\mathcal{B}_{\min}$, is $B_{\min,1}; \cdots; B_{\min,max(n,m)}$, where $B_{\min,i} = (B_i \cup B'_i)$ if $\bigcup_{1 \leq j \leq i}(B_j \cup B'_j)$ is Cn-consistent, empty otherwise.

The useful result of merging $\mathcal{B}$ and $\mathcal{B}'$ using the $\max$ operator is $B_{\max,1}; \cdots; B_{\max,max(n,m)}$, where $B_{\max,i} =$

$(\cup_{\phi\in B_i,\psi\in B'_1\cup\cdots\cup B'_i}(\phi\vee\psi))\cup(\cup_{\phi\in B_1\cup\cdots\cup B_i,\psi\in B'_i}(\phi\vee\psi))$ with B_i (resp. B'_i) is composed of tautology when $i > n$ (resp. $i > m$).

A possibility distribution π can also be written under a well ordered partition, of the set of all possible worlds Ω, of the form $(E_1, \cdots, E_n)$ such that

- $E_1 \cup \cdots \cup E_n = \Omega$,
- $E_i \cap E_j = \emptyset$ for $i \neq j$,
- $\forall \omega, \omega' \in \Omega$, $\omega \in E_i, \omega' \in E_j$ with $i < j$ iff $\pi(\omega) > \pi(\omega')$.

Rules in possibilistic logic

The qualitative representation of a possibilistic base is a particular kind of rules, using Algorithm 1 to compute the possibility distribution associated with a set of rules (Pearl 1990; Benferhat, Dubois, & Prade 1992). Let $R = \{\phi_i \to \psi_i : i = 1, \cdots, n\}$, and let $\mathcal{C} = \{(L(C_i), R(C_i)) : L(C_i) = Mod(\phi_i \wedge \psi_i), R(C_i) = Mod(\phi_i \wedge \neg\psi_i), \phi_i \to \psi_i \in R\}$, where $Mod(\xi)$ is the set of worlds satisfying ξ.

Algorithm 1: Possibility distribution associated with a rule base.

begin
  $l \leftarrow 0$;
  while $\Omega \neq \emptyset$ **do**
    – $l \leftarrow l + 1$;
    – $E_l = \{\omega : \forall (L(C_i), R(C_i)) \in \mathcal{C}, \omega \notin R(C_i)\}$;
    if $E_l = \emptyset$ **then**
      Stop (inconsistent rules);
      $E_l = \Omega$;
    – $\Omega = \Omega - E_l$;
    – $\mathcal{C} = \mathcal{C} \backslash \{(L(C_i), R(C_i)) : L(C_i) \cap E_l \neq \emptyset\}$;
  return $(E_1, \cdots, E_l)$
end

Algorithms have been defined to translate one representation into another. For example, Algorithm 2 translates a set of rules into a possibilitic base given in a well ordered partition (Benferhat, Dubois, & Prade 2001).

Algorithm 2: Translating R into B.

begin
  $m \leftarrow 0$;
  while $R \neq \emptyset$ **do**
    – $m \leftarrow m + 1$;
    – $A = \{\phi_k \supset \psi_k : \phi_k \to \psi_k \in R\}$;
    – $H_m = \{\phi_k \supset \psi_k : \phi_k \to \psi_k \in R, A \cup \{\phi_k\}$ is Cn-consistent$\}$;
    – **if** $H_m = \emptyset$ **then** Stop (inconsistent rules);
    – $R = R \backslash \{\phi_k \to \psi_k : \phi_k \supset \psi_k \in H_m\}$;
  return $\Sigma = \Sigma_1; \cdots; \Sigma_n$ *s.t.* $\Sigma_i = H_{m-i+1}$.
end

Let R be a set of rules and $\Sigma = \Sigma_1; \cdots; \Sigma_n$ be its associated possibilistic base using Algorithm 2. We define a stratification of R as $R = R_1; \cdots; R_n$ with $R_i = \{\phi_k \to \psi_k : \phi_k \supset \psi_k \in \Sigma_i\}$. This stratification of R will be used in the next section.

Moreover, the associated rule base of $\Sigma = \Sigma_1; \cdots; \Sigma_n$ is (Benferhat *et al.* 2001):

$$R = \{\top \to \Sigma_n,$$
$$\neg\Sigma_{n-1} \vee \neg\Sigma_n \to \Sigma_{n-1},$$
$$\cdots,$$
$$\neg\Sigma_1 \vee \neg\Sigma_2 \to \Sigma_1\},$$

where $\Sigma_i = \bigwedge_{\phi\in\Sigma_i} \phi$.

Merging rules in possibilistic logic

For the particular kind of rules defined in possibilistic logic, we can thus define a merging operator as follows. Given a set of rules, transform the set of rules to a possibilistic base. Then apply a merging operator from possibilistic logic. Finally, transform the useful part of the merged base back into a set of rules.

Definition 8 *Let $\mathcal{R}$ and $\mathcal{R}'$ be two rule bases. The result of merging $\mathcal{R}$ and $\mathcal{R}'$ using the* min *operator, written as $R_{\min}$, is obtained by translating R and R' to B and B' using Algorithm 2, merging B and B' according to the* min *operator, and translating the useful result of merging back into a set of rules. The result of merging $\mathcal{R}$ and $\mathcal{R}'$ using the* max *operator is defined analogously.*

Instead of this indirect way, we also define the merger directly. We consider again the min and max mergers.

Definition 9 *Let R and R' be two rule bases, and let $R_1; \ldots; R_n$ and $R'_1; \ldots; R'_m$ be their stratifications according to Algorithm 2. Let $R[k]$ be the set of rules in the first k equivalence classes, $\bigcup_{i=1\ldots k}(R_i \cup R'_i)$. The merger of R and R' according to* min*, written as $R_{\min}$, is $R[k]$ such that $\{\phi_l \supset \psi_l : \phi_l \to \psi_l \in R[k]\}$ is Cn-consistent, and k is maximal.*

Definition 10 *Let R and R' be two rule bases. The merger of R and R' according to* max*, written as $R_{\max}$, is $\{(\phi \wedge \xi) \to (\psi \vee \varphi) \mid \phi \to \psi \in R, \xi \to \varphi \in R'\}$.*

The direct merging approach is twofold interest. It avoids the different translations and also provides more intuitive results at the syntactic level.

Example 1 *Assume there is only a single rule $\phi \to \psi$ which is merged with the empty base. The indirect approach leads to $\{\top \to (\phi \supset \psi)\}$ and the direct approach leads to $\{\phi \to \psi\}$. The two sets are equivalent in the sense that they lead to the same possibility distribution using Algorithm 1.*

The indirect and direct approaches are in this sense equivalent.

Proposition 2 *Let $\mathcal{R}$ and $\mathcal{R}'$ be two rule bases. Let R_1 (resp. R_2) be the result of merging $\mathcal{R}$ and $\mathcal{R}'$ using the* min *operator following Definition 8 (resp. Definition 9). Then R_1 and R_2 are equivalent in the sense that they induce the same possibility distribution.*
This result holds for the max *operator as well.*

The latter example illustrates that the kind of rules studied in possibilistic logic are of a particular kind, and it raises the question how the merging operation can be generalized for arbitrary rules. This is studied in the following section.

Application

We now consider the generalization of this approach for an arbitrary notion of rules. We first introduce the following generalization of the stratification Algorithm 2.

Algorithm 3: Stratification of a rule base R.

begin
  $m \leftarrow 0$;
  while $R \neq \emptyset$ **do**
    – $m \leftarrow m + 1$;
    – $H_m = \{\phi_k \rightarrow \psi_k : \phi_k \rightarrow \psi_k \in R, out(R, \phi_k) \cup \{\phi_k\}$ is Cn-consistent$\}$;
    – **if** $H_m = \emptyset$ **then** Stop (inconsistent base);
    – remove H_m from R;
  return $R = R_1; \cdots ; R_n$ *s.t.* $R_i = H_{m-i+1}$.
end

That Algorithm 3 is a generalization of Algorithm 2 can be seen by setting $out(R) = \{\phi \supset \psi \mid \phi \rightarrow \psi \in R\}$ in the above.

The following example illustrates two distinct kinds of examples of rule sets.

Example 2 *Consider the following two:*

1. $\top \rightarrow f, d \rightarrow \neg f$
2. $\top \rightarrow \neg f, f \rightarrow w$

Both examples will be stratified in two equivalence classes using the algorithm above, but for completely different reasons. In the first example, "d" causes an inconsistency, and in the second example, "f" is an inconsistency. (the first is the usual kind of specificity in the Tweety example, the second is the contrary-to-duty studied in deontic logic; the first is an exception, the second a violation). The first base is stratified into $\{d \rightarrow \neg f\}; \{\top \rightarrow f\}$ and the second base is stratified into $\{f \rightarrow w\}; \{\top \rightarrow \neg f\}$.

Proposition 3 *A set of rules is inconsistent, according to Algorithm 3, when for all rules $(\phi, \psi) \in R$, we have that $out(R, \{\phi\})$ is Cn-inconsistent.*

We can use the proposition to define a merging operator according to the min operator, which is again a selection operator. We therefore can use the same definition as above; clearly it is again a generalization.

Definition 11 *Let R and R' be two rule bases, and let $R_1; \ldots ; R_n$ and $R'_1; \ldots ; R'_m$ be their stratifications according to Algo. 3. Let $R[k]$ be the set of rules in the first k equivalence classes, $\bigcup_{i=1\ldots k}(R_i \cup R'_i)$. The merger of R and R' according to* $\min$*, written as $R_{\min}$, is $R[k]$ such that $R[k]$ is consistent according to Algo. 3, and k is maximal.*

For the merging operator based on max, we have to use the notion of implication in out. We simply use the same operator as above.

Definition 12 *Let R and R' be two rule bases. The merger of R and R' according to* $\max$*, written as $R_{\max}$, is $\{(\phi \wedge \xi) \rightarrow (\psi \vee \varphi) \mid \phi \rightarrow \psi \in R, \xi \rightarrow \varphi \in R'\}$.*

Variations

The product operator in possibilistic logic may be seen as a combination of the min and the max operator, in the sense that the merger contains both selections and disjunction. We conjecture that it can be defined analogously in our generalized setting. There are other ways to generalize Algorithm 2 for an arbitrary notion of rules. If we write $R[\phi] = \{(\xi \rightarrow \psi) \in R | (\phi \leftrightarrow \xi) \in Cn(\emptyset)\}$, we can find alternatives for the relative line of the algorithm, for example:

– $H_m = \{\phi_k \rightarrow \psi_k : \phi_k \rightarrow \psi_k \in R, out(R, \phi_k)\}$ is Cn-consistent$\}$;
– $H_m = \{\phi_k \rightarrow \psi_k : \phi_k \rightarrow \psi_k \in R, out(R \setminus R[\phi_k], \{\phi_k\})$ is Cn-consistent$\}$;

Summary

In this paper we introduce a general framework to study rule merging and develop rule merging operators as a generalization of belief merging operators. We use simple rules defined as pairs of formulas of a base logic, i.e., as conditionals. We distinguish weak consistency of rules only in contexts of the given rules, and strong consistency for all possible consistent contexts. We define a notion of implication among rules based on implication in the base language:

$$out(R) = \{\phi \rightarrow \psi \mid \phi = \wedge S, \psi \in out(R, S)\}.$$

We use the framework to study two examples.

Booth and Richter introduce a merging operator based on a notion of consistency. Using our strong notion of consistency of a set of rules, we define a rule merging operator. For the proof of completeness we use a proof-theoretical characterization. This illustrates a general point: to use belief merging operators for rule merging, we may need to prove some additional properties of the rule system, such as a notion of compactness.

In possibilistic logic, a framework has been proposed to study a variety of merging operators. Since also a kind of rules have been studied in the framework of possibilistic logic, these merging operators can also be used for this particular kind of rules. When generalizing the operators for other kinds of rules, several new issues arise.

Since we considered only two examples of generalizing belief merging operators to rule merging, there are many possible studies for further research. We expect that a study of such examples will lead to a further refinement of our general framework.

Acknowledgements

Thanks are due to the reviewers for some helpful comments.

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic* 50:510–530.

Benferhat, S.; Dubois, D.; Prade, H.; and Williams, M. 1999. A practical approach to fusing and revising prioritized belief bases. In *Proc. of EPIA 99*, 222–236.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2001. Bridging logical, comparative and graphical possibilistic representation frameworks. In *Conf. on Symbolic and Quantitative Approaches to Reas. and Uncert.*, 422–431.

Benferhat, S.; Dubois, D.; and Prade, H. 1992. Representing default rules in possibilistic logic. In *Int. Conf. of Principles of Knowledge Rep. and Reas. (KR'92)*, 673–684.

Benferhat, S.; Dubois, D.; and Prade, H. 2001. Towards a possibilistic logic handling of preferences. *Applied Intelligence* 14(3):303–317.

Bochman, A. 2005. *Explanatory non-monotonic reasoning*. World scientific.

Booth, R., and Richter, E. 2005. On revising fuzzy belief bases. *Studia Logica* 80(1):29–61.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic. *In Handbook of Logic in Artificial Intelligence and Logic Programming* pages 439–513.

Gerla, G. 2001. *Fuzzy Logic: Mathematical Tools for Approximate Reasoning*. Kluwer Academic Publishers.

Hansson, S. O. 1999. *A Textbook of Belief Dynamics*. Kluwer Academic Publishers.

Makinson, D., and van der Torre, L. 2000. Input-output logics. *Journal of Philosophical Logic* 29:383–408.

Makinson, D., and van der Torre, L. 2001. Constraints for input-output logics. *Journal of Philosophical Logic* 30(2):155–185.

Pearl, J. 1990. System Z: A natural ordering of defaults with tractable applications to default reasoning. In Eds, R. P., ed., *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'90)*, 121–135. Morgan Kaufmann.

6.5 A reversible framework for propositional bases merging

A reversible framework for propositional bases merging

Julien Seinturier and **Odile Papini**
LSIS UMR CNRS 6168 - Equipe INCA - Université de Toulon et du Var
Avenue de l'Université - BP20132, 83957 LA GARDE CEDEX - FRANCE
{papini, seinturier}@univ-tln.fr

Pierre Drap
MAP UMR CNRS 694
Ecole D'architecture de Marseille Pierre.Drap@gamsau.archi.fr

Keywords

Knowledge representation, Knowledge composition, Decision.

Abstract

The problem of merging multiple sources information is central in several domains of computer science. In knowledge representation for artificial intelligence, several approaches have been proposed for merging propositional bases. However none of these approaches allows us the reversibility of the merging process. In this paper, we propose a very general reversible framework for merging ordered as well as not ordered pieces of information coming from various sources either ordered or not. A semantic approach of merging in the proposed reversible framework is first presented, stemming from a representation of total pre-orders by means of polynomials on real numbers. The syntactic counter-part is then presented, based on belief bases weighted by polynomials on real numbers. We show the equivalence between semantic and syntactic approaches. Finally, we show how this reversible framework is suitable for easily representing the approach of merging propositional bases stemming from Hamming distance and how the proposed framework is suitable for generalizing the revision of an epistemic state by an epistemic state to the fusion of several epistemic states.

Introduction

Merging information coming from different sources is an important issue in various domains of computer science like knowledge representation for artificial intelligence, decision making or databases. The aim of fusion is to obtain a global point of view, exploiting the complementarity between sources, solving different existing conflicts, reducing the possible redundancies. Among the various approaches of multiple sources information merging, logical approaches gave rise to increasing interest in the last decade (Baral *et al.* 1992; Revesz 1993; Lin 1996; Revesz 1997; Cholvy 1998). Most of these approaches have been defined within the framework of classical logic, more often propositional, and have been semantically defined. Different postulates characterizing the rational behaviour of fusion operators have been proposed (Konieczny & Pérez 1998) and various operators have been defined according to whether explicit or implicit priorities are available (Konieczny & Pérez 1998), (Laffage & Lang 2000). More recently, new approaches have been proposed like semantic merging for propositional bases stemming from the Hamming distance (Konieczny, Lang, & Marquis 2002) or syntactic fusion in a possibilistic framework (Dubois, Lang, & Prade 1994; Benferhat *et al.* 2002a) which is a real advantage at a computational point of view. However these frameworks do not allow for the reversibility of the fusion operations. On a theoretical point of view, reversibility is interesting because it involves the definition of a new framework that enables to express priorities independantly from the merging operators. When facing real scale applications, large amount of data are produced by numerous users. Robust merging techniques and error recovering techniques are necessary. Data management applications require the reversibility of the merging process in case of errors. In archaeological applications, various kinds of errors linked to the measure process may occur. Besides, several surveys of a same object, made at two different instants by a same person or by two different persons may lead to inconsistencies. Indeed, the result of the fusion in a first survey is performed from measures and hypothesis on the object stemming from archeologists' expert knowledge. In the following surveys, new measures may conflict with the hypothesis of the previous survey. Excavations generally take place during several years, surveys made at a certain year may produce knowledge that may invalidate the hypothesis made years before, therefore there is a necessity to come back to initial information. We propose a very general reversible framework for fusion. This framework is suitable for both ordered or not ordered sources as well as for items of information with explicit or implicit priorities or without priorities. Information is represented in propositional calculus and the fusion operation are semantically and syntactically defined. The reversibility of the fusion operations is obtained by an appropriate encoding of the pre-orders on interpretations and on formulas by polynomials on real numbers (Papini 2001; Benferhat *et al.* 2002b).

Preliminaries and notations

In this paper we use propositional calculus, denoted by $\mathcal{L}_{\mathcal{PC}}$, as knowledge representation language with the usual connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. The lower case letters a, b, c, $\cdots$, are used to denote propositional variables, the lower case

Greek letters ϕ, ψ, $\cdots$, are used to denote formulas, the upper case letters A, B, C, are used to denote sets of formulas. We denote by $\mathcal{W}$ the set of interpretations[1] and by $Mod(\psi)$ the set of models of ψ, that is $Mod(\psi) = \{\omega \in \mathcal{W}, \omega \models \psi\}$ where $\models$ denotes the inference relation used for drawing conclusions. Let ψ and ϕ be formulas and X be a set of formulas, $\psi \models \phi$ denotes that $Mod(\psi) \subseteq Mod(\phi)$ and $X \models \phi$ denotes that $\forall \psi \in X$, $Mod(\psi) \subseteq Mod(\phi)$. The symbol $\equiv$ denotes logical equivalence, and $\psi \equiv \phi$ means that $\psi \models \phi$ and $\phi \models \psi$.

Pre-orders and polynomials

The aim of this section is to briefly recall some definitions on polynomials and to remind how polynomials can be used to represent total pre-orders as well as changes on total pre-orders(Papini 2001)(Benferhat *et al.* 2002b).

Polynomials and pre-order on polynomials Let $\mathbb{R}$ be the set of real numbers. We denote by $\mathbb{R}[x]$ the set of polynomials such that $p = \sum_{i=0}^{n} p_i\, x^i$, $p_i \in \mathbb{R}$. We call right (resp. left) shift of k positions a multiplication (resp. division) by x^k. The support of a polynomial p is the set of elements of $\mathbb{N}$, denoted by S_p, composed by the indices i for which $p_i \neq 0$. Moreover, $max(S_p) = deg(p)$, and $max(\emptyset) = 0$, $deg(p)$ denotes the degree of p.

Pre-orders on polynomials Let p and q be two polynomials on real numbers such that $p = \Sigma_{i=0}^{k}\ p_i\ x^i$ and $q = \Sigma_{i=0}^{l}\ q_i\ x^i$. We use various pre-orders for comparing polynomials.

Maximum The pre-order $\leq_{MAX}$ is:
$p \leq_{MAX} q$ iff $max_{i=0}^{k}(p_i) \leq max_{i=0}^{l}(q_i)$.
Where $max_{i=0}^{l}(q_i)$ denotes the maximum of the set $\{q_0, \cdots, q_l\}$

Sum The pre-order $\leq_{SUM}$ is:
$p \leq_{SUM} q$ iff $\Sigma_{i=0}^{k} p_i \leq \Sigma_{i=0}^{l} q_i$.

Weighted sum Let $\{a_i,\ 1 \leq i \leq k\}$ and $\{b_j,\ 1 \leq j \leq l\}$ two sets of scalars. The pre-order $\leq_{WS}$ is:
$p \leq_{WS} q$ iff $\Sigma_{i=0}^{k} a_i \times p_i \leq \Sigma_{j=0}^{l} b_j \times q_j$.

Lexicographic The pre-order $\leq_{LEX}$ is:
$p \leq_{LEX} q$ iff $\exists i \in \mathbb{N}\ \forall j \in \mathbb{N},\ j < i,\ (p_j = q_j$ and $p_i < q_i)$.

Leximax Let v and w be two vectors composed by the coefficients of p and q ordered in increasing order. Let $p' = \Sigma_{i=0}^{n} v_i\ x^i$ and $q' = \Sigma_{j=0}^{m} w_j\ x^j$ be two polynomials built with the components of v and w respectively. The pre-order $\leq_{LMAX}$ is such that $p \leq_{LMAX} q$ iff $p' \leq_{LEX} q'$.

[1] Interpretations are represented by set of literals.

Representing pre-orders by polynomials Let $(A,\ \leq_A)$ be a finite set with a total pre-order. Representing $\leq_A$ by polynomials requires the definition of a weighting function that assigns each element a_i of A a polynomial. This weighting function is such that $rk(a_i) \in \mathbb{N}$ is the rank of a_i in the pre-order $\leq_A$ [2]. From the binary decomposition of $rk(a_i)$, denoted by $(v_0, \ldots, v_m)$, with $2^{m-1} \leq rk(a_i) < 2^m$, we build a polynomial $p(a_i)$ such that $p(a_i) = \Sigma_{i=0}^{m} v_{m-i}\ x^i$. These polynomials are ordered according to the lexicographic order to represent $\leq_A$. For details see (Papini 2001).

Semantic approach

From a semantic point of view, the priorities between interpretations are represented by polynomials as well as the result of the merging by classical fusion operators. Let $E = \{K_1,\ \ldots,\ K_n\}$ be a set of n propositional bases representing the information provided by n sources . We use two kinds of total pre-orders, a pre-order between the bases, called external pre-order and pre-orders on the interpretations of $\mathcal{L}_{PC}$ relative to each base, called internal pre-orders. In the reversible framework, external and internal pre-orders are total pre-orders represented by polynomials. In the following, preferred elements are minimal elements in total the pre-order.

External pre-order Let $E = \{K_1,\ \ldots,\ K_n\}$ be a set of propositional bases, an external weighting function is a function q that assigns each base K_i an integer called external weight and denoted by $q(K_i)$. An external pre-order denoted by $\leq_E$ is defined such that:

$$\forall K_i,\ K_j \in E,\ \ K_i \leq_E K_j \quad \text{iff} \quad q(K_i) \leq q(K_j)$$

where $q(K_i) = rk(K_i)$. When the sources are explicitly ordered, the weights $q(K_i)$ are the ranks within the total pre-order $\leq_E$. When the sources are not ordered, the bases are equally preferred and $\forall K_i \in E,\ q(K_i) = 0$.

Internal pre-order Let $K_i \in E$ be a propositional bases and $\mathcal{W}$ be the set of interpretations of $\mathcal{L}_{PC}$. An internal weighting function assigns each interpretation ω a polynomial on real numbers called internal weight and denoted by $p_{K_i}(\omega)$. For each base K_i, an internal pre-order denoted by $\leq_{K_i}$ is defined such that:

$$\forall \omega_j,\ \omega_k \in \mathcal{W},\ \ \omega_j \leq_{K_i} \omega_k \ \text{ iff } \ p_{K_i}(\omega_j) \leq p_{K_i}(\omega_k)$$

Three cases arise. When a total pre-order is given for K_i, the $p_{K_i}(\omega)$ are encoded by polynomials as mentioned in the polynomial pre-order representation section.[3]. When K_i is implicitely pre-ordered the $p_{K_i}(\omega)$ can be computed using, for example the Hamming distance (see section 6) and encoded by constant polynomials (or integers). Finally, when

[2] We call rank of a_i in the pre-order $\leq_A$ the index of a_i in the list of the elements of A ordered in ascending ordering according to $\leq_A$.

[3] For the sake of homogeneity, since pre-orders are represented by polynomials, weights are encoded by polynomials which reflect the rank of the interpretations in the total pre-order.

no pre-order is defined all the interpretations are equally preferred and we have $\forall \omega \in \mathcal{W},\ p_{K_i}(\omega) = 0$.

Global weight computation

For the semantic approach in the reversible framework, external and internal pre-orders are represented by polynomials. The merging is the combination of these pre-orders in a global pre-order. This is done by the combination of external and internal weights in a global weight independant of the merging operator.

Definition 1 *Let $q(K_i)$ be the external weight for the bases K_i, $1 \leq i \leq n$. The global external weight is such that:*

$$q_{\oplus} = \Sigma_{j=0}^{n-1}\ q(K_{j+1})x^j$$

However, the bases cannot be identified by their rank. It is necessary to define an absolute ranking in order to define an inversible function. An absolute ranking defines a one to one correspondence between ranks and bases. The absolute ranking function is only used to encode internal pre-orders in the global pre-order. The absolute ranking is not a merging priority definition.

Definition 2 *Let $E = \{K_1, \ldots, K_n\}$ be a set of propositional bases. An absolute ranking funtion, denoted by r, is an application from E to $\mathbb{N}$ which assigns each base K_i an absolute rank $r(K_i)$ such that:*

- *if $K_i <_E K_j$ then $r(K_i) < r(K_j)$.*
- *else if $K_i =_E K_j$ and $i < j$ then $r(K_i) < r(K_j)$.*

The Global weight of an interpretation is the sum of all the internal weights shifted as many times as necessary in order to produce disjoint supports. The aim of the disjoint supports is to ensure that each coefficient of an internal weight is not summed with a coefficient from another internal weight. The number of shifts depends on the absolute rank of a base. More formally, for a set $E = \{K_1, \ldots, K_n\}$ of propositional bases, with internal weights $p_{k_i}(\omega)$, $1 \leq i \leq n$ and external weights $q(K_i)$. Let r be the absolute ranking funtion for E. The global weight for an interpretation ω, denoted by $p_{K_1 \oplus \cdots \oplus K_n}(\omega)$ is such that

$$p_{K_1 \oplus \cdots \oplus K_n}(\omega) = \Sigma_{i=1}^{n}\ p_{K_i}(\omega)\ x^{\Sigma_{j=1}^{r(K_i)-1} MAX_{r^{-1}(j)}}$$

with $MAX_{r^{-1}(j)} = \max_{\omega' \in \mathcal{W}}\ deg(p_{r^{-1}(j)}(\omega')) + 1$. In the following for the sake of simplicity, we denote by $p_{\oplus}(\omega)$ the global weight.

Semantic merging within the reversible framework

In the semantic approach, merging expresses a global pre-order on the global weights. The result is the set of preferred interpretations in this pre-order. Within the reversible framework, the global pre-order, denoted by $\leq_{K_1 \oplus \cdots \oplus K_n}$ is such that $\forall \omega,\ \omega' \in \mathcal{W},\ \omega \leq_{K_1 \oplus \cdots \oplus K_n} \omega'$ iff $p_{\oplus}(\omega) \leq p_{\oplus}(\omega')$. The choice of the merging operator provides a way for comparing global weight polynomials. The use of $\leq_{MAX}$ provides the behavior of the MAX merging operator, the use of $\leq_{SUM}$ provides the behavior of the SUM merging operator, and so on. The following example illustrates the reversible framework for merging.

Example 1 *Let's use the well known example given in (Revesz 1993). Let $K_1 = \{(s \vee o) \wedge \neg d\}$, $K_2 = \{(\neg s \wedge d \wedge \neg o) \vee (\neg s \wedge \neg d \wedge o)\}$ and $K_3 = \{s \wedge o \wedge d\}$ three propositional bases. The set of interpretations $\mathcal{W}$ is sucht that $\omega_0 = \{\neg s, \neg d, \neg o\}, \omega_1 = \{\neg s, \neg d, o\}, \omega_2 = \{\neg s, d, \neg o\}, \omega_3 = \{\neg s, d, o\}, \omega_4 = \{s, \neg d, \neg o\}, \omega_5 = \{s, \neg d, o\}, \omega_6 = \{s, d, \neg o\}, \omega_7 = \{s, d, o\}$. The external pre-order is $K_3 \leq_E K_1 =_E K_2$ thus the external weights are $q(K_1) = 2$, $q(K_2) = 2$, $q(K_3) = 1$. The computation of external global weight gives $q_{\oplus} = 2 + 2x + x^2$. The absolute ranking function is defi ned by $r(K_1) = 2$, $r(K_2) = 3$, $r(K_3) = 1$ and $r^{-1}(1) = K_3$, $r^{-1}(2) = K_1$, $r^{-1}(3) = K_2$. Table 1 shows the result of the computation of internal and global weights. We have $MAX_{r^{-1}(1)} = MAX_{K_3} = max_{\omega' \in \mathcal{W}}\ deg(p_{K_3}(\omega')) + 1 = 2$, $MAX_{r^{-1}(2)} = MAX_{K_1} = 2$ and $MAX_{r^{-1}(3)} = MAX_{K_2} = 3$. For an interpretation ω_i the global weight is $p_{\oplus}(\omega_i) = p_{K_3}(\omega_i) + p_{K_1}(\omega_i)x^2 + p_{K_2}(\omega_i)x^4$. If we use the SUM merging operator, the pre-order $\leq_{SUM}$ is used and the global pre-order is $\omega_1 =_{SUM} \omega_2 =_{SUM} \omega_5 =_{SUM} \omega_7 <_{SUM} \omega_3 =_{SUM} \omega_4 =_{SUM} \omega_6 <_{SUM} \omega_0$. Minimal interpretations in this pre-order are the set $Mod(K_1 \oplus \cdots \oplus K_n) = \{\omega_1, \omega_2, \omega_5, \omega_7\}$.*

ω	p_{K_1}	p_{K_2}	p_{K_3}	$p_{\oplus}$
ω_0	x	1	$1+x$	$1+x+x^3+x^4$
ω_1	1	0	x	$x+x^2$
ω_2	x	0	x	$x+x^3$
ω_3	x	1	1	$1+x^3+x^4$
ω_4	1	x^2	x	$x+x^2+x^6$
ω_5	0	1	1	$1+x^4$
ω_6	x	1	1	$1+x^3+x^4$
ω_7	x	x^2	0	x^3+x^6

Table 1: *Interprétations, internal and global weights*

ω	$\leq_{MAX}$	$\leq_{SUM}$	$\leq_{WS}$	$\leq_{LEX}$	$\leq_{LMAX}$
ω_0	1	4	6	5	5
ω_1	1	2	3	1	1
ω_2	1	2	3	2	2
ω_3	1	3	5	4	4
ω_4	1	3	5	6	6
ω_5	1	2	3	3	3
ω_6	1	3	5	4	4
ω_7	1	2	4	7	7

Table 2: *Interprétations and rank for differents merging operators*

Reversibility

Reversibility allows us to retrieve the external pre-order as well as internals pre-orders from the global weight. Let $q_{\oplus}$ be the external weight polynomial, by the construction of the polynomial, the number of propositional bases is $n = deg(q_{\oplus}) + 1$. Moreover from the polynomial encoding of the global external weight, it is possible to retrieve all the externals weights

$$q(K_i) = \frac{q_{\oplus}\ mod\ x^i}{x^{i-1}}$$

and therefore the absolute ranks. Polynomials also allow us to retrieve the internal weights from global weights assigned to interpretations. Since the construction of global weights right shifts the internal weights in order to produce disjunct supports, the inverse operation consists in breaking the global weight into internal weights by left shifting a number of times equal to the maximum degree of the support corresponding to the greatest internal weight of the base. More formally, for each interpretation ω and for each propositional base K_i we have:

$$p_{K_i}(\omega) = \frac{p_{\oplus}(\omega) \; mod \; x^{\Sigma_{l=1}^{r(K_i)} MAX_{r^{-1}(l)}}}{x^{\Sigma_{k=1}^{r(K_i)-1} MAX_{r^{-1}(k)}}}$$

Example 2 *Using the results of example 1, we illustrate the reversibility. From the polynomial* $q_{\oplus} = 2 + 2x + 1x^2$ *we can retreive the number of merged bases that is* $deg(q_{\oplus}) + 1 = 3$. *Moreover, we can retreive the external weights since* $q(K_1) = q_{\oplus} \; mod \; x^1 = 2$, $q(K_2) = \frac{q_{\oplus} \; mod \; x^2}{x} = 2$, *and* $q(K_3) = \frac{q_{\oplus} \; mod \; x^3}{x^2} = 1$ *and thus* $K_3 \leq_E K_1 =_E K_2$. *The external weights allows us to recover the absolute ranks* $r(K_1) = 2$, $r(K_2) = 3$, $r(K_3) = 1$ *and* $r^{-1}(1) = K_3$, $r^{-1}(2) = K_1$, $r^{-1}(3) = K_2$. *We can retrieve the internal weights as follows. With* $MAX_{r^{-1}(1)} = MAX_{K_3} = 2$, $MAX_{r^{-1}(2)} = MAX_{K_1} = 2$ *and* $MAX_{r^{-1}(3)} = MAX_{K_2} = 3$. *For example, let* $p_{\oplus}(\omega_3) = 1 + x^3 + x^4$ *be the global weight for the interpretation* ω_3. *The internal weights are the followings :*

$$p_{K_1}(\omega) = \frac{p_{\oplus}(\omega) \; mod \; x^{\Sigma_{l=1}^{r(K_1)} MAX_{r^{-1}(l)}}}{x^{\Sigma_{k=1}^{r(K_1)-1} MAX_{r^{-1}(k)}}} = \frac{p_{\oplus}(\omega) \; mod \; x^{MAX_{K_3}+MAX_{K_1}}}{x^{MAX_{K_3}}} = \frac{1+x^3+x^4 \; mod \; x^4}{x^2} = x$$

then $p_{K_2}(\omega) = \frac{1+x^3+x^4 \; mod \; x^7}{x^4} = 1$ *and* $p_{K_3}(\omega) = 1 + x^3 + x^4 \; mod \; x^2 = 1$.

Syntactic approach

Let $\mathcal{B} = \{\Sigma_1, \ldots, \Sigma_n\}$ be a set of n weighted bases. Each base Σ_i is a finite set of weighted formulas such that $\Sigma_i = \{(\phi_j, p_{\Sigma_i}(\phi_j)) \mid \phi_j \in \mathcal{L}_{\mathcal{PC}}, p_{\Sigma_i}(\phi_j) \in \mathbb{R}[x]\}$. In the reversible framework, external and internal preferences are respectively represented by an external total pre-order on the weighted bases and by internal total pre-orders on the formulas. These total pre-orders are encoded by polynomials. In the syntactic approach, preferred items of the total pre-orders on the formulas are the maximum in the pre-orders.

External pre-order Let $\mathcal{B} = \{\Sigma_1, \ldots, \Sigma_n\}$ be a set of weighted bases, an external weighting function is a function that assigns each weighted base an integer denoted by $q(K_i)$. An external pre-order denoted by $\leq_{\mathcal{B}}$ is defined such that:

$$\forall \Sigma_i, \Sigma_j \in \mathcal{B}, \; \Sigma_i \leq_{\mathcal{B}} \Sigma_j \text{ iff } q(\Sigma_i) \leq q(\Sigma_j)$$

where $q(K_i) = rk(\Sigma_i)$. When the sources are explicitly ordered, the weights $q(\Sigma_i)$ are the ranks within the total pre-order $\leq_{\mathcal{B}}$. When the sources are not ordered, the bases are equally preferred and $\forall \Sigma_i \in \mathcal{B}, \; q(\Sigma_i) = 0$.

Internal pre-order Let $\Sigma_i \in \mathcal{B}$ be a weighted base. An internal weighting function for Σ_i assigns each formula ϕ of Σ_i a polynomial on real numbers denoted by $p_{\Sigma_i}(\phi)$. An internal pre-order denoted by $\leq_{\Sigma_i}$ is defined such that:

$$\forall \phi, \psi \in \Sigma_i, \; \phi \leq_{\Sigma_i} \psi \text{ iff } p_{\Sigma_i}(\phi) \leq p_{\Sigma_i}(\psi)$$

Three cases arise. When a total pre-order is given for the Σ_i, the $p_{\Sigma_i}(\phi)$ are encoded by polynomials as mentioned in polynomial pre-orders section. When Σ_i is implicitly pre-ordered the $p_{\Sigma_i}(\phi)$ can be computed as constant polynomials. Finally, when no pre-order is defined all the formulas are equally preferred and we have $\forall \phi \in \Sigma_i, \; p_{\Sigma_i}(\phi) = 0$.

Computation of the global weighted base

The merging of weighted bases is the construction of a base containing the formulas of each base, the disjunctions of two formulas coming from two bases, the disjunctions of three formulas coming from three bases, and so on until disjunctions of n formulas coming from n bases. For that, a global weight has to be computed.

Definition 3 *Let* $q(\Sigma_i)$ *be the external weight for* Σ_i. *The global external weight is such that:*

$$q_{\otimes} = \Sigma_{j=0}^{n-1} \; q(\Sigma_{j+1}) \; x^j$$

In the syntactic approach for merging, global weighted base is composed by formulas with a global weight. This weight has to take into account the external pre-order. However, the bases cannot be identified by their rank. It is necessary to define an absolute ranking function in order to define inversible function.

Definition 4 *Let* $\mathcal{B} = \{\Sigma_1, \ldots, \Sigma_n\}$ *be a set of weighted bases. An absolute ranking function, denoted by* r, *is an application from* $\mathcal{B}$ *to* $\mathbb{N}$ *which assigns each base* Σ_i *an absolute rank* $r(\Sigma_i)$ *such that:*

- *if* $\Sigma_i <_{\mathcal{B}} \Sigma_j$ *then* $r(\Sigma_i) < r(\Sigma_j)$
- *else if* $\Sigma_i =_{\mathcal{B}} \Sigma_j$ *and* $i < j$ *then* $r(\Sigma_i) < r(\Sigma_j)$

The construction of the global weighted base requires the definition of the disjunction of k formulas, denoted by D_k. The disjunction is such that:

$$D_k = \phi_{j_1} \vee \cdots \vee \phi_{j_i} \vee \cdots \vee \phi_{j_k}$$

where each ϕ_{j_i} comes from a different base Σ_i. Moreover, we denote by s the mapping which assigns each formula of D_k the weighted base from where it is coming from. More formally, let $D_k = \phi_{j_1} \vee \cdots \vee \phi_{j_i} \vee \cdots \vee \phi_{j_k}$, if $(\phi_{ji}, p_{\Sigma_l}(\phi_{ji})) \in \Sigma_l$, then $s(\phi_{ji}) = \Sigma_l$. The definition of D_k and s allows us to define a global weight.

Definition 5 *Let* $D_k = \phi_{j_1} \vee \cdots \vee \phi_{j_i} \vee \cdots \vee \phi_{j_k}$ *be a disjunction of formulas coming from* k *weighted bases. The global weight of* D_k, *denoted by* $p_{\Sigma_1 \otimes \cdots \otimes \Sigma_n}(D_k)$ *is such that:*

$$p_{\Sigma_1 \otimes \cdots \otimes \Sigma_n}(D_k) = \sum_{i=1}^{k} p_{s(\phi_{ji})}(\phi_{ji}) \times x^{\Sigma_{m=1}^{r(s(\phi_{ji}))-1} MAX_{r^{-1}(m)}}$$

with $MAX_{r^{-1}(m)} = max_{\phi' \in r^{-1}(m)}(deg(p_{r^{-1}(m)}(\phi')) + 1)$.

For the sake of simplicity, $p_\otimes(D_k)$ denotes $p_{\Sigma_1\otimes\cdots\otimes\Sigma_n}(D_k)$. The global weighted base consists in all the possible disjunctions of formulas from the bases of $\mathcal{B}$ assigned a global weight. More formally:

Definition 6 *Let $\mathcal{B} = \{\Sigma_1, \ldots, \Sigma_n\}$ be a set of weighted bases. Let D_k be a disjunction of k formulas. The global weighted base, denoted by Σ_g is such that:*

$$\Sigma_g = \bigcup_{k=1}^{n}\{(D_k, p_\otimes(D_k))\}$$

Syntactic merging and reversible framework

The result of the merging process in the syntactical approach is the set of weighted formulas of maximal global weights according to the pre-order defined as follows. Let $(\phi, p_\otimes(\phi)),\ (\psi, p_\otimes(\psi)) \in \Sigma_g$ two weighted formulas of the global stratified base. The global pre-order, denoted by $\leq_{\Sigma_1\otimes\cdots\otimes\Sigma_n}$ is such that:

$$\phi \leq_{\Sigma_1\otimes\cdots\otimes\Sigma_n} \psi \text{ iff } p_\otimes(\phi) \leq p_\otimes(\psi)$$

The choice of a merging operator involves the use of a specific pre-order on polynomials. For example, the use of the MAX operator involves the use of the $\leq_{MAX}$ pre-order. The following example illustrates the syntactic merging.

Example 3 *Let $\mathcal{B} = \{\Sigma_1, \Sigma_2, \Sigma_3\}$ be a set of weighted bases such that $\Sigma_1 = \{(\phi_1, 1)\}$, $\Sigma_2 = \{(\phi_2, x^2)\}$ and $\Sigma_3 = \{(\phi_3, 1+x)\}$. We give an arbitrary external pre-order pre-order is such that $\Sigma_3 \leq_\mathcal{B} \Sigma_1 =_\mathcal{B} \Sigma_2$, thus the external weights are $q(\Sigma_1) = 2$, $q(\Sigma_2) = 2$ and $q(\Sigma_3) = 1$. The computation of external global weight gives*

$$\begin{array}{rcl} q_\otimes & = & q(\Sigma_1)\times x^0 + q(\Sigma_2)\times x^1 + q(\Sigma_3)\times x^2 \\ & = & q(\Sigma_1) + q(\Sigma_2)x + q(\Sigma_3)x^2 \\ & = & 2 + 2x + x^2 \end{array}$$

The absolute ranking function is defi ned by $r(\Sigma_1) = 2$, $r(\Sigma_2) = 3$, $r(\Sigma_3) = 1$ and $r^{-1}(1) = \Sigma_3$, $r^{-1}(2) = \Sigma_1$, $r^{-1}(3) = \Sigma_2$. We have $MAX_{r^{-1}(1)} = MAX_{\Sigma_3} = max_{\phi'\in\Sigma_3} deg(p_{\Sigma_3}(\phi')) + 1 = 2$, $MAX_{r^{-1}(2)} = MAX_{\Sigma_1} = 1$ and $MAX_{r^{-1}(3)} = MAX_{\Sigma_2} = 3$. Moreover $s(\phi_1) = \Sigma_1$, $s(\phi_2) = \Sigma_2$ and $s(\phi_3) = \Sigma_3$. According to the defi nition, $(\phi_i, p_\otimes(\phi_i)) \in \Sigma_g$ and the global internal weights $p_\otimes(\phi_i)$ are computed as follows:

$$\begin{array}{rcl} p_\otimes(\phi_1) & = & \sum_{i=1}^{k} p_{s(\phi_1)}(\phi_1)\times x^{\sum_{m=1}^{r(s(\phi_1))-1} MAX_{r^{-1}(m)}} \\ & = & \sum_{i=1}^{k} p_{\Sigma_1}(\phi_1)\times x^{\sum_{m=1}^{r(\Sigma_1)-1} MAX_{r^{-1}(m)}} \\ & = & p_{\Sigma_1}(\phi_1)\times x^{MAX_{r^{-1}(1)}} \\ & = & p_{\Sigma_1}(\phi_1)\times x^{MAX_{\Sigma_3}} \\ & = & p_{\Sigma_1}(\phi_1)\times x^2 = x^2 \end{array}$$

Using the same way we have $p_\otimes(\phi_2) = x^5$ and $p_\otimes(\phi_3) = 1+x$. For disjunctions of more than one formula, the computation is recursive. As we can see:

$$\begin{array}{rcl} p_\otimes(\phi_1\vee\phi_3) & = & p_{\Sigma_1}(\phi_1)\times x^{MAX_{\Sigma_3}} + p_{\Sigma_3}(\phi_3)\times x^0 \\ & = & p_\otimes(\phi_1) + p_\otimes(\phi_3) \\ & = & x^2 + x + 1 \end{array}$$

The computation of all the disjunctions D_k and all the global weights gives the global weighted base:

$$\begin{array}{rcl} \Sigma_g & = & \{(\phi_1, x^2), (\phi_2, x^5), (\phi_3, 1+x), \\ & & (\phi_1\vee\phi_2, x^5+x^2), (\phi_1\vee\phi_3, x^2+x+1), \\ & & (\phi_2\vee\phi_3, x^5+x+1), \\ & & (\phi_1\vee\phi_2\vee\phi_3, x^5+x^2+x+1)\} \end{array}$$

A global pre-order characterizes the behavior of the merging operator, for example, if the operator SUM is used, the result of the merging is $\Sigma_1\otimes\Sigma_2\otimes\Sigma_3 = \{(\phi_1\vee\phi_2\vee\phi_3, x^5+x^2+x+1)\}$.

Reversibility

The reversibility allows us to retrieve the external and internal pre-orders from the global weight. Let $q_\otimes$ be the external weight polynomial, the number of propositional bases is $n = deg(q_\otimes)+1$. Moreover, it is possible to retrieve external weights:

$$q(\Sigma_i) = \frac{q_\otimes \bmod x^i}{x^{i-1}}$$

and therefore the absolute ranks. Polynomials also allow us to retrieve internal weights assigned to the formulas. Since the construction of global weight rights shift the internal weights in order to produce disjunct supports, the inverse operation consists in breaking the global weight into internal weights by left shifting a number of times equal to the maximum degree of the support corresponding to the greatest internal weight of the base. More formally, let $\psi = \phi_{j_1}\vee\cdots\vee\phi_{j_i}\vee\cdots\vee\phi_{j_n}$ be a formula coming from the disjunction of n formulas, that is a formula wich weight is composed of a maximum number of monoms, we have

$$p_{\Sigma_i}(\phi_j) = \frac{p_\otimes(\psi) \bmod x^{\sum_{l=1}^{r(s(\phi_j))} MAX_{r^{-1}(l)}}}{x^{\sum_{k=1}^{r(s(\phi_j))-1} MAX_{r^{-1}(k)}}}$$

Example 4 *Coming back to the previous example, $(\phi_1\vee\phi_2\vee\phi_3, x^5+x^2+x+1)$ is the formula coming from Σ_g with weight composed of a maximum number of monoms. The polynomial $q_\otimes = 2+2x+x^2$ allows us to know that three bases have been merged because $deg(q_\otimes)+1 = 3$. Moreover, $q(\Sigma_1) = 2$, $q(\Sigma_2) = 2$, $q(\Sigma_3) = 1$. The external weights enable to recover the function r with $r(\Sigma_1) = 2$, $r(\Sigma 2) = 3$, $r(\Sigma 3) = 1$ and therefore $r^{-1}(1) = \Sigma_3$, $r^{-1}(2) = \Sigma_1$, $r^{-1}(3) = \Sigma_2$. Moreover $s(\phi_1) = \Sigma_1$, $s(\phi_2) = \Sigma_2$ and $s(\phi_3) = \Sigma_3$ The internal weights are $p_{\Sigma_1}(\phi_1) = \frac{p_\otimes(\psi) \bmod x^{\sum_{l=1}^{r(s(\phi_1))} MAX_{r^{-1}(l)}}}{x^{\sum_{k=1}^{r(s(\phi_1))-1} MAX_{r^{-1}(k)}}} = \frac{x^5+x^2+x+1 \bmod x^3}{x^2} = 1$, $p_{\Sigma_2}(\phi_2) = \frac{x^5+x^2+x+1 \bmod x^6}{x^3} = x^2$, and $p_{\Sigma_3}(\phi_3) = \frac{x^5+x^2+x+1 \bmod x^2}{x^0} = 1+x$.*

Equivalence between semantic and syntactic approaches

We now show the equivalence between the semantic and syntactic approaches in the reversible framework. For that, like in the possibilistic framework or in the system Z (Benferhat *et al.* 2002a) (Pearl 2003) we use a function denoted by κ_{Σ_i} which for each weighted base Σ_i attaches to each interpretation ω the maximal weight of the formulas of Σ_i falsified by ω. More formally, $\forall\omega\in\mathcal{W}$, $\kappa_{\Sigma_i}(\omega) = max(\{p_{\Sigma_i}(\phi),\ (\phi, p_{\Sigma_i}(\phi)) \in \Sigma_i \text{ and } \omega \not\models \phi\})$. This function allows us to define the syntactic counterpart of a propositional base.

Definition 7 *Let $E = \{K_1, \ldots, K_n\}$ be a set of propositional bases and let $\mathcal{B} = \{\Sigma_1, \ldots, \Sigma_n\}$ be a set of*

weighted bases, Σ_i is syntactic counterpart of K_i iff $\forall \omega \in \mathcal{W}$, $\kappa_{\Sigma_i}(\omega) = p_{K_i}(\omega)$. Moreover, $\mathcal{B}$ is syntactic counterpart of E if and only if each weighted base of Σ_i de $\mathcal{B}$ is a syntactic counterpart of $K_i \in E$ and $q(K_i) = q(\Sigma_i)$ (the external pre-orders are the same).

Equivalence between the two approaches is provided by the construction of the syntactic counterpart of a set of propositional bases. For that, we construct the syntactic counterpart of a propositional base as follows. For each interpretation ω ranked according to the internal pre-orders. We first generate all the formulas falsified by ω for K_i and attach to them the internal weight $p_{K_i}(\omega)$. We then remove the formulas falsified by an interpretation already processed. We finally remove subsummed formulas [4] and discard the formulas with null weight. From the construction of the syntactic counterpart of a set of propositional bases, we show the equivalence between semantic and syntactic approaches for merging within in the reversible framework.

Algorithm 1 syntactic counterpart

$\Sigma \leftarrow \emptyset, M \leftarrow \emptyset, S \leftarrow \emptyset, T \leftarrow \emptyset$
for each $\omega \in \mathcal{W}$ **do**
  $S \leftarrow \emptyset, T \leftarrow \emptyset$
  $\Sigma' \leftarrow \{(D_j, p_{K_i}(\omega)),\ 1 \leq j \leq card(\omega),\ \omega \not\models D_j)\}$
  $M \leftarrow M \cup \Sigma'$
  $S \leftarrow \{(D_j, p_{K_i}(\omega)) \in \Sigma',\ \exists (D_j, p_{K_i}(\omega')) \in M,\ with\ p_{K_i}(\omega') < p_{K_i}(\omega)\}$
  $\Sigma' \leftarrow \Sigma' - S$
  $T \leftarrow \{(D_k, p_{K_i}(\omega)) \in \Sigma' \mid \exists (D_j, p_{K_i}(\omega)) \in \Sigma',\ D_k \models D_j\}$
  $\Sigma' \leftarrow \Sigma' - T$
  $\Sigma \leftarrow \Sigma \cup \Sigma'$
end for
$\Sigma \leftarrow \{(\phi,\ p_{\Sigma_i}(\phi)) \in \Sigma_i \mid p_{\Sigma_i}(\phi) \neq 0\}$
return Σ

Proposition 1 *Let $E = \{K_1, \ldots K_n\}$ be a set of propositional bases and let $\mathcal{B} = \{\Sigma_1, \ldots \Sigma_n\}$ be its syntactic counterpart.*

$$\forall \omega \in \mathcal{W},\ \kappa_{\Sigma_1 \otimes \cdots \otimes \Sigma_n}(\omega) = p_{K_1 \oplus \cdots \oplus K_n}(\omega)$$

The proof is based on the construction of the global pre-order according to the semantic approach and the global weighted base according to the syntactic approach. We shows that the global weights are the same in the two approaches.

Generalizations

The approach for merging propositional belief bases stemming from Hamming distance (S. & R. 2005) can be easily captured within our reversible framework. In this approach no pre-order between sources is considered. In contrast local pre-orders are implicit pre-orders induced by the Hamming distance between interpretations. A distance between an interpretation ω and a propositional belief base K_i is defined by $d(\omega,\ K_i) = min_{\omega' \in Mod(K_i)}(d(\omega,\ \omega'))$. In our framework, for each belief base K_i an internal weight is a constant polynomial such that $p_{K_i}(\omega) = d(\omega,\ K_i)$. Performing the merging of n propositional belief bases amounts to compute a global weight as presented, the distance based fusion operators are represented within the reversible framework by pre-orders on polynomials. The pre-orders $\leq_{MAX}$, $\leq_{SUM}$, $\leq_{WS}$ and $\leq_{LMAX}$ are used to compare the polynomials corresponding to the global weights for the fusion operators MAX, SUM, WS et $LMAX$ given in (Konieczny & Pérez 1998). Like in (Benferhat *et al.* 2002a), the proposed framework gives a syntactic counterpart to the distance based fusion operations and moreover brings reversibility to both semantic and syntactic approaches. Besides the proposed reversible framework allows us to generalize the revision of an epistemic state by another epistemic state to the fusion of several epistemic states. Let $\Psi_1, \ldots, \Psi_n$, be n epistemic states. each epistemic state Ψ_i can be represented by a total pre-order on interpretations $\leq_{\Psi_i}$ or by a weighted belief base Σ_i. In case of revision $n = 2$. Revision can be easily captured within the proposed framework. According to a semantical point of view, the two epistemic states Ψ_1 and Ψ_2 are respectively represented by the internal pre-orders $\leq_{\Psi_1}$ and $\leq_{\Psi_2}$. For the revision of Ψ_1 by Ψ_2, the external pre-order is $\Psi_1 <_E \Psi_2$. For the revision with memory proposed in (Papini 2001), the global pre-order is obtained from the lexicographic pre-order on polynomials $\leq_{LEX}$. Under these hypothesis, we found again the results presented in (Benferhat *et al.* 2000). According to the syntactic approach, the two epistemic states Ψ_1 and Ψ_2 are respectively represented by the weighted bases Σ_1 and Σ_2. The external pre-order is the same as for the semantic approach and is denoted by $\Psi_1 <_{\mathcal{B}} \Psi_2$. After the construction of the global weight, the global pre-order on formulas is obtained by means of the lexicographic pre-order on polynomials. For $n > 2$, the reversible framework makes it possible to represent the fusion of epistemic states.

Conclusion

In this paper we presented a very general reversible framework for merging propositional belief bases. It makes it possible to represent within the same framework the case where the sources are ordered or not as well as the case where the items of information are explicitely or implicitely ordered or not ordered. We proposed both semantic and syntactic approaches for fusion within the reversible framework and we showed the equivalence between the semantic and syntactic approaches. We sowed that the proposed framework allows us to represent with a reversible framework the approach of merging propositional belief bases with implicit priorities stemming from Hamming distance and to provide a syntactic counterpart. We also showed that this framework allows for generalizing the revision of an epistemic state by another epistemic state to fusion of epistemic states in the case where the epistemic states are represented by total pre-orders.

In the context of submarine archeology, the construction of models of archeological objects requires photogrammetric measures and measures in laboratory. These measures are represented in propositional calculus and the proposed

[4] $(\phi,\ p_{\Sigma_i}(\phi)$ is subsummed by $(\psi,\ p_{\Sigma_i}(\psi)$ iff $\psi \models \phi$ and $p_{\Sigma_i}(\psi)) \leq p_{\Sigma_i}(\phi)$

reversible framework is suitable for the fusion of such pieces of information. However, we also have to deal with structured , semi-structured or hierarchical pieces of information. The fusion of such items of information is still problematical and will be the focus of a future work.

Acknowledgements

This work was supported by the Region PACA and COMEX company. We also thank the anonymous rewievers for their helpful remarks.

References

Baral, C.; Kraus, S.; Minker, J.; and Subrahmanian, V. 1992. Combining knowledge bases consisting in first order theories. *Computational Intelligence* 8(1):45–71.

Benferhat, S.; Konieczny, S.; Papini, O.; and Perez, R. P. 2000. Iterated revision by epistemic states: axioms, semantics and syntax. In *Proceedings of the 14th European conference on Artificial Intelligence (ECAI 2000)*, 13–17.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2002a. Possibilistic Merging and Distance-based Fusion of Propositional Information. *Annals of Mathematics and Artificial Intelligence* 34((1-3)):217–252.

Benferhat, S.; Dubois, D.; Lagrue, S.; and Papini, O. 2002b. Making revision reversible: an approach based on polynomials. *Fundamenta Informaticae* 53((3-4)):251–288.

Cholvy, L. 1998. Reasoning about merging information. *Handbook of Defeasible Reasoning and Uncertainly Management Systems* 3:233–263.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic Logic. *in Handbook of Logic in Artificial Intelligence and Logic Programming* 3:439–513.

Konieczny, S., and Pérez, R. P. 1998. On the logic of merging. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento*, 488–498.

Konieczny, S.; Lang, J.; and Marquis, P. 2002. Distance-based merging: A general framework and some complexity results. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, 97–108.

Laffage, C., and Lang, J. 2000. Logical representation of preferences for group decision making. In *7th International Conference on Krinciples of Knowledge Representation and Reasoning*, 457–468.

Lin, J. 1996. Integration of weighted knowledge bases. *Artificial Intelligence* 83:363–378.

Papini, O. 2001. Iterated revision operations stemming from the history of an agent's observations (extended version). In Williams, M. A., and Rott, H., eds., *Frontiers of Belief revision*, 279–301. Kluwer Academic Press.

Pearl, J. 2003. System Z: a natural ordering of default with tractable applications to default reasoning. In *Proc. of the 3rd Conf. on Theoritical Aspects of Reasoning about Knowledge (TARK'90)*, volume 2, 121–135. Morgan Kaufmann.

Revesz, P. Z. 1993. On the semantics of theory change: arbitration between old and new information. 12^{th} *ACM SIGACT-SGMIT-SIGART symposium on Principes of Databases* 71–92.

Revesz, P. Z. 1997. On the semantics of arbitration. *Journal of Algebra and Computation* 7(2):133–160.

S., K., and R., P. P. 2005. Propositionnal belief base merging or how to merge belief/goals coming from several sources and some links with social choice theory. *European Journal of Operational Research* 160(3):785–802.

6.6 Mutual Enrichment for Agents Through Nested Belief Change

Mutual Enrichment for Agents Through Nested Belief Change A Semantic Approach

Laurent Perrussel and **Jean-Marc Thévenin**
IRIT–Université Toulouse 1, Manufacture des Tabacs
21 allée de Brienne, F-31042
Toulouse Cedex - France
laurent.perrussel@irit.fr, thevenin@univ-tlse1.fr

Thomas Meyer
National ICT Australia and
Univerity of New South Wales
Sydney, Australia
thomas.meyer@nicta.com.au

Abstract

This paper focuses on the dynamics of nested beliefs in the context of agent interactions. Nested beliefs represent what agents believe about the beliefs of other agents. We consider the *tell* KQML performative which allows agents to send their own beliefs to others. Whenever agents accept a new belief, or refuse to change their own beliefs after receiving a message, both receiver and sender enrich their nested beliefs by refining their beliefs about (i) the other agent's beliefs and (ii) the preferences of the other agent. The main objective of nested beliefs is to improve cooperation between agents. We propose a logical formalisation of the acquisition process of nested beliefs and preferences. This acquisition process is the first step toward the elaboration of sophisticated interaction protocols.

Introduction

In a multi-agent context *nested beliefs* represent what agents believe about the beliefs of other agents. Several papers have shown how nested beliefs improve interaction and cooperation among agents (Fagin *et al.* 1995; Grasso, Cawsey, & Jones 2000; Cohen & Levesque 1990; Kinny *et al.* 1992; Sperber & Wilson 1986). Agents can

- initiate messages to solve discrepancies such as in argumentation dialogues (Grasso, Cawsey, & Jones 2000): the aim is that two agents try to converge toward a mutual belief. If agents handle beliefs and nested beliefs, they can evaluate how close they are from a common agreement;
- build teams and joint goals (Cohen & Levesque 1990; Kinny *et al.* 1992): nested beliefs are the first step that enable to establish mutual belief which next help to ensure that all members of a team share the same goal;
- evaluate the relevance of the messages they intend to send (Sperber & Wilson 1986): nested beliefs help agents involved in a persuasion dialogue to define in a more efficient way the relevant arguments.

There are very few papers dealing with the problem of how agents acquire nested beliefs (Herzig & Longin 2000; Dragoni, Giorgini, & Serafini 2002). None of them describe this process in the context of cooperative dialogues. In this paper we propose an acquisition process for nested beliefs which describes how agents construct nested beliefs as they receive information. The acquisition of nested beliefs is the foundation on which we can build sophisticated dialogues or cooperation protocols.

We consider multi-agent dialogues based on the *tell* KQML performative (Finin, Labrou, & Mayfield 1997) which enables an agent to send a message to inform a receiver agent that a statement holds in its own belief base. To increase the benefit of the cooperation we suppose that the sender provides the *source* of this statement (Perrussel & Thévenin 2004). The receiver may accept or ignore the incoming statement leading to non-prioritised revision of basic beliefs (Hansson 1999). Indeed to perform, revision agents need to be able to define preferences over their basic beliefs and thus the incoming statement may be considered as less preferred than conflicting basic beliefs. A common way to define preferences over basic beliefs is to maintain preferences over agents at the origin of basic beliefs; these preferences represent the degree of trust that an agent has in the other agents (Ramchurn, Hunyh, & Jennings 2004).

Based on this, we can define the acquisition of nested beliefs for the receiver of a *tell* performative. Since we consider a cooperative context, agents are supposed to be honest and thus regardless of the acceptance of the *tell* performative, the receiver is informed that the sender believes the incoming statement. After a *tell* performative the receiver performs non-prioritised revision on its nested beliefs about the sender. Since the sender provides the source of its beliefs, the receiver can deduce the preferences of the sender over the source of its beliefs, i.e. the receiver can deduce nested preferences concerning the sender.

Considering that the receiver may ignore the incoming statements, the sender can also enrich its nested beliefs in several ways. For this, we introduce two performatives *accept* and *deny* which enable the receiver agent to inform the sender whether or not it actually changed its own beliefs after a *tell*. Using these performatives the sender can enrich its nested beliefs in several ways. After an *accept* the sender believes that the receiver believes in the content of the *tell*. If, during the revision process, an inconsistency occurs in the nested beliefs of the receiver, the sender can refine its knowledge about the preferences of the receiver. This is also the case after a *deny* statement.

Consider, for example, agent Peter receiving messages from agent Paul and the police department. Suppose Peter

considers Paul to be less reliable than the police department. At time 0, Paul tells Peter that *John is a murderer* (origin Paul). At time 1 Peter accepts the message. At time 2, Paul tells Peter that *if John is a murderer, John will go to jail* (origin Paul). Peter accepts the message and consequently Peter believes that *John will go to jail* at time 3. In addition, Peter believes that Paul believes both statements. After the second accept, Paul also believes that Peter believes both statements. At time 4, the police department tells Peter that *John is not a murderer* (origin Police). Because of his preferences, Peter accepts message at time 5 and now believes *John is not a murderer*. In addition, Peter believes that the police department believes *John is not a murderer*. After the accept message, the police believes that Peter believes *John is not a murderer*. Because he believes that Paul still believes that *John is a murderer*, at time 6, Peter tells Paul that *John is not a murderer* and the origin of this information is the police. Whether or not Paul accepts Peter's message, Paul has to change his nested beliefs about Peter. Paul now believes that Peter believes *John is not a murderer* and that Peter considers the police to be more reliable than himself.

As we can see, messages received by an agent trigger changes in its nested beliefs. A new nested belief may entail inconsistencies with the already adopted nested beliefs. So agents have to reconsider not only their nested beliefs, but also their nested preferences.

The paper is structured as follows. First we present a logical system for representing nested beliefs, preferences and performatives *tell*, *accept*, and *deny*. This is followed by a description of the organisation of a cooperative *tell/accept-deny* dialogue. After that, we describe the mutual enrichment; first for the receiver and then for the sender. We conclude with a discussion on related work and outlining some future work.

The Logical Framework

To represent an agent's beliefs we use signed statements. A *signed statement* is a pair ⟨statement, origin of the statement⟩ (usually the sender of the statement). Let $\mathcal{L}_0$ be a propositional language and A a set of agent identities. We define a signed statement as a pair $\langle \phi_0, a \rangle$ where ϕ_0 is a $\mathcal{L}_0$-formula and $a \in A$ is the origin of ϕ_0. Let $\mathcal{S}$ be the set of all sets of signed statements: $\mathcal{S} = 2^{\mathcal{L}_0 \times A}$. A set $S \in \mathcal{S}$ of signed statements is *consistent* iff the conjunction of its propositional formulae is consistent: $\bigwedge_{\langle \phi_0, b \rangle \in S} \phi_0 \not\models_{\mathcal{L}_0} \bot$. We assume that inference for agents is a one step process.

Belief state

The *belief state* of an agent is a pair ⟨set of signed statements, set of sets of signed statements⟩. The first set describes the *basic beliefs* of the agent: what it currently believes. The second set describes the *nested beliefs* of the agent: what it believes about the basic beliefs of other agents. We focus on nested beliefs in a dialogue context: what a receiver agent believes about the sender of a statement. In this context we handle nested beliefs at only one level. Nested beliefs are organised as one set for each agent. Basic and nested beliefs change with respect to the flow of messages and thus are indexed by integers which represent state labels.

Definition 1 (Belief state) *A belief state BS^n_a of agent a is a pair $\langle CB^n_a, NB^n_a \rangle$ s.t. (i) $CB^n_a \in \mathcal{S}$ represents the basic beliefs of agent a at time n and (ii) $\forall CB^n_{a,b} \in NB^n_a, CB^n_{a,b} \in \mathcal{S}$ represents the nested beliefs of agent a about agent b at n. We require that $CB^n_a = CB^n_{a,a}$ and that every $CB^n_{a,b} \in NB^n_a$ be consistent. Let $\mathcal{B}$ be the set of all possible belief states.*

Observe that the requirement that $CB^n_a = CB^n_{a,a}$ is a simple introspection property. Agents revise their basic beliefs and nested beliefs each time they received a *tell* performative. Let S be a consistent set of signed beliefs and $*$ be a revision operator (Gärdenfors 1988); $S^*_{\langle \phi_0, a \rangle}$ denotes the revision of S by $\langle \phi_0, a \rangle$.

Preferences of agents

Preferences may be defined taking various matters into account (Perrussel & Thévenin 2004; Perrussel 2003; del Cerro *et al.* 1998). We simply assume that agents have preferences over the set of agents A which describe the reliability of the sources of information (Dragoni & Giorgini 1999; Perrussel & Thévenin 2004), i.e. the level of trust an agent has about the other agents with which it interacts. We suppose that agents are equally reliable when they can't be distinguished, which entails a total preorder. Let $\preceq^n_a$ be a total preorder over A representing agent a's preferences at time n. $b \preceq^n_a c$ stands for agent c is at least as preferred as b for agent a at time n. $b \prec^n_a c$ stands for agent c is strictly preferred to b: $b \preceq^n_a c$ and $c \not\preceq^n_a b$ at n. $b =^n_a c$ stands for agents b and c are equally preferred: $b \preceq^n_a c$ and $c \preceq^n_a b$.

As is the case for beliefs, agents can handle nested preferences. Nested preferences represent what agents believe about the preferences of other agents. $c \preceq^n_{a,b} d$ means: agent a believes that for agent b agent d is at least as preferred as c at n. The meaning of $\prec^n_{a,b}$, respectively $=^n_{a,b}$, is similar to the meaning of $\prec^n_a$, respectively $=^n_a$.

Definition 2 (Preference state) *A preference state PS^n_a of agent a is a pair $\langle \preceq^n_a, NP^n_a \rangle$ s.t. (i) $\preceq^n_a$ is a total preorder representing basic preferences of agent a at time n and (ii) every $\preceq^n_{a,b} \in NP_a$ is a total preorder representing agent b's preferences according to a at time n. We require that $\preceq^n_a = \preceq^n_{a,a}$. $\mathcal{P}$ is the set of all possible preference states.*

The sequence of belief and preference states of each agent is represented by a function D which associates integers representing state labels, and agent identities with the agent states:

$$D : \mathbb{N} \rightarrow A \rightarrow \mathcal{B} \times \mathcal{P}$$

Let $D(n)(a) = \langle BS^n_a, PS^n_a \rangle = \langle \langle CB^n_a, NB^n_a \rangle, \langle \preceq^n_a, NP^n_a \rangle \rangle$. From the pair $\langle BS^n_a, PS^n_a \rangle$ we extract a pair representing the basic belief and preferences of a: let D^B be a function based on D s.t. for all a and n, $D^B(a)(n) = \langle CB^n_a, \preceq^n_a \rangle$. Nested beliefs and preferences entail the definition of other basic models. Let NB^n_a, NP^n_a be the nested beliefs and preferences based on $D(n)(a)$.

Let $D^B_{N(a)}$ be a function issued from D s.t. for all b, $D^B_{N(a)}(n)(b) = \langle CB^n_{a,b}, \preceq^n_{a,b}\rangle$. So $D^B_{N(a)}(n)$ represents all beliefs and preferences appearing as nested beliefs and preferences in $D(n)(a)$.

Let S be any set of signed statements (basic or nested beliefs) and $\preceq$ the corresponding preference relation (basic or nested). The logical closure of S w.r.t. $\preceq$ is obtained as follows. By $\min(S, \preceq)$ we denote the set of the least preferred agent identities w.r.t. $\preceq$ among agent identities signing beliefs of S:

$$\min(S, \preceq) = \{a | \langle \phi_0, a\rangle \in S \text{ and } \forall \langle \psi_0, b\rangle \in S, a \preceq b\}$$

We suppose that statements entailed by S are signed with the least preferred agent identities of the minimal subsets of S entailing them:

$$\begin{aligned}\mathrm{Cn}(S, \preceq) = \{\langle \psi_0, a\rangle | \exists S' \subseteq S \text{ s.t. } & \bigwedge_{\langle \phi_0, b\rangle \in S'} \phi_0 \models_{\mathcal{L}_0} \psi_0 \\ & \text{and } \nexists S'' \subset S' \text{ s.t. } \bigwedge_{\langle \phi_0, b\rangle \in S''} \phi_0 \models_{\mathcal{L}_0} \psi_0 \\ & \text{and } a \in \min(S', \preceq)\} \\ \cup \quad & \{\langle \psi_0, a\rangle | \models_{\mathcal{L}_0} \psi_0 \text{ and } a \in A\}\end{aligned}$$

Example 1 *We continue with our running example. At time 3, agent peter has the following belief state* $\langle CB^3_{peter}, NB^3_{peter}\rangle$*:*

$$CB^3_{peter} = \{\langle murd, paul\rangle, \langle murd \rightarrow jail, paul\rangle\}$$
$$CB^3_{peter,paul} = \{\langle murd, paul\rangle, \langle murd \rightarrow jail, paul\rangle\}$$
$$CB^3_{peter,police} = \emptyset$$

Considering that peter does not know paul and police's preferences at the beginning, the preference state $\langle \preceq^3_{peter}, NP^3_{peter}\rangle$ *of peter is:*

$$paul \prec^3_{peter} peter \prec^3_{peter} police$$
$$paul =^3_{peter,paul} peter =^3_{peter,paul} police$$
$$paul =^3_{peter,police} peter =^3_{peter,police} police$$

The first line represents peter's basic preferences and the last two lines the nested preferences of peter. According to these belief and preference states we get that:

$$\langle jail, paul\rangle \in \mathrm{Cn}(CB^3_{peter}, \preceq^3_{peter})$$

Notice that the belief and preference states correspond to the following value of function D:

$$D(3)(peter) = \langle \langle CB^3_{peter}, NB^3_{peter}\rangle, \langle \preceq^3_{peter}, NP^3_{peter}\rangle\rangle$$

Action performatives

Now, we present the performatives which lead to the dynamics of belief and preference states change. As mentioned in the introduction, an agent may issue a *tell* performative to inform a receiver agent about its basic beliefs. The receiver uses a prioritised belief revision operator $*$ to change its nested beliefs about the sender. As an acknowledgment of the *tell* performative, the receiver informs the sender with an *accept* (respectively *deny*) performative if the incoming statement has been incorporated in its basic beliefs. The sender in turn applies prioritised revision to its nested beliefs about the receiver.

We represent these interactions between agents using the following three performatives:

- $\texttt{Tell}(s, r, \phi_0, a)$ stands for: agent s informs r that it believes ϕ_0 signed by a according to the standard KQML semantics (Finin, Labrou, & Mayfield 1997). When receiving a Tell performative, agent r revises its belief state by $\langle \phi_0, a\rangle$ in a non-prioritised way (Hansson 1999) according to its preferences.
- $\texttt{Accept}(r, s, p)$ stands for: agent r informs s that it accepts the performative p. If $p = \texttt{Tell}(s, r, \phi_0, a)$ then $\texttt{Accept}(r, s, p)$ means that agent r has revised its basic beliefs by $\langle \phi_0, a\rangle$ and thus believes ϕ_0.
- $\texttt{Deny}(r, s, p)$ stands for: agent r informs s that it refuses to process the performative p. If performative $p = \texttt{Tell}(s, r, \phi_0, a)$ it means that agent r has not revised its basic beliefs by $\langle \phi_0, a\rangle$.

Definition 3 (Performative) *The set* $\mathcal{PERF}$ *of all performative actions is defined as follows. If* $\phi_0 \in \mathcal{L}_0$ *s.t.* $\phi_0 \not\models_{\mathcal{L}_0} \bot$, $s, r, a \in A$, *then* $\texttt{Tell}(s, r, \phi_0, a) \in \mathcal{PERF}$. *If* $p \in \mathcal{PERF}$ *then* $\texttt{Accept}(r, s, p)$, $\texttt{Deny}(r, s, p) \in \mathcal{PERF}$.

The dynamics of the system is given by performatives. At each step one agent receives a performative and may change its state accordingly, i.e. performatives are viewed as transition functions. The sequence of states will be defined with respect to a sequence of performatives. A sequence of actions σ is a function which associates integers with performatives:

$$\sigma : \mathbb{N} \rightarrow \mathcal{PERF}$$

Example 2 *We continue our running example. The performatives that occur at* 0, 1, 2 *and* 3 *define the following sequence:*

$$\begin{aligned}\sigma(0) &= \texttt{Tell}(paul, peter, murd) \\ \sigma(1) &= \texttt{Accept}(peter, paul, \texttt{Tell}(paul, peter, murd)) \\ \sigma(2) &= \texttt{Tell}(paul, peter, murd \rightarrow jail) \\ \sigma(3) &= \texttt{Accept}(peter, paul, \texttt{Tell}(paul, peter, murd \rightarrow jail))\end{aligned}$$

Dynamics of a cooperative dialogue

In this section, we describe how basic beliefs and preferences and performatives are linked when we consider a cooperative dialog. In the following, we formally describe what we mean by a cooperative dialogue. First, we express the honesty postulate (Hon) as follows. This postulate, which is recommended in a cooperative context, states that if a Tell performative occurs, then at the same time the sender believes the corresponding signed statement. Let CB^n_s be the basic belief of agent s at n w.r.t. D.

(Hon) For any n if $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ then $\langle \phi_0, a\rangle \in \mathrm{Cn}(CB^n_s, \preceq^n_s)$.

This actually enforces the standard KQML semantics of Tell (Finin, Labrou, & Mayfield 1997).

In the following we constrain the basic preferences of all agents so that these preferences should not change. We justify this constraint with one of the aims of this paper: how agents can deduce nested preferences. This aim becomes useless if first agents are not honest and second agents always change their basic preferences.

(FPref) For any n and any $a \in A$ $\preceq_a^n = \preceq_a^{n+1}$.

This bring us to the receiver's acknowledgement in a cooperative context. Whenever agent r receives $\texttt{Tell}(s, r, \phi_0, a)$, r determines if its beliefs are consistent with ϕ_0, and if not, which beliefs have to be dropped by taking into account its preference relation. Due to the non-prioritised aspect the receiver may deny or accept the Tell performative. If agent r believes $\neg\phi_0$ and the signatures of $\neg\phi_0$ are at least as preferred as a (the signature of ϕ_0), agent r denies the Tell performative. It means that basic preferences are used to implement non-prioritised revision on basic beliefs. In the next sections we show that nested preferences change according to the prioritised revision applied to nested beliefs.

(DT) $\sigma(n+1) = \texttt{Deny}(r, s, \texttt{Tell}(s, r, \phi_0, a))$ iff $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ $\&\exists\langle\neg\phi_0, b\rangle \in \mathrm{Cn}(CB_r^n, \preceq_r)$ s.t. $a \preceq_r b$.

Otherwise there is no conflict or a is strictly more preferred than all the signatures of $\neg\phi_0$, and r thus accepts the Tell performative.

(AT) $\sigma(n+1) = \texttt{Accept}(r, s, \texttt{Tell}(s, r, \phi_0, a))$ iff $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ $\& \forall\langle\neg\phi_0, b\rangle \in \mathrm{Cn}(CB_r^n, \preceq_r), b \prec_r a$

In other words, in a cooperative context, if agent r has no reason to refuse an incoming statement, then it should accept it. At the opposite, if it refuses an incoming statement then it should have a rationale which justifies this refusal. It means that the nested beliefs and preferences of agents which are not involved in a dialogue do not change. Agents are honest and thus they change their beliefs only if they received a *tell* performative. The conditions (KeNB1-1) and (KeNB1-2) enforce this constraint.

(KeNB1-1) if $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ then $(\forall b \in A - \{r\}) D(n+1)(b) = D(n)(b)$.

(KeNB1-2) if $\sigma(n) = \texttt{Accept}(r, s, p)$ or $\texttt{Deny}(r, s, p)$ then $(\forall b \in A - \{s\}) D(n+1)(b) = D(n)(b)$.

Dynamics of the Receiver

In this section, we describe the dynamics of the nested beliefs and nested preferences of agent r for the Tell performative. First, the nested beliefs and preferences about all agents except s are unchanged. Condition (KeNB2) propagates them to the next state and ensures that no new nested beliefs or preferences appear.

(KeNB2) If $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ then $(\forall b \in A - s\}) (D(n+1))^B_{N(r)}(b) = (D(n))^B_{N(r)}(b)$.

Next, agent r changes its beliefs about s, whether or not it ignores the input statement. Because agents are supposed to be honest, agent r performs a standard prioritised revision of its nested beliefs about s by signed statement $\langle\phi_0, a\rangle$. For the next condition, let $*$ be a revision function and D be a dynamic model and $D(n)^B_{N(r)}(s) = \langle CB^n_{r,s}, \preceq^n_{r,s}\rangle$. Condition (AdNB1) states that after the Tell performative r believes that s believes ϕ_0 and thus does not believe $\neg\phi_0$ anymore (because belief states are consistent).

(AdNB1) If $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ then $CB^{n+1}_{r,s} = (CB^n_{r,s})^*_{\langle\phi_0, a\rangle}$.

As mentioned before, we do not insist on a specific method for revising nested beliefs.

If agent r currently believes that s believes $\neg\phi_0$ signed by b then it refines the s preferences as follows. According to the honesty principle and condition (AT), a is better than b. In addition, we require that the nested preferences of agent s about all agents except a are unchanged. And lastly, if r does not believe that s believes $\neg\phi_0$ then it cannot draw conclusion about agent s preferences. Indeed, agent r can refine preferences only if it faces inconsistencies during the revision process. So, we require that if r does not believe that s believes $\neg\phi_0$ then all the nested preferences about s are unchanged.

In order to formalise these requirements, we provide an explicit procedure that refines preferences. The receiver refines its nested preferences because it can remove some preferences; i.e. it helps agent r to go toward an order that is more precise.

(Ad-KeNP1) Let $D(n)^B_{N(r)}(s) = \langle CB^n_{r,s}, \preceq^n_{r,s}\rangle$. Let $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$. All nested preferences at n are propagated at time $n+1$ as follows: $\preceq^{n+1}_{r,s} = (\preceq^n_{r,s} - \{a \preceq^n_{r,s} b | \langle\neg\phi_0, b\rangle \in CB^n_{r,s}\}) \cup \{b \preceq^n_{r,s} a | \langle\neg\phi_0, b\rangle \in CB^n_{r,s}\}$.

It is easily shown that this condition preserves the ordering for nested preference as a total preorder.

Dynamics of the Sender

In this section we describe the dynamics of the sender's nested beliefs and preferences with respect to the performatives Accept and Deny. Firstly, the nested beliefs of agents not involved in the Accept or Deny performative do not change.

(KeNB3) If $\sigma(n) = \texttt{Accept}(r, s, p)$ or $\texttt{Deny}(r, s, p)$ and $\sigma(n-1) = \texttt{Tell}(s, r, \phi_0, a)$ then $(\forall b \in A - \{a\}) (D(n+1))^B_{N(s)}(b) = (D(n))^B_{N(s)}(b)$.

Next, we focus on the nested beliefs about r. According to condition (AT), if r accepts the message of s, agent s believes that r believes ϕ_0 and thus does not believe $\neg\phi_0$, which results in s revising its nested beliefs about r with $\langle\phi_0, a\rangle$ (a prioritised revision).

(AdNB3-1) If $\sigma(n) = \texttt{Accept}(r, s, \texttt{Tell}(s, r, \phi_0, a))$ then $CB^{n+1}_{s,r} = (CB^n_{s,r})^*_{\langle\phi_0, a\rangle}$.

According to condition (DT), if r refuses the Tell performative then s concludes that r believes $\neg\phi_0$ and r believes that the signature of $\neg\phi_0$ has to be more trusted than a. So s revises its nested beliefs about r with the signed statement $\langle\neg\phi_0, b\rangle$ (again, a prioritised revision) so that signature b of

$\neg\phi_0$ is preferred to a w.r.t. the nested preferences of s about r:

(AdNB3-2) If $\sigma(n) = \texttt{Deny}(r, s, \texttt{Tell}(s, r, \phi_0, a))$ then $CB^{n+1}_{s,r} = (CB^n_{s,r})^*_{\langle\neg\phi_0,b\rangle}$ s.t. $a \preceq_{s,r} b$.

This bring us to the sender's nested preferences. Firstly, the sender's nested preferences about agents other than r do not change. This is, in fact, a consequence of condition (KeNB3). Next, agent s draws conclusions about a depending on whether r accepts or refuses a `Tell` performative. Whenever agent r accepts, agent s refines its nested preferences only if s currently believes that r believes $\neg\phi_0$. In that context s refines agent r's preferences. We require that a be strictly more preferred than the signatures of $\neg\phi_0$ if r accepts the performative $\texttt{Tell}(s, r, \phi_0, a)$. We also require that if r refuses the message, the nested preferences of s about the signatures of $\neg\phi_0$ change. According to condition (DT) they have to be as preferred as a. Finally, we need a requirement, similar to those for the receiver's nested preferences, for those nested preferences, about r, of s that do not change. In order to formalise these requirements, we give an explicit procedure to change the nested preferences, as in the case for the receiver's nested preferences. The `Accept` performative helps the sender to refine its nested preferences since it allows to remove some preferences; i.e. it helps agent s to go toward a stricter order. Removing preferences means that agent s already believes that r believes $\neg\phi_0$ and thus a is strictly preferred to all the signatures of $\neg\phi_0$.

(Ad-KeNP2) Let $D(n)^B_{N(s)}(r) = \langle CB^n_{s,r}, \preceq^n_{s,r}\rangle$. Let $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ and $\sigma(n+1) = \texttt{Accept}(r, s, \texttt{Tell}(s, r, \phi_0, a))$. All nested preferences at n are propagated at time $n+1$ as follows: $\preceq^{n+1}_{s,r} = \preceq^n_{s,r} - \{a \preceq^n_{s,r} b | \langle\neg\phi_0, b\rangle \in CB^n_{s,r}\} \cup \{b \preceq^n_{s,r} a | \langle\neg\phi_0, b\rangle \in CB^n_{s,r}\}$.

If r does not accept the incoming message, agent s also changes its nested beliefs about r. The `Deny` performative does not help agent s to refine its nested preferences since we only add nested preferences.

(AdNP3) Let $D(n)^B_{N(s)}(r) = \langle CB^n_{s,r}, \preceq^n_{s,r}\rangle$. Let $\sigma(n) = \texttt{Tell}(s, r, \phi_0, a)$ and $\sigma(n+1) = Deny(r, s, \texttt{Tell}(s, r, \phi_0, a))$. All nested preferences at n are propagated at time $n+1$ as follows: $\preceq^{n+1}_{s,r} = \preceq^n_{s,r} \cup \{a \preceq^n_{s,r} b | \langle\neg\phi_0, b\rangle \in CB^n_{s,r}\}$.

Again, it is easily shown that the conditions preserves the ordering for nested preference as total preorder.

Example 3 *We now conclude our example. Let us suppose that each agent shares the same kind of revision function $*$ defined as safe base revision actions (Alchourrón & Makinson 1985; Nebel 1989; 1992). At time 4, the police tells peter that* John is not a murderer*, and peter accepts it. Formally, $p_4 = \texttt{Tell}(police, peter, \neg murd, police)$ and $p_5 = \texttt{Accept}(peter, police, p_4)$. According to conditions (AdNB1) and (AdNB3-1) we get the new nested beliefs:*

$$\langle\neg murd, police\rangle \in CB^4_{peter,police}$$
$$\langle\neg murd, police\rangle \in CB^5_{police,police}$$

In other words the police believes that peter believes that John is not a murderer. *At time 1 peter accepted paul's message that* John is a murderer *(murd) which gives the following nested belief and preference (because of condition (KeNB1)):*

$$\langle murd, paul\rangle \in CB^1_{paul,peter}$$
$$paul =_{peter} police \in \preceq^1_{paul,peter}$$

At time 6, this nested belief is still true according to the accepted behaviour of nested belief given by conditions (KeNB1-1), (KeNB1-2), (AdNB3-1) and (Ad-KeNP2). At time 6 peter tells paul that John is not a murderer*: $p_6 = \texttt{Tell}(peter, paul, \neg murd, police)$. According to conditions (AdNB1) and (Ad-KeNP1) we obtain the following new nested beliefs and preferences for paul:*

$$\langle\neg murd, police\rangle \in CB^6_{paul,peter}$$
$$paul <_{peter} police \in \preceq^6_{paul,peter}$$

So, at stage 6 paul has refined its nested preferences about peter. At the beginning of the dialogue paul does not know whether peter prefers paul or the police. At stage 6, paul believes that peter prefers the police to himself.

Conclusion

We have presented a formalisation for handling the dynamics of nested beliefs and preferences in the context of agent interactions where agents are cooperative. We have shown how agents acquire nested beliefs and preferences. For this we have presented a logical framework to describe nested beliefs, preferences, and performatives. This framework is useful for specifying properly the expected behaviour of agents handling the `Tell`, `Accept` and `Deny` performatives.

Related work includes that of (Dragoni, Giorgini, & Serafini 2002), which proposes a logic-based definition for nested belief change and mental state construction related to agent-communication languages. Our proposal differs from theirs in that it deals with preferences of agents. Our work is also closely related to (Roorda, van der Hoek, & Meyer 2002; Perrussel & Thévenin 2004) where it is shown how to embed basic belief change in a logical system. We extend this by considering nested beliefs and preferences.

We have started to build a logical language based on dynamic epistemic logic (Meyer & van der Hoek 1995; van der Hoek & Wooldridge 2003) in order to reason about dialogues. Our aim is to define a semantics based on the semantics proposed in this paper. Moreover, this will give us a proof theory which will help us to reason about dialogue specifications. In a more long term, we plan to revisit persuasion protocols (Parsons, Wooldridge, & Amgoud 2002) so that agents, with the help of nested beliefs and preferences, produce smarter arguments in order to be more convincing.

References

Alchourrón, C., and Makinson, D. 1985. The logic of theory change: safe contraction. *Studia Logica* 44:405–422.

Cohen, P., and Levesque, H. 1990. Rational Interaction as the Basis for Communication. In Cohen, P.; Morgan, J.;

and Pollack, M., eds., *Intentions in Communication*, 221–256. MIT Press.

del Cerro, L.; Herzig, A.; Longin, D.; and Rifi, O. 1998. Belief reconstruction in cooperative dialogues. In Giunchiglia, F., ed., *Proc. of AIMSA'98*, volume 1480 of *LNCS*, 254–266. Springer Verlag.

Dragoni, A., and Giorgini, P. 1999. Revising beliefs received from multiple sources. In Williams, M., and Rott, H., eds., *Frontiers of Belief Revision, Applied Logic*. Kluwer.

Dragoni, A.; Giorgini, P.; and Serafini, L. 2002. Mental States Recognition from Communications. *J. Logic Computat.* 12(1):119–136.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. Cambridge MA: The MIT Press.

Finin, T.; Labrou, Y.; and Mayfield, J. 1997. KQML as an agent communication language. In Bradshaw, J., ed., *Software Agents*. MIT Press.

Gärdenfors, P. 1988. *Knowledge in flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Grasso, F.; Cawsey, A.; and Jones, R. 2000. Dialectical argumentation to solve conflicts in advice giving: a case study in the promotion of healthy nutrition. *International Journal of Human-Computer Studies* 53(1077–1115).

Hansson, S. O. 1999. A survey of non-prioritized belief revision. *Erkenntnis* 50:413–427.

Herzig, A., and Longin, D. 2000. Belief dynamics in cooperative dialogues. *J. of Semantics* 17(2). vol. published in 2001.

Kinny, D.; Ljungberg, M.; Rao, A. S.; Sonenberg, E.; Tidhar, G.; and Werner, E. 1992. Planned team activity. In Castelfranchi, C., and Werner, E., eds., *Artificial Social Systems — Proceedings of MAAMAW'92*, volume 830 of *LNAI*, 226–256. Springer-Verlag.

Meyer, J.-J. C., and van der Hoek, W. 1995. *Epistemic Logic for AI and Computer Science*. Cambridge University Press.

Nebel, B. 1989. A knowledge level analysis of belief revision. In Brachman, R.; Levesque, H.; and Reiter, R., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the first International Conference (KR'89)*, 301–311. Toronto (ON), Canada: Cambridge University Press.

Nebel, B. 1992. Syntax-based approaches to belief revision. In Gärdenfors, P., ed., *Belief revision*, volume 29 of *Journal of Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press. 52–88.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2002. An analysis of formal inter-agent dialogues. In Gini, M.; Ishida, T.; Castelfranchi, C.; and Johnson, W. L., eds., *Proceedings of AAMAS'02*, 394–401. ACM Press.

Perrussel, L., and Thévenin, J. 2004. A logical approach for describing (dis)belief change and message processing. In *Proceedings of AAMAS'04*, 614–621. IEEE C.S.

Perrussel, L. 2003. Handling sequences of belief change in a multi-agent context. In Schillo, M.; Klusch, M.; Muller, J.; and Tianfield, H., eds., *Proceedings of MATES'03*, volume 2831 of *LNCS*. Springer.

Ramchurn, S. D.; Hunyh, D.; and Jennings, N. R. 2004. Trust in multi-agent systems. *Knowledge Engineering Review*.

Roorda, J.; van der Hoek, W.; and Meyer, J. 2002. Iterated Belief Change in MAS. In Castelfranchi, C., and Johnson, W., eds., *Proceedings of AAMAS'02*, 889–896. ACM Press.

Sperber, D., and Wilson, D. 1986. *Relevance*. Harvard University Press.

van der Hoek, W., and Wooldridge, M. 2003. Towards a logic of rational agency. *Journal of the IGPL* 11(2):133–157.

6.7 Getting Possibilities from the Impossible

Getting Possibilities from the Impossible

Corinna Elsenbroich, Dov Gabbay, Odinaldo Rodrigues
Department of Computer Science
King's College London
corinna.j.elsenbroich@kcl.ac.uk, dov.gabbay@kcl.ac.uk, odinaldo.rodrigues@kcl.ac.uk

Abstract

Abduction is often constructed as a consistent expansion of a database, i.e. abduction demands that a database Δ expanded by an explanation ε is not inconsistent; $\Delta \cup \varepsilon \not\models \bot$. The constraint is sensible if we stay within classical logic. Without it we end up with every formula inconsistent with Δ being an explanation for any other formula. But what if Δ is inconsistent itself? Then consistent abduction battles like Don Quixote against windmills. All we want from abduction is that it does not *introduce* inconsistency into a set Δ.

This article proposes a proof theory for abduction that can be applied to inconsistent databases by demanding that an abductively derived formula must be from the intersection of all maximal consistent subsets of the database.

Introduction

Understanding the form of reasoning traditionally called abduction is essential for common sense reasoning as our reasoning from observations to explanations is a fundamental source of new knowledge, i.e. learning. In general, abduction is a form of backward reasoning from a set of events back to a cause. For example from a set of clues to a murderer (criminology), a set of symptoms to a disease (medical diagnosis), or from a malfunction in a system to a faulty part in a system (model-based diagnosis). Formally, this inference can be expressed as

$$\frac{\begin{array}{c} B \\ A \rightarrow B \end{array}}{A}$$

Within classical logic this is a *non sequitur* inference, called *affirming the consequent*. Thus we have to constrain this reasoning in some ways and cannot just add this rule to a deductive calculus. There are different constraints for abduction, the most common go by the following names:[1]

Consistency	Minimality	Relevance$_1$	Relevance$_2$
$\Delta + A \not\models \bot$	A is *minimal*.	$A \not\models B$	$\Delta \not\models B, A \not\models B$

These criteria can be taken in a pick and choose fashion. For example, we might want consistent and minimal abduction but leave the case, where B is added as its own explanation, as a limiting case and thus forsake both relevance criteria. However, the most fundamental of the above table is consistency. There is basically no definition of abduction in the literature that does not have the requirement of consistency, unless the inference is left completely unconstrained[2] as $\Delta + A \models B$. The reason for this requirement is clear. If our object language is classical logic, the addition of A to a set Δ could lead to a trivialisation of $Cn(\Delta + A)$ by inconsistency, thus leading to every formula inconsistent with Δ being an explanation for every formula.

The first effect of the consistency rule is that abduction in the literature is seen as a special form of belief expansion[3]. This means that a set $\mathcal{E}$ of explanations for B is found and an element $A \in \mathcal{E}$ is selected and then added to the database. If however no consistent explanation is found, either B itself is added (expansion as the limiting case) or the abduction is declined (failure as the limiting case).[4]

Clearly this is a restrictive view of abduction and has the problem that abduction cannot challenge our background theory Δ as it never triggers a revision. This problem was identified by Lobo and Uzcategui in (Lobo & Uzcátegui1996a,1996b,1997) where the authors define abductive change on top of an AGM revision operator. This way formulae arrived at by abductive inference might trigger a revision of beliefs.

Although we are in favour of abduction challenging Δ and abduction not being restricted to the lucky, consistent cases, we think that it is dangerous to treat abductively inferred beliefs equal to knowledge. After all, explanations are hypothetical and abductive inference can be superseded by new information.[5] Any revision framework must deal with this hypothetical character of abduction and the above cited expansion and revision frameworks do not.

This problem becomes especially pressing when the theory Δ is inconsistent from the outset. We might be able

[1] The constraint Relevance$_2$ is often called *explanatory abduction*, cf. (Aliseda-Llera 1997).

[2] Often referred to as *plain abduction*.

[3] Used here as a technical term in line with the AGM framework of belief change, cf. (Alchourrón, Gärdenfors, & Makinson 1985) and (Alchourrón & Makinson 1982).

[4] For a detailed discussion of abductive belief expansion see (Pagnucco1996, 1995) and (Dias & Wassermann 2001).

[5] Abduction is an intrinsically non-monotonic form of reasoning.

to find a set of explanations for some observation and perform a revision on Δ with the selected explanation A. This revision however would be according to AGM constraints. Thus the revision is triggered by a hypothetically inferred formula and might occur in a part of the theory that is *completely independent* of the newly obtained beliefs. Still it would trigger a decision of which formulae to contract to restore consistency.

To combat the first problem of enlarging the scope of abduction to explanations that might be inconsistent with a theory, we define a proof theory for abduction in the next section, where abductive inferences are modalised, i.e. explanations are inferred as possibilities. This way all possible explanations can be added to a proof without causing inconsistency with each other. On potentially causing inconsistency with the theory, explanations can be retracted.

To combat the second problem, where the underlying theory is inconsistent from the outset, we adapt the above proof theory to a paraconsistent setting. In the remainder of the article we give semantics for both proof theories which intuitively capture the syntactic frameworks introduced and we show correspondence between syntax and semantics.

Abductive Proof Theory

In constructing an abductive proof theory, the main idea is to apply a rule of inference within a proof to specific formulae under certain conditions. The starting point for the original idea of what is called *adaptive logic*[6] was a paraconsistent logic which denies the application of the disjunctive syllogism. However, if the specific formulae the rule is applied to are consistent the disjunctive syllogism is a truth preserving rule of inference. In an adaptive logic, in these *specific cases* the application of the disjunctive syllogism is permitted. Similarly, the rule of inference

$$\frac{\begin{array}{c}B\\A \rightarrow B\end{array}}{A}$$

is a fallacy, i.e. a non-permitted rule of inference in classical logic. However, if we want to deduce a formula B from a set Δ containing the rules $A_1 \rightarrow B$,..., $A_n \rightarrow B$, some A_i must be true.[7] Thus, if all other A_j are known to be false, the inference to A_i seems reasonable. Thus we can describe a proof rule saying that if A_i does not cause inconsistency, i.e. it is not in the set of eliminated A_j, we can infer it. As we are looking for an A_i that does not cause inconsistency, we have to define the proof rule with respect to a database Δ in order to be able to check for inconsistency. Thus the above proof rule is stated with respect to some Δ as follows:[8]

$$\frac{\Delta \vdash A \rightarrow B \quad \Delta + A \nvdash \bot}{\Delta \vdash A}$$

Now however, we might have the situation that the set $\Delta' = \Delta + A$, we obtain after application of the proof rule for abduction, gives us other explanations as well. These explanations might not be inconsistent with Δ but with A. At this point A is blocking the addition of other explanations. In order to safeguard against this situation we have to make an addition to this rule. Let us define the language of abductive proofs in order to describe the proof theory. Let us have a classical propositional logic with modus ponens as rule of inference. We use a division into an object and a metalanguage.

Definition 1.

Let $\mathcal{P}$ be an infinite set of propositional variables $p, q, \ldots$. With this set we associate two languages, $\mathscr{L}_0$ and $\mathscr{L}_{A1}$, s.th. $\mathscr{L}_0$ has the logical connectives $\neg, \vee, \wedge, \rightarrow$ defined in the traditional way. $\mathscr{L}_{A1}$ has the additional operator $\langle B\rangle A$. A formula C of $\mathscr{L}_{A1}$ is either:

1. A of $\mathscr{L}_0$ or
2. $\langle B\rangle A$, where $A, B \in \mathscr{L}_0$.

We call formulae of the form $\langle B\rangle A$ *strict* $\mathscr{L}_{A1}$-formulae. Formulae in the normal propositional $\mathscr{L}_0$ language can be deduced by the standard rules of the propositional calculus.

Remark 2.

$\Delta \vdash_{\mathscr{L}_0} A \Leftrightarrow \Delta \vdash_{CL} A$

The inference to an $\mathscr{L}_{A1}$ formula is conditional on the inferred formula not causing inconsistency.

On top of the basic propositional logic $\vdash_{A_0}$, we define two versions of abduction, one *consistent* ($\vdash_{A_C}$) and one *minimal* ($\vdash_{A_M}$). Both of those have the same basic structure. Firstly, we annotate the explanation A_i with a modal operator $\langle B\rangle$ saying that A_i is a possible explanation for B. This way we know why the formula entered the proof as well as avoiding that several different explanations, that are inconsistent with each other, produce an inconsistency within the database Δ. We will call this *careful proving*. On an abduction step where $\langle B\rangle A$ is added, we put $\Delta + A \nvdash \bot$ as a condition. This means that any proof line depending on $\langle B\rangle A$ only holds until $\neg A$ occurs in the proof. When $\neg A$ is deduced, $\langle B\rangle A$ is deleted, as well as all lines depending on it. We call this *provisional proving*.

Definition 3.

[Rule for Consistent Abduction]

$$\frac{\Delta \vdash_{A_C} A \rightarrow B, \Delta + A \nvdash_{A_C} \bot}{\Delta \vdash_{A_C} \langle B\rangle A}$$

The above rule captures the definition of consistent abduction as defined in the table in the introduction. We interpret the formula modally like a possibility operator, with the amendment that $\langle B\rangle$ means that there is a possible B-world, i.e. a world where B is true, in which A is true. Thus we label the world by B. Let w be a B-world, then we have $w \models A \rightarrow B$ and $w \models B$ and due to the condition, A deduces B consistently. Thus in this world, A is a possible consistent explanation for B.

We can define a further kind of abduction from the table in the introduction, *minimal abduction*.

[6] Cf. Batens et al. in (Batens 1989).

[7] We assume that our set is exhaustive. Within an abduction framework this is a common assumption when abduction is only concerned with explanation-selection.

[8] For abbreviation we use $\Delta + A$ for $\Delta \cup \{A\}$.

Definition 4.

[Rule for Minimal Abduction][9]

$$\frac{\Delta \vdash_{A_M} A \to B \qquad \forall C : \Delta \not\vdash_{A_M} C \to A \qquad \Delta + A \not\vdash_{A_M} \bot}{\Delta \vdash_{A_M} \langle B \rangle A}$$

We still have the condition of consistency, but on top of this we have the condition, that the formula A is not implied by any other formula in Δ, making sure that we find the explanation at the greatest depth. The interpretation of $\langle B \rangle A$ is the same as above.

We need one more rule for further deductions with abduction formulae. As this rule applies to $\vdash_{A_C}$ as well as $\vdash_{A_M}$ we omit the subscripts.

Definition 5.

[Propagation of $\langle B \rangle$]

$$\frac{\Delta \vdash_A \langle B \rangle A \qquad \Delta \vdash_A A \to C}{\Delta \vdash_A \langle B \rangle C}$$

Additionally to this we have reflexivity for $\mathscr{L}_{A1}$ formulae:

$$\frac{\forall \langle B \rangle A \in \Delta}{\Delta \vdash_A \langle B \rangle A}$$

Note that for each of the abduction logics the respective three rules are the *only* rules covering $\mathscr{L}_{A1}$-formulae. Especially, we do not have a rule for left monotonicity. Thus, when Δ is expanded, $\mathscr{L}_{A1}$-formulae, that were derivable before, might cease to be derivable, as the inference is only conditional. We show an example in which Δ is expanded with further facts so that earlier abductions are eliminated.

When faced with an abduction, we can have different policies. We might want to find all possible explanations at the start and then wait for elimination; alternatively we can adopt a *lazy abduction* policy, so that we only abduce one formula at the time and only let deletion of this abduction trigger further search for explanations. In the following example we adopt a lazy approach for simplicity of presentation.

Example

Sherlock Holmes and Dr Watson are called to Dartmoor; to Colonel Ross' training stables King's Pyland. His invaluable race horse Silver Blaze, first favourite for the Wessex Cup, was abducted a few nights before and the trainer, John Straker, was found dead on the moor. It is assumed that a stranger abducted the horse, the trainer went after the abductor and was killed in action. There are however other possible explanations for a run-away horse than a stranger. The door to the stable might have been left open and the horse ran away. Alternatively, not a stranger abducted the horse, but someone known in the stables. During the investigations Holmes finds out that the door was not left open. The puzzle is solved when Holmes finds out that the dog did not bark on the night in question and 'deduces', as Conan Doyle called it, that it must have been a familiar person abducting the horse as otherwise the dog would have barked. The solution is that John Straker took the horse to the moors to injure its ankle in order to rig the coming race. In this endeavour the horse kicked him to death and ran to the neighbouring farm, where the owner, as well keeping race horses, kept it to make sure that one of *their* horses wins the race. We formalise this example in the following initial database. Let $\Delta' = \Delta + (\mathsf{n} \wedge \mathsf{d\_o})$.

$\Delta = \{$nobody $\wedge$ door open $\to$ horse gone $((\mathsf{n} \wedge \mathsf{d\_o}) \to \mathsf{h})$,
somebody $\to$ horse gone $(\mathsf{sm} \to \mathsf{h})$,
friend $\to$ somebody $(\mathsf{fr} \to \mathsf{sm})$,
stranger $\to$ somebody $(\mathsf{st} \to \mathsf{sm})$,
friend $\to \neg$dog barks $(\mathsf{fr} \to \neg\mathsf{d\_b})$,
nobody $\to \neg$dog barks $(\mathsf{n} \to \neg\mathsf{d\_b})$,
stranger $\to$ dog barks $(\mathsf{st} \to \mathsf{d\_b})\}$

Observation: h

The observation h has a special information status as it is the observation we want to explain. Other information entering later in the example, like $\neg$d_o is processed as a simple expansion. We want to explain why the horse is gone but not why the door is not open.

1. $\Delta \vdash_{A_M} (\mathsf{n} \wedge \mathsf{d\_o}) \to \mathsf{h}$ $\quad \forall C : \Delta \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$ $\quad \Delta' \not\vdash_{A_M} \bot$
2. $\Delta \vdash_{A_M} \langle \mathsf{h} \rangle (\mathsf{n} \wedge \mathsf{d\_o})$

We now get the new information that the door was closed ($\neg$d_o).[10] Let $\Delta_1 = \Delta + \neg\mathsf{d\_o}$ and $\Delta_1' = \Delta + (\mathsf{n} \wedge \mathsf{d\_o}) + \neg\mathsf{d\_o}$.

1. $\Delta \vdash_{A_M} (\mathsf{n} \wedge \mathsf{d\_o}) \to \mathsf{h}$ $\quad \forall C : \Delta \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$ $\quad \Delta' \not\vdash_{A_M} \bot$
2. $\Delta_1 \vdash_{A_M} \langle \mathsf{h} \rangle (\mathsf{n} \wedge \mathsf{d\_o})$ $\quad \forall C : \Delta_1 \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$ $\quad \Delta_1' \vdash_{A_M} \bot$

At this point we have to delete the two lines and start anew. Let $\Delta_1'' = \Delta_1 + \mathsf{st}$.

1. ~~$\Delta \vdash_{A_M} (\mathsf{n} \wedge \mathsf{d\_o}) \to \mathsf{h}$~~ $\quad$ ~~$\forall C : \Delta \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\Delta \not\vdash_{A_M} \bot$~~
2. ~~$\Delta_1 \vdash_{A_M} \langle \mathsf{h} \rangle (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\forall C : \Delta_1' \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\Delta_1' \vdash_{A_M} \bot$~~
3. $\Delta_1 \vdash_{A_M} \mathsf{sm} \to \mathsf{h}$ $\quad \forall C : \Delta_1 \not\vdash_{A_M} C \to \mathsf{st}$ $\quad \Delta_1'' \not\vdash_{A_M} \bot$
4. $\quad \Delta_1 \vdash_{A_M} \mathsf{st} \to \mathsf{sm}$
5. $\Delta_1 \vdash_{A_M} \mathsf{st} \to \mathsf{h}$ $\quad \forall C : \Delta_1 \not\vdash_{A_M} C \to \mathsf{st}$ $\quad \Delta_1'' \not\vdash_{A_M} \bot$
6. $\Delta_1 \vdash_{A_M} \langle \mathsf{h} \rangle \mathsf{st}$

Now we get to know that the dog did not bark ($\neg$d_b). We have to check the previous conditions now for the new sets $\Delta_2 = \Delta_1 + \neg\mathsf{d\_b}$ and $\Delta_2' = \Delta_2 + \mathsf{st}$.

1. ~~$\Delta \vdash_{A_M} (\mathsf{n} \wedge \mathsf{d\_o}) \to \mathsf{h}$~~ $\quad$ ~~$\forall C : \Delta \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\Delta' \not\vdash_{A_M} \bot$~~
2. ~~$\Delta_1 \vdash_{A_M} \langle \mathsf{h} \rangle (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\forall C : \Delta_1 \not\vdash_{A_M} C \to (\mathsf{n} \wedge \mathsf{d\_o})$~~ $\quad$ ~~$\Delta_1' \vdash_{A_M} \bot$~~
3. $\Delta_1 \vdash_{A_M} \mathsf{sm} \to \mathsf{h}$ $\quad \forall C : \Delta_1 \not\vdash_{A_M} C \to \mathsf{sm}$ $\quad \Delta_1'' + \mathsf{sm} \not\vdash_{A_M} \bot$
4. $\quad \Delta_1 \vdash_{A_M} \mathsf{st} \to \mathsf{sm}$
5. $\Delta_1 \vdash_{A_M} \mathsf{st} \to \mathsf{h}$ $\quad \forall C : \Delta_1 \not\vdash_{A_M} C \to \mathsf{st}$ $\quad \Delta_1'' \not\vdash_{A_M} \bot$
6. $\Delta_2 \vdash_{A_M} \langle \mathsf{h} \rangle \mathsf{st}$ $\quad \forall C : \Delta_2 \not\vdash_{A_M} C \to \mathsf{st}$ $\quad \Delta_2' \vdash_{A_M} \bot$

Again we have some more deletion to do as another condition does not hold. We delete the lines that depend on the condition that st does not cause inconsistency, leaving only lines 3, 4 in our proof. Let $\Delta_2'' = \Delta_2 + \mathsf{fr}$.

[9] $\forall C$ should be restricted to $C \neq A$ to exclude tautologies. For efficiency we can constrain it to all C that appear in Δ.

[10] Note that the new information is processed differently from the abduced information. Abduced formulae only occur as expansions of Δ in the condition columns (Column 2&3) whereas real information is added in Column 1 and thus is a 'real' expansion to the belief base.

1. ~~$\Delta \vdash_{A_M}$ (n ∧ d_o) → h~~	~~$\forall C : \Delta \not\vdash_{A_M} C \rightarrow$ (n ∧ d_o)~~	~~$\Delta' \not\vdash_{A_M} \bot$~~
2. ~~$\Delta_1 \vdash_{A_M}$ ⟨h⟩(n ∧ d_o)~~	~~$\forall C : \Delta_1 \not\vdash_{A_M} C \rightarrow$ (n ∧ d_o)~~	~~$\Delta'_1 \vdash_{A_M} \bot$~~
3. $\Delta_1 \vdash_{A_M}$ sm → h	$\forall C : \Delta_1 \not\vdash_{A_M} C \rightarrow$ sm	$\Delta''_1 +$ sm $\not\vdash_{A_M} \bot$
4.	$\Delta_1 \vdash_{A_M}$ st → sm	
5. ~~$\Delta_1 \vdash_{A_M}$ st → h~~	~~$\forall C : \Delta_1 \not\vdash_{A_M} C \rightarrow$ st~~	~~$\Delta''_1 \not\vdash_{A_M} \bot$~~
6. ~~$\Delta_2 \vdash_{A_M}$ ⟨h⟩st~~	~~$\forall C : \Delta_2 \not\vdash_{A_M} C \rightarrow$ st~~	~~$\Delta'_2 \not\vdash_{A_M} \bot$~~
7. $\Delta_2 \vdash_{A_M}$ fr → h	$\forall C : \Delta_2 \not\vdash_{A_M} C \rightarrow$ fr	$\Delta''_2 \not\vdash_{A_M} \bot$
8. $\Delta_2 \vdash_{A_M}$ ⟨h⟩fr		

We now have an inference to 'friend' as an explanation for the observation that the horse is gone. For Δ_2 this is the only formula for which both conditions hold. In Δ we could have had 'nobody ∧ door open', 'stranger' or 'friend', (in Δ' still 'stranger' or 'friend').

Restricted Access Logics and Abduction

In this section we adapt the above proof theory based on classical logic to the setting, where the database Δ is inconsistent.

Paraconsistent logic is the naughty child standing in the corner of the logic class. It seems to propose that the world is not cut up into true and false. For some formulae it might be the case that the formula as well as its negation are true, or that a formula is true and false at the same time. Usually paraconsistent logics are formalised by restricting the inference rules. For example, the disjunctive syllogism is not a permitted rule of inference for most paraconsistent logics.

Another approach to paraconsistency is not to weaken the proof rules but to restrict access to data, so that in the same proof only consistent formulae may occur. This idea was first entertained by Rescher and Manor in (Rescher & Manor 1970). In (Gabbay & Hunter 1993) Gabbay and Hunter develop a concise proof theory to enable reasoning from inconsistent databases. Fully fledged propositional classical logic is at our disposal but rules can only be applied to consistent subsets of the original (inconsistent) database Δ. As we only want our abduction to be performed on the consistent part of the database we find this approach very fruitful and we discuss it in more detail.

To provide a framework where inconsistency can be controlled we label the formulae in our database. We have the set of natural numbers $\mathbb{N}$ such that $i \in \mathbb{N}$ is a label and, if A is a formula $i : A$ is a labelled formula. A labelled database is a database consisting of formulae each of which is labelled with a unique singleton set. We have all rules of propositional classical logic at our disposal but have to make sure that the sets they are applied to are consistent. To facilitate this a function $h(i)$ is defined on the labels as follows:

$$h(i) = 1 \text{ iff } \{A | j \subseteq i \text{ and } j : A \in \Delta\} \not\vdash \bot$$

The condition $h(i) = 1$ makes sure that no inconsistency was used in the proof, thus ensuring that the proof is performed from a consistent subset of Δ.

Definition 6.

[Labelled Rules of RAc]

$$\frac{i : A, j : B, h(i \cup j) = 1}{i \cup j : A \wedge B} \wedge I \qquad \frac{i : A \wedge B, h(i) = 1}{i : A} \wedge E$$

$$\frac{\begin{array}{c}\varnothing : A \\ \vdots \\ i : B, h(i) = 1\end{array}}{i : A \rightarrow B} \rightarrow I \qquad \frac{\begin{array}{c}\varnothing : \neg A \\ \vdots \\ i : \bot, h(i) = 1\end{array}}{i : A} RAA$$

$$\frac{i : A, j : A \rightarrow B, h(i \cup j) = 1}{i \cup j : B} \rightarrow E \qquad \frac{i : \bot, h(i) = 1}{i : A} EFQ$$

$$\frac{i : A, j : \neg A, h(i \cup j) = 1}{i \cup j : \bot} \neg E \qquad \frac{\begin{array}{c}\varnothing : A \\ \vdots \\ i : \bot, h(i) = 1\end{array}}{i : \neg A} \neg I$$

Definition 7.

$\Delta \vdash_c A$ iff $\exists i$ such that there is a proof of $i : A$ from Δ using the RAc proof rules.

This is the basic restricted consequence relation RAc defined in (Gabbay & Hunter 1993) and it will suffice for our purpose. We cite two examples from the original paper for illustration.

Let us have the database $\Delta = \{(1) : \neg a, (2) : \neg a \rightarrow \neg b, (3) : a\}$. The first deduction of $\neg b$ is admitted, the second is not:

1. $$\frac{(1) : \neg a, (2) : \neg a \rightarrow \neg b, h(1 \cup 2) = 1}{(1 \cup 2) : \neg b}$$

2. $$\frac{(1) : \neg a, (3) : a, h(1 \cup 3) \neq 1}{\dfrac{(1 \cup 3) : \bot, h(1 \cup 3) \neq 1}{(1 \cup 3) : \neg b}}$$

We can now define the abduction rules on top of RAc. The condition $h(i) \neq 1$ should be read as holding for all i.

Definition 8.

[RAc-Consistent and RAc-Minimal Abduction]

$$\frac{\begin{array}{l} i : A \rightarrow B, h(i) = 1 \\ j : A \rightarrow \neg B, h(j) \neq 1 \end{array}}{i' : \langle B \rangle A} \qquad \frac{\begin{array}{l} i : A \rightarrow B, h(i) = 1 \\ j : A \rightarrow \neg B, h(j) \neq 1 \\ k : C \rightarrow B, h(k) \neq 1 \end{array}}{i' : \langle B \rangle A}$$

$$\frac{k : \neg A, h(k) \neq 1}{i' : \langle B \rangle A} \qquad \frac{l : \neg A, h(l) \neq 1}{i' : \langle B \rangle A}$$

Definition 9.

[Propagation of $\langle B \rangle$]

$$\frac{\begin{array}{l} i' : \langle B \rangle A, h(i') = 1, \\ j : A \rightarrow C, h(j) = 1, \\ k : A \rightarrow \neg C, h(k) \neq 1 \end{array}}{i' \cup j : \langle B \rangle C} \text{ Prop}_P$$

The subscripts C and M are omitted as the rule holds for both. As we discussed before, we want abduction only to apply to the consistent core of Δ. This is expressed in the condition that $A \rightarrow \neg B$ is not deducible from any consistent subset.

Example Let us again take our Sherlock Holmes Story, however this time with an inconsistent initial database.

$$\Delta = \{(1) : \mathsf{st} \rightarrow \mathsf{h}, (2) : \mathsf{fr} \rightarrow \mathsf{h}, (3) : \mathsf{st} \rightarrow \mathsf{d\_b}, (4) : \mathsf{d\_b}, (5) : \mathsf{d\_b} \rightarrow \bot\};$$

We can now try to perform abduction in order to explain the horse being abducted, h.

1. $(1) : \mathsf{st} \rightarrow \mathsf{h} \quad (3 \cup 5) : \neg\mathsf{st} \quad 1 = h(1) = h(3 \cup 5)$

Thus the inference to 2. $(1') : \langle\mathsf{h}\rangle\mathsf{st}$ is blocked as $\neg\mathsf{st}$ is derivable from the consistent subset

$$\{(3) : \mathsf{st} \rightarrow \mathsf{d\_b}, (5) : \mathsf{d\_b} \rightarrow \bot\}.$$

That an abduction from an inconsistent set *fails* is the situation that we would have had in our normal logic as well. In Δ we have another potential explanation available for h. The following is a successful abduction of $\langle\mathsf{h}\rangle\mathsf{fr}$:

2. $(2) : \mathsf{fr} \rightarrow \mathsf{h} \quad (4 \cup 5) : \neg\mathsf{fr} \quad 1 = h(1) \neq h(4 \cup 5)$
3. $(2') : \langle\mathsf{h}\rangle\mathsf{fr}$

$\neg\mathsf{fr}$ can, in theory, be deduced from Δ by EFQ with the label $(4 \cup 5)$. However, $h(4 \cup 5) \neq 1$ and thus in our system the deduction fails and the abduction is allowed.

We now have a proof theory for abduction to be applied to consistent databases and a simple adaptation of this proof theory to inconsistent sets. Clearly, it is sufficient to have the RAc abduction proof theory as the consistent case is a limiting case of it. In the next section we describe semantics for these syntactic frameworks.

Semantics

In this section we introduce semantics for abduction. The process of abduction is expressed as a modal expansion of models, as can be seen in Figure 1.

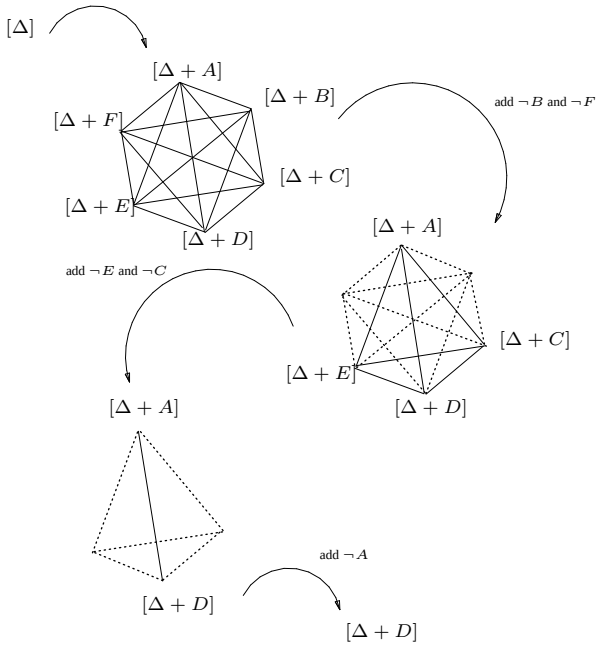


Figure 1: Abduction with Explanations A, B, C, D, E, F

The proof theories described in the last sections are essentially dynamic and the dynamic of this form of reasoning has to be mirrored in the semantics. Static semantic defines the *meaning* of a sentence as its *truth conditions*. In the research of applying logical semantics to natural language, a paradigm shift has occurred as described by Groenendijk et.al. in (Groenendijk, Stokhof, & Veltman 1996), away from defining the meaning of a sentence as its *truth conditional content* to meaning of a sentence as its *information change potential*.

In logic it is obviously not only the meaning of sentences that is defined as the conditions under which it is true, but as well the consequence relation is defined as that, i.e.

$$\Delta \models A \text{ iff } [\Delta] \subseteq [A]$$

meaning that if all the sentences in Δ are true, A must be true as well.

Abduction is the transference of this new definition of meaning as change to information state to a consequence relation. The following are the three consequence relations defined in Veltman's framework[11] for the operator *might* (M) which defines inference to compatible formulae.

Definition 10.

[Consequence Relations]

Let $\mathscr{L}$ be a set and consider a frame $\mathcal{F} = (\Sigma, ([p]_\mathcal{F})_{p \in \mathscr{L}})$. fix, rge and do stand for fixpoint, range and domain of a binary relation respectively. $\circ$ stands for relational composition.

1. Test Consequence:
   $\mathcal{F} \models_{tc} p_1 \ldots p_n \rightarrow q$ iff $fix([p_1]_\mathcal{F}) \cap \ldots \cap fix([p_n]_\mathcal{F}) \subseteq fix([q]_\mathcal{F})$
2. Ignorant Consequence:
   $\mathcal{F} \models_{ic} p_1 \ldots p_n \rightarrow q$ iff $(\top, t) \in ([p_1]_\mathcal{F}) \circ \ldots \circ ([p_n]_\mathcal{F})$ implies $(t, t) \in ([q]_\mathcal{F})$
3. Update Test Consequence:
   $\mathcal{F} \models_{utc} p_1 \ldots p_n \rightarrow q$ iff $rge([p_1]_\mathcal{F} \circ \ldots \circ [p_n]_\mathcal{F}) \subseteq fix([q]_\mathcal{F})$

The first clause is the same as the classical consequence relation, where any state that supports the premises supports the conclusion. $\models_{ic}$ and $\models_{utc}$ are more dynamic interpretations of consequence where $\models_{ic}$ is a special case of $\models_{utc}$ starting from a state of ignorance. Consequence relation 3. is the one we find most interesting. It says that a set of premises supports a conclusion (or an argument is *valid*) iff any state that can be obtained by addition of the premises supports the conclusion. This informal interpretation of consequence relation 3. is just what abduction means, that a state in which an explanation A is added supports the observation B. The only additional constraint is that the state in which A is added does not produce the empty set.

The facette of these semantics that is interesting for us is that the update is formulated within a modal setting. This modal formulation of $\mathsf{M}A$ as A *might hold* gives us the tools that we need for hypothetical reasoning, i.e. for the inference that A *might be the right explanation for* B. The

[11] Cf. (Veltman 1996) and (van der Does, Groeneveld, & Veltman 1997)

semantics introduced for *might* and adapted for abduction in this section equates satisfiability with acceptance in the sense that $\Delta \models A$, i.e. Δ satisfies A, iff $[\Delta + A] = [\Delta]$, i.e. A is already accepted in Δ.

However, apart from the similarities of this semantics to the abductive proof rule described in the section on abductive proof theory, these semantics do not capture fully what an *abductive* update of an information state involves. Although the MA models hypothetical addition of a formula to an information state, or database, abduction involves more than that. Abduction is a *specific* hypothetical update following from an observation B and the belief in a certain rule by which $A \rightarrow B$. Thus our semantics should model this specificity as well as the hypothetical character of the update.

In the proof theory we accomplished the specificity by labeling the $\langle B\rangle$-operator with the observation B. In relation to the proof theory described above, these semantics correspond to the proof rule that we can infer $\langle B\rangle A$ iff $\neg A$ is not derivable from the database, as $i \models \langle B\rangle A$ iff $i[A] \neq \varnothing$.

In the update semantics the semantic definition of a formula changed from the conditions under which the formula is true to what effect the addition of the formula has on a belief state on addition. We adopt this framework to abduction, modelling it closely on the *Update Test Consequence*.

Definition 11.

[Abductive Consequence]

Let Δ be a set and $[\Delta]$ the associated sets of models. fix and rge stand for fixpoint and range respectively. In abductive logic, a consequence is valid iff.

$$[\Delta] \models_{Abd} A \rightarrow B \text{ iff } \text{rge}\,[\Delta + A] \subseteq \text{fix}\,[\Delta + B]$$

We now prove soundness and completeness of the abduction rules of the consistent proof theory with respect to a special abductive update semantics.

The update semantics for might are defined as follows:

Definition 12.

[Information Structures, Models, Updates]

An *information structure* is a structure $\mathcal{I} := \langle I, ^c, \top, F_i\rangle$ consisting of a Boolean algebra $\langle I, ^c, \top\rangle$ with a family of operators $F_i : I^n \longrightarrow I$. A *model* for a set of propositional variables $\mathcal{P} = \{p, q, \ldots\}$ is an information structure $\mathcal{I} := \langle I, ^c, \top, [[p]]^{\mathcal{I}}\rangle_{p\in\mathcal{P}}$, where the update function $[[p]]^{\mathcal{I}} : I \longrightarrow I$ must satisfy:

1. $i[[p]] \subseteq i$ introspective
2. If $i \subseteq j$ then $i[[p]] \subseteq j[[p]]$ monotone
3. If $i \subseteq j[[p]]$ then $i \subseteq i[[p]]$ stable

With every formula A an update function $[A]^{\mathcal{I}} : I \longrightarrow I$ is given as follows:

$$\begin{array}{lllll}
\text{a.} & i[p] & = & i[[p]] \\
\text{b.} & i[\neg A] & = & i - i[A] \\
\text{c.} & i[A \wedge B] & = & i[A] \wedge i[B] \\
\text{d.} & i[\mathsf{M}A] & = & \begin{cases} i \text{ if } i[A] \neq \varnothing \\ \varnothing \text{ otherwise .} \end{cases}
\end{array}$$

Essentially, an information structure is a set of possible worlds where each world $w \in I$ is a function $w : \mathcal{P} \mapsto \{\top, \bot\}$ expressed by the valuation function $[[]]^{\mathcal{I}}$. On top of that we have the update function $[A]$ for each formula A recursively defined as above.

The foundation of the semantics for abduction is the same as above. We model abduction as an operation that allows inference to a formula A as a *possible explanation* for B ($\langle B\rangle A$) iff A is acceptable in a set Δ ($[\Delta + A] \neq \varnothing$) and there is a certain relation between A and B ($\text{rge}\,[\Delta + A] \subseteq \text{fix}\,[\Delta + B]$). To express this we replace clause d. by the following clauses for the modal operator in the two abductions.

Definition 13.

[Semantics for Abduction]

Let $[\Delta]$ be the set of models for Δ. $\Delta \models A$ iff $[\Delta + A] = [\Delta]$. Let satisfiability for consistent abduction $\models_{A_C}$ be defined as follows:

$$\textit{d-C. } [\Delta + \langle B\rangle A] = \begin{cases} [\Delta] \text{ if } [\Delta + A] \neq \varnothing \\ \& \text{ rge}\,[\Delta + A] \subseteq \text{ fix}\,[\Delta + B] \\ \varnothing \text{ otherwise .} \end{cases}$$

For minimal abduction $\models_{A_M}$ we substitute the semantic definition with

$$\textit{d-M. } [\Delta + \langle B\rangle A] = \begin{cases} [\Delta] \text{ if } [\Delta + A] \neq \varnothing \\ \&\forall C \in \Delta, \text{ rge}\,[\Delta + C] \not\subseteq \text{ fix}\,[\Delta + A] \\ \& \text{ rge}\,[\Delta + A] \subseteq \text{ fix}\,[\Delta + B] \\ \varnothing \text{ otherwise .} \end{cases}$$

Now we take the abductive logics defined in Definitions 3, 4 and 5 and show soundness and completeness of these logics with respect to the semantics defined above. Our first consideration is that we build a supra-classical logic. That means that we can use the completeness of the propositional part of our logic. We now have to show that the abductive addition is complete with respect to the specific update semantics in 13. However, as we are dealing with a very restricted logic, we show that any abductive inference can be reduced to a propositional inference. The strategy for doing this is as follows. We take any set Δ and reduce it to a set Δ_0 that does not contain $\mathscr{L}_{A1}$ formulae, in such a way that although $\Delta \neq \Delta_0$, the deductive closures are equivalent, i.e. $Cn(\Delta) = Cn(\Delta_0)$.

Definition 14.

[Reduction of Δ to Δ_0]

If Δ does not contain any $\mathscr{L}_{A1}$ formulae, $\Delta = \Delta_0$. Otherwise, enumerate all formulae of the form $\langle X\rangle Y \in \Delta$ from $1 - n$. Then the set Δ_{n-1} is defined recursively as follows:

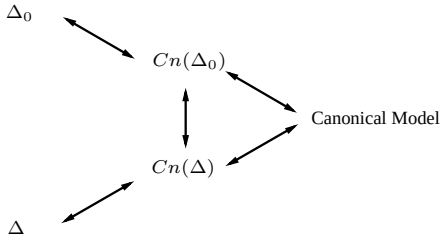


Figure 2: Reduction of $\mathscr{L}_{A1}$-set Δ to Δ_0

$$\Delta_{n-1} = \begin{cases} \Delta_n \backslash \langle X \rangle Y \text{ if } \exists Z \neq Y, \Delta_n \vdash_{A_C} \langle X \rangle Z \\ \qquad\qquad \& \Delta \vdash_{A_C} Z \rightarrow Y \Delta \\ (\Delta_n \backslash \langle X \rangle Y) \cup Y \rightarrow X \text{ otherwise .} \end{cases}$$

Δ_0 is a set without $\mathscr{L}_{A1}$ formulae.

This definition is important as we now can reduce any proof of a $\mathscr{L}_{A1}$-formula to an initial propositional proof of an implication. We need this preliminary Lemma for the proofs of soundness and completeness.

Lemma 15.

For any $\langle B \rangle A$, if $\Delta_0 \vdash_{Abd} \langle B \rangle A$, there $\exists X$ s.th. $\Delta_0 \vdash_{Abd} X \rightarrow B$ and $\Delta_0, X \nvdash_{Abd} \bot$.

Proof: Assume that $\Delta_0 \vdash_{Abd} \langle B \rangle A$. By definition, Δ_0 does not contain any $\mathscr{L}_{A1}$-formulae. We have to consider two cases:

1. $\Delta_0 \vdash_{Abd} A \rightarrow B$ and $\Delta_0 \nvdash_{Abd} \bot$. Then $X = A$.
2. $\Delta_0 \vdash_{Abd} \langle B \rangle C$ and $\Delta_0 \vdash_{Abd} C \rightarrow A$. For $\Delta_0 \vdash_{Abd} \langle B \rangle C$, either $\Delta_0 \vdash_{Abd} C \rightarrow B$ and $\Delta_0, C \nvdash_{Abd} \bot$, or propagation is applied again to some D s.th. $\Delta_0 \vdash_{Abd} \langle B \rangle D$ and $\Delta_0 \vdash_{Abd} D \rightarrow C$. At some point we end up with a Z which cannot be abduced by propagation anymore. ■

With this reduction we can proof soundness as follows.

Theorem 16.

[Soundness of Consistent Abduction]

$$\text{If } \Delta \vdash_{A_C} A \text{ then } \Delta \models_{A_C} A.$$

Proof: We only prove soundness of the abduction rule and the propagation rule. Assume $A = \langle B \rangle C$ and $\Delta \vdash_{A_C} \langle B \rangle C$. By our reduction we know that $\Delta_0 \vdash_{A_C} \langle B \rangle C$. As we do not have any $\mathscr{L}_{A1}$ formula in Δ_0, this can only be done by application of either the abduction rule or the propagation rule. Thus one of the following must hold:

1. Either $\Delta_0 \vdash_{A_C} C \rightarrow B$ and $\Delta_0 + C \nvdash_{Abd_C} \bot$. By soundness of propositional logic we have that $\Delta_0 \models_{Abd_C} C \rightarrow B$ and $\Delta_0 + C \not\models_{Abd_C} \bot$. By Definition 11 above we have that $\text{rng } [\Delta_0 + C] \subseteq \text{ fix } [\Delta_0 + B]$ and that $[\Delta_0 + C] \neq \varnothing$. By clause d-C. of Definition 13 we have that $[\Delta_0 + \langle B \rangle C] = [\Delta_0]$, thus $[\Delta_0] \models_{Abd_C} \langle B \rangle C$ and finally $[\Delta] \models_{Abd_C} \langle B \rangle C$.
2. Alternatively, $\Delta_0 \vdash_{A_C} D \rightarrow C$ and $\Delta_0 \vdash_{Abd_C} \langle B \rangle D$ for some D. Let D be the X of Lemma 15. Then $\Delta_0 \models_{Abd_C} D \rightarrow C$ and $\Delta_0 \models_{Abd_C} D \rightarrow B$ and $[\Delta_0 + D] \neq \varnothing$ by definition of Δ_0 together with Lemma 15. That means $\text{rng } [\Delta_0 + D] \subseteq \text{ fix } [\Delta_0 + B]$, $\text{rng } [\Delta_0 + D] \subseteq \text{ fix } [\Delta_0 + C]$ and $[\Delta_0 + D] \neq \varnothing$. Thus there exists a world $i \in [\Delta]$ s.th. $i[D] \neq \varnothing$. As it holds for all $j \in [\Delta_0 + D]$ that $j \subseteq \text{ fix } [\Delta_0 + C]$ and $j \subseteq \text{ fix } [\Delta_0 + B]$, this holds in particular for i. Hence $\exists i | i \neq \varnothing$ and $i[C] \subseteq i[B]$. Thus $[\Delta_0 + \langle B \rangle C] = [\Delta_0]$ and as $[\Delta_0] = [\Delta]$ we have $\Delta \models_{A_C} \langle B \rangle C$. ■

Theorem 17.

[Soundness of Minimal Abduction]

$$\text{If } \Delta \vdash_{A_M} A \text{ then } \Delta \models_{A_M} A.$$

Proof: Similar to above just with the extra condition that if $A = \langle B \rangle C$, $\forall D, i[D] \not\subseteq i[C]$. From $\Delta_0 \vdash \langle B \rangle C$ we know that $\Delta_0 \nvdash_{A_M} D \rightarrow C$ for any D. Thus the semantic condition holds and $\Delta \models_{A_M} \langle B \rangle C$. ■

We now prove completeness of consistent, minimal abduction respectively with respect to the semantics described above. Then the canonical model of them will be the same and due to completeness of $\mathscr{L}_0$ the result holds as well for $\mathscr{L}_{A1}$, cf. Figure 2.

Theorem 18.

[Completeness of Consistent Abduction]

$$\text{If } \Delta \models_{A_C} A \text{ then } \Delta \vdash_{A_C} A.$$

Proof: We know that if A is a $\mathscr{L}_0$ formula, the above holds as we simply have propositional logic. So let us assume that $A = \langle B \rangle C$ and $\Delta \nvdash_{A_C} \langle B \rangle C$. We reduce Δ to Δ_0 according to Definition 14. We have $\Delta_0 \nvdash_{A_C} \langle B \rangle C$. By the definition of Δ_0 we know that

$$\Delta_0 \nvdash_{A_C} C \rightarrow B \text{ or } \Delta_0 + C \vdash_{A_C} \bot.$$

By completeness of propositional logic we have that

$$\Delta_0 \not\models_{A_C} C \rightarrow B \text{ or } \Delta_0 + C \models_{A_C} \bot.$$

By Definition 11 we have that

$$\text{rng } [\Delta_0 + C] \not\subseteq \text{ fix } [\Delta_0 + B] \text{ or } [\Delta_0 + C] = \varnothing.$$

By Definition 13 d-C. this means that $[\Delta_0 + \langle B \rangle C] \neq [\Delta_0]$ unless $[\Delta_0] = \varnothing$. This however is impossible as $\Delta_0 \nvdash_{A_C} C \rightarrow B$. As $[\Delta_0] = [\Delta]$, $\Delta \not\models_{A_C} \langle B \rangle C$. ■

Theorem 19.

[Completeness of Minimal Abduction]

$$\text{If } \Delta \models_{A_M} A \text{ then } \Delta \vdash_{A_M} A.$$

Proof: The strategy of this proof is the same as before. We just have to adapt the semantic definition for minimal abduction. We know that we can go over from $\Delta \nvdash_{A_M} \langle B \rangle C$ we can go over to $\Delta_0 \nvdash_{A_M} \langle B \rangle C$. By completeness of propositional logic we have

$$\exists D \neq C | \Delta_0 \vdash_{A_M} D \rightarrow C \text{ or } \Delta_0 \nvdash_{A_M} C \rightarrow B \text{ or } \Delta_0 + C \vdash_{A_M} \bot.$$

By completeness of propositional logic we have that

$$\exists D \neq C | \Delta_0 \models_{A_M} D \rightarrow C \text{ or}$$
$$\Delta_0 \not\models_{A_M} C \rightarrow B \text{ or}$$
$$\Delta_0 + C \models_{A_M} \bot.$$

By Definition 11 we have that

$$\exists D | \text{ rng } [\Delta_0 + D] \subseteq \text{ fix } [\Delta_0 + C] \text{ or}$$
$$\text{rng } [\Delta_0 + C] \not\subset \text{ fix } [\Delta_0 + B] \text{ or}$$
$$[\Delta_0 + C] = \varnothing.$$

By Definition 13 d-M. this means that $[\Delta_0 + \langle B \rangle C] \neq [\Delta_0]$ unless $[\Delta_0] = \varnothing$. This however is impossible as $\Delta_0 \not\vdash_{A_M} C \rightarrow B$. As $[\Delta_0] = [\Delta]$, $\Delta \not\models_{A_M} \langle B \rangle C$. ■

We now have to refine the update semantics for the inconsistent case, as for an inconsistent Δ, $[\Delta] = \varnothing$ thus trivialising the definition of

$$\Delta \models_P A \text{ iff } [\Delta + A] = [\Delta].$$

So we have to think about how we can still have models on which we perform the abduction. In the normal abduction the models give a tree structure where the models of the hypothetical explanations are the children of the root models of Δ, (cf. Figure 3.a). For an inconsistent Δ we define the models to be the models of consistent subsets Δ_i^C of Δ. Let us call this set of sets of models $[\Delta]$. Classically, $[\Delta] = \varnothing$ if Δ is inconsistent, but for our paraconsistent logic we define $[\Delta]$ as just the tree structure in Figure 3.b, expressed by

$$[\Delta] = \bigcup_{i=1}^{n} [\Delta_i^C]$$

Thus we interpret an inconsistency similar to a modal interpretation. In inconsistent information state an agent does not know which model corresponds to the real world, but it is either one of the models that satisfy the negated formula, say $\neg A$ or A.

On top of these consistent subset models we define abduction as shown in Figure 3.c. With an abduction we are adding another level of hypothetical reasoning to the model. The agent with an inconsistent information state does not know which consistent subset captures the facts and on top of all these possible consistent subsets we have the possible abductions.

Now we know the general structure of the models we are working with and have to adapt the updated semantics, that served us so well for the consistent case, to the inconsistent case. We have to define how an update with a formula A changes the models of $[\Delta]$ and under which conditions $[\Delta + A] = [\Delta]$. The second part is easily done as shown in the following definition.

Definition 20.

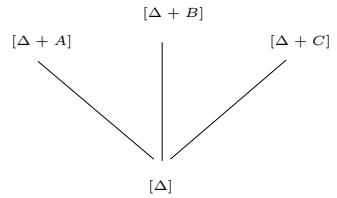
1. For $A \in \mathscr{L}_0$ we define $\Delta \models_{\mathcal{P}} A$ as

$$[\Delta + A] = [\Delta] \text{ iff } \exists \Delta^C : [\Delta^C + A] = [\Delta^C]$$

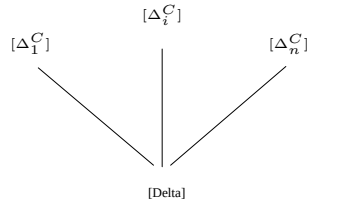
2. For strict $\mathscr{L}_{A1}$ formulae we define $\Delta \models_{\mathcal{P}} \langle X \rangle Y$ as

$$[\Delta + \langle B \rangle A] = [\Delta] \text{ iff } \forall \Delta^C : [\Delta^C + \langle B \rangle A] = [\Delta^C]$$

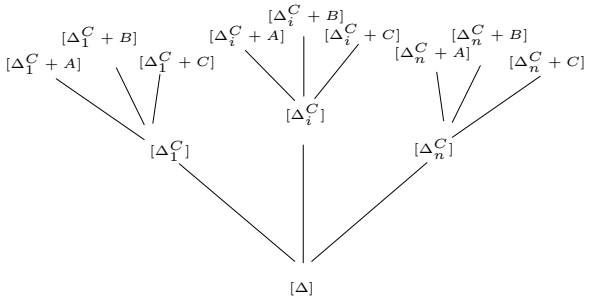
where $[\Delta^C + \langle B \rangle A] = [\Delta^C]$ is defined exactly as in Definition 13 depending on whether we want to model a *consistent* or *minimal* abduction.



a) Consistent Δ with explanations A, B, C.



b) Model of an Inconsistent Delta.



c) Δ with explanations A, B, C.

Figure 3: Consistent and Inconsistent Models

Condition 1. means that there must be at least one Δ^C where A is true in all $w \in [\Delta^C]$. Condition 2. is a lot stronger, i.e. it has to hold for all $[\Delta^C]$ that there exists a w s.th. $w[\langle B \rangle A] = w$. When reasoning with inconsistent information we cannot make the same strong conditions for truth as we apply to classical logic, hence in Condition 1. we only demand that there exists a maximal consistent subset that satisfies A. However, when we get into the realms of hypothetical reasoning, i.e. abduction, we want the constraints as strong as we can make them. Condition 2. means that we can only abduce A for some B, if $A \rightarrow B$ is always true, $\neg A$ is never true, or expressed differently, if $A \rightarrow B$ is true in the consistent core of Δ and $\neg A$ is false in the consistent core.

Defining how a model changes when new information arrives is more difficult. In Definition 12, the update of an information state i with an atom is defined as

$$i[p] = i[[p]]$$

thus returning exactly those worlds in which p is true. This is clearly not the definition we want as it might be the case that $i[\neg p] = i$ and by addition of p, we would delete viable

worlds. However, we lifted the inconsistency by introducing a layer of consistent models into the tree structure. This has to be reflected in the semantic definition. Thus, for all $[\Delta^C]$ which do not satisfy $\neg p$ we want as a return $[\Delta^C + p]$ just as defined in Definition 12. For the $[\Delta^C]$ that satisfy $\neg p$ however, we do not want change on addition of p to Δ. Thus we arrive at the following definition.

Definition 21.

Let A be a $\mathscr{L}_0$-formula.

$$[\Delta + A] = \bigcup_{i=1}^{n}([\Delta_i^C + A] | [\Delta_i^C + A] \neq \varnothing, [\Delta_j^C] | [\Delta_j^C + A] = \varnothing)$$

For $\mathscr{L}_{A1}$-formulae we have the following:

$$[\Delta + \langle X \rangle Y] = \begin{cases} \bigcup_{i=1}^{n} [\Delta_i^C] \text{ if } \forall \Delta^C : [\Delta^C + Y] \neq \varnothing \\ \varnothing \text{ otherwise .} \end{cases}$$

Obviously, if A is in all consistent subsets, then $[\Delta + A] = [\Delta]$. Thus we have the logic we described in the beginning as a limiting case of RAc-abduction.

Conclusion and Future Work

We introduced a propositional proof theory for abduction which captures the important dynamic features of explanatory reasoning. In addition we provided an adaptation of the proof theory to inconsistent databases. This work puts abduction into an new setting altogether. First it is not usual to define abduction as a proof theory. Secondly, to define it modally we express the hypothetical character of abduction. Thirdly, we showed a way to broaden the application of abduction to inconsistent databases by modifying the restrictive overall consistency condition usually used to constrain abductive inference. This feature of abduction is not only formally interesting but has implications for the implementability of an abduction framework. When an abductive proof theory is to be applied to a given database, there is no intrinsic reason, why the database should be consistent. In case it is inconsistent, no abduction that is constrained by consistency could ever be performed. It is however not workable to just give up consistency. Our proposal ensures that abductions on inconsistent sets can be performed without degenerating into triviality.

We as well gave semantics for theses two proof theories. For the normal abductive proof theory we proved soundness and completeness. The proof of this for RAc-abduction is future work. However, the proofs of soundness and completeness for para-consistent as well as para-minimal abduction will work in a similar way as the soundness and completeness of the normal abductive proof theory.

Additional future work is to make the proof theories goal directed. This is again done with an eye on implementation. For the normal proof theory this has been achieved in (Elsenbroich 2006) but for the paraconsistent adaptation this is outstanding work.

References

Alchourrón, C. A., and Makinson, D. 1982. The logic of theory change: Contraction functions and their associated revision functions. Theoria.

Alchourrón, C. A.; Gärdenfors, P.; and Makinson, D. 1985. The logic of theory change: Partial meet contraction and revision functions. The Journal of Symbolic Logic 50(2).

Aliseda-Llera, A. 1997. Seeking Explanations: Abduction in Logic, Philosophy of Science and Artificial Intelligence. Ph.D. Dissertation, ILLC University of Amsterdam.

Batens, D. 1989. Dynamic dialectical logics. In Priest, G., ed., Paraconsistent Logic. Essays on the Inconsistent. Munchen, Philosophia Verlag. 187–217.

Dias, W., and Wassermann, R. 2001. Abductive expansion of belief bases. In Proceedings of the IJCAI Workshop on Abductive Reasoning.

Elsenbroich, C. 2006. Instinct for Detection. Ph.D. Dissertation, King's College London.

Gabbay, D., and Hunter, A. 1993. Restricted access logics for inconsistent information. In Clarke, M.; Kruse, R.; and Moral, S., eds., Lecture Notes in Computer Science, volume 747 of ECSQARU'93. Springer. 137–144.

Groenendijk, J.; Stokhof, M.; and Veltman, F. 1996. Coreference and modality. In Lappin, S., ed., Handbook of Contemporary Semantic Theory. Oxford: Blackwell. 179–216.

Lobo, J., and Uzcátegui, C. 1996a. Abduction and change. In Procceeding of the Non-Monotonic Reasoning Workshop.

Lobo, J., and Uzcátegui, C. 1996b. Abductive change operators. Fundamenta Informaticae 27(4):385–411.

Lobo, J., and Uzcátegui, C. 1997. Abductive consequence relations. Artificial Intelligence 89:149–171.

Pagnucco, M.; A.C.Nayak; and N.Y.Foo. 1995. Abductive reasoning, belief expansion and nonmonotonic consequence. In Proceedings of the ICLP-95 Joint Workshop on Deductive Databases and Logic Programming and Abduction in Deductive Databases and Knowledge-based Systems.

Pagnucco, M. 1996. The Role of Abductive Reasoning Within the Process of Belief Revision. Ph.D. Dissertation, University of Sydney.

Rescher, N., and Manor, R. 1970. On inference from inconsistent premises. Theory and Decision 1:179–219.

van der Does, J.; Groeneveld, W.; and Veltman, F. 1997. An update on ??might?? Journal of Logic, Language and Information 6(4):361–380.

Veltman, F. 1996. Defaults in update semantics. Journal of Philosophical Logic (25):221–261.

6.8 Rethinking Semantics of Dynamic Logic Programming

Rethinking Semantics of Dynamic Logic Programming

Ján Šefránek
Institute of Informatics, Comenius University Bratislava, Slovakia, sefranek@fmph.uniba.sk

Abstract

A dynamic logic program represents an evolving knowledge base. Research in the field is dominated by the causal rejection principle: the conflicting rule from the less preferred program is rejected in the case of conflict. Some drawbacks of the causal rejection principle (irrelevant updates, inconsistencies which cannot be solved according to the causal rejection principle, disagreement with other fields relevant for updates of nonmonotonic knowledge bases such as belief revision or preferences handling) are identified and discussed in the paper. The discussion of those drawbacks pointed out the role of assumptions and dependencies on assumptions for a careful attitude to the topic. A solution based on a dependency framework is presented and evaluated in the paper.

Keywords: multidimensional dynamic logic programming, nonmonotonic knowledge base, stable model semantics, belief revision, update, preference, dependency framework.

Introduction

Background. Multidimensional dynamic logic programming (MDyLP) (Alferes et al. 1998; Leite et al. 2001; Leite 2003; Alferes et al.2005; Banti et al. 2005) contributed to logic-based knowledge representation research by focusing on dynamic aspects of knowledge. MDyLP can be considered as a formal model of (evolving) nonmonotonic knowledge bases (NMKB). Evolution of (incomplete) knowledge is a key to understanding of nonmonotonic reasoning. Therefore, a foundational research of the framework proposed by MDyLP is of interest also for other approaches to non-monotonic reasoning and knowledge representation.

Problem. However, there are some drawbacks of the approach that didn't attract a sufficient attention until now. They are connected to the causal rejection principle (CRP),[1] which dominates research in the field. We will identify and discuss

- irrelevant updates,
- inconsistencies which cannot be solved according to the causal rejection principle,
- disagreement with other approaches relevant for updates of NMKB (f.ex. with research in the fields of belief revision or preferences handling, see (Gärdenfors and Rott 1995; Delgrande et al. 2003) and many others).

Proposed solution. Non-monotonic (defeasible, default) assumptions play a crucial role in non-monotonic reasoning. Our analysis of drawbacks of CRP leads to an opinion that the attention should be moved from conflicts of rules to conflicts involving assumptions and dependencies on assumptions. A dependency framework is proposed in the paper. Rejection or insertion of rules is avoided; the framework is aiming at a coherent view on a (possibly incoherent) MDyLP[2] by *ignoring* some dependencies. Our approach is close to the spirit of answer set programming (thanks to the stress on sets of assumptions, see ([Konczak et al. 2004; Novák draft)).

The presented dependency framework evolved from our Kripkean semantics of MDyLP (Šefránek 2000; Šefránek 2004). It can be said that the Kripkean semantics served as a scaffold for the proposal of the dependency framework, which is simpler, more general, has better intuitive and computational properties. We believe that our approach[3] contributes to a shift from a discussion of examples and counterexamples to a principle-based approach (see also (Alferes et al.2005; Banti et al. 2005)) to the updates of NMKB topic.

Contributions. Main contributions of the paper are as follows:

- some drawbacks of MDyLP, which did not attract a sufficient attention until now are discussed,
- a dependency framework is proposed in which those drawbacks are overcome,
- a method of constructing a coherent semantic view on a set of dependencies is presented,

[1] If there is a conflict between the heads of rules then reject the less preferred rule.

[2] We use the shorthand MDyLP both for MDyL- programs and programming. It is hoped that this ambivalence do not cause problems.

[3] For a preliminary report see (Hpomola et al. 2005).

- properties of the semantics based on dependencies on assumptions are discussed.

Roadmap. First basic definitions relevant for understanding MDyLP are recapped. Then the dependency framework is introduced. After that we analyze solutions of conflicts between dependencies. Postulates specifying solutions of conflicts are introduced. Next, drawbacks caused by the principle of causal rejection are analyzed. Semantics of logic program updates based on the dependency framework is described. Finally, our approach is evaluated, contributions are summarized and open problems characterized.

Multidimensional dynamic logic programming

Let $\mathcal{A}$ be a set of atoms. The set of *literals* is defined as $Lit = \mathcal{A} \cup \{not\ A : A \in \mathcal{A}\}$. Literals of the form $not\ A$, where $A \in \mathcal{A}$ are called *subjective* (not is intended as default negation). Notation: $Subj = \{not\ A \mid A \in \mathcal{A}\}$. A convention: $not\ not\ A = A$. If X is a set of literals then $not\ X = \{not\ L \mid L \in X\}$.

A *rule* is each expression of the form $L \leftarrow L_1, \ldots, L_k$, where $k \geq 0$, L, L_i are literals. If r is a rule of the form as above, then L is denoted by $head(r)$ and $\{L_1, \ldots, L_k\}$ by $body(r)$. A finite set of rules is called *generalized logic program* (program hereafter).

The set of *conflicting literals* is defined as $CON = \{(L_1, L_2) \mid L_1 = not\ L_2\}$. Two rules r_1, r_2 are called conflicting, if $head(r_1)$ and $head(r_2)$ are conflicting literals. Notation: $r_1 \bowtie r_2$. A set of literals S is *consistent* if it does not contain a pair of conflicting literals, i.e. $(S \times S) \cap CON = \emptyset$. An *interpretation* is a consistent set of literals. A *total* interpretation is an interpretation I such that for each atom A either $A \in I$ or $not\ A \in I$. Let I be an interpretation. Then $I^- = I \cap Subj$.

A literal is *satisfied* in an interpretation I iff $L \in I$. A set of literals S is satisfied in I iff $S \subseteq I$.

The basic semantic concepts introduced by (Alferes et al. 1998; Leite et al. 2001; Leite 2003; Alferes et al.2005) are recapped below.

Definition 1 ((Alferes et al. 1998)) A total interpretation S is a *stable model* of a program P iff

$$S = least(P \cup S^-),$$

where $P \cup S^-$ is considered as a Horn theory and $least(P \cup S^-)$ is the least model of the theory. A program is *coherent* iff it has a stable model. □

Definition 2 ((Leite et al. 2001)) A *multidimensional dynamic logic program* (also *multiprogram* hereafter) is a pair $\mathcal{P} = (\Pi, G)$, where $G = (V, E)$ is an acyclic digraph, $|V| \geq 2$, and $\Pi = \{P_i : i \in V\}$ is a set of (generalized logic) programs.

We denote by $i \prec j$ that there is a path from i to j and $i \preceq j$ means that $i \prec j$ or $i = j$. If i and j are incomparable w.r.t. $\preceq$, it is denoted by $\|$. If $i \prec j$, we say that P_j is *more preferred* than P_i (we denote it by $P_i \prec P_j$ with a little abuse of $\prec$). □

If G is a path, we speak about *dynamic* logic program. If $\mathcal{P}$ is a multiprogram then $\mathcal{A}$ is the set of all atoms occurring in $\bigcup_{j \in V} P_j$ and for each $P_i \in \Pi$ an interpretation of P_i is a consistent subset of $\mathcal{A} \cup not\ \mathcal{A}$.

Definition 3 (Dynamic stable model, (Leite et al. 2001)) Let $\mathcal{P}$ be a multiprogram. A total interpretation M is called *dynamic stable model* of $\mathcal{P}$ iff

$$M = least(\bigcup_{i \in V} P_i \setminus Rej(\mathcal{P}, M)) \cup Def(\mathcal{P}, M)), \qquad (1)$$

where $Rej(\mathcal{P}, M) = \{r \in P_i \mid \exists r' \in P_j\ (i \prec j, r \bowtie r', M \models body(r'))\}$ and $Def(\mathcal{P}, M) = \{not\ A \mid \neg\exists r \in \bigcup_{i \in V} P_i\ (A = head(r), M \models body(r))\}$. □

Refined dynamic stable model is defined in (Alferes et al.2005) similarly, with only a little difference – condition $i \preceq j$ is used in the definition of rejected rules instead of $i \prec j$. We will use for that modified concept notation $Rej^R(\mathcal{P}, M)$). The set of all refined dynamic stable models of $\mathcal{P}$ is denoted by $RDSM(\mathcal{P})$. Troubles with tautological and cyclic updates of dynamic logic programs are overcome in refined semantics. However, the refined semantics for the general case of multiprograms is not known. The well supported semantics of multiprograms is defined in (Banti et al. 2005), in order to improve the behaviour of semantics based on CRP. The well supported semantics for MDyLP coincides with the refined one on dynamic logic programs.

We will use refined semantics in the analysis of examples, which contain elementary dynamic logic programs of the form $\langle P, U \rangle$, where $P \prec U$. P can be viewed as an original program and U as an updating program. We have chosen the most simple case in order to be comprehensible, but the analysis of examples is relevant for CRP in general.

Dependency framework

A specification of a semantics of multiprograms in terms of dependencies and assumptions is presented in this paper. Our approach is aiming at overcoming some drawbacks of semantics based on the causal rejection principle. The basic features of proposed dependency framework are described in this section.[4]

Two notions of dependency relation are going to be defined. First, a more general one and second, a dependency relation generated by a program.

Definition 4 (Dependency relation) A *dependency relation* is a set of pairs $\{(L, W) \mid L \in Lit, W \subseteq Lit, L \notin W\}$.

A literal L *depends* on a set of literals W, $L \notin W$, *with respect to a program* P ($L \ll_P W$) iff there is a sequence of rules $\langle r_1, \ldots, r_k \rangle$, $k \geq 1$, and

- $head(r_k) = L$,

[4] We could choose between two policies. The first one begins with a motivation and then provides a formal framework. The second one, to the contrary, uses the formal framework for an analysis of motivating examples. We prefer the second one in this paper. Motivation for our framework is detailed later.

- $W \models body(r_1)$,
- for each i, $1 \leq i < k$, $W \cup \{head(r_1), \ldots, head(r_i)\} \models body(r_{i+1})$.

It is said that the dependency relation $\ll_P$ is *generated* by the program P. □

Notice that a literal cannot depend on itself (also in a context of other literals). The dependency relation $\ll_P$ cannot be identified with derivability from P. If $P = \{a \leftarrow b\}$ then $a \ll_P \{b\}$, but a is not derivable from P.

Definition 5 (Closure property) A *closure operator Cl* assigns the set of all pairs $\{(L, W) \mid L \ll W \vee (\exists U\ (L \ll U \wedge \forall L' \in U \setminus W\ (L' \ll W)))\}$ to a dependency relation $\ll$.

A dependency relation $\ll$ has the closure property iff $\mathrm{Cl}(\ll) = \ll$. □

Notice that $\ll_P$ has the closure property.

Proposition 6 *Let P be a program. Then* $\mathrm{Cl}(\ll_P) = \ll_P$.

Proof sketch: Suppose that $L \ll_P U, \forall L' \in U \setminus W\ L' \ll_P W$. Consider a sequence of rules, satisfying the conditions of Definition 4 such that each $L' \in U \setminus W$ is derived from W. Concatenate a sequence deriving L from U. We have proved $L \ll_P W$, □

Example 7 Let P be

$$\begin{array}{rcl} \{a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \\ c & \leftarrow & a \\ d & \leftarrow & b\} \end{array}$$

It holds that $a \ll_P \{not\ b\}$, $b \ll_P \{not\ a\}$, $c \ll_P \{a\}$, $d \ll_P \{b\}$, $c \ll_P \{not\ b\}$, $d \ll_P \{not\ a\}$, but also $c \ll_P \{not\ b, not\ d\}$, $d \ll_P \{not\ a, not\ c\}$ or $d \ll_P \{not\ a, not\ c, not\ d\}$ etc. We can see that some dependencies of L on W are of crucial interest, namely those, where $W \subseteq Subj$ and W generates (or contributes to a generation) of a stable model. □

We have seen in Example 7 that dependencies on subjective literals are crucial from the viewpoint of stable semantics. Therefore the role of (default) assumptions is emphasized.

Definition 8 (SSOA, TSSOA) $Ay \subseteq Subj$ is called a *sound set of assumptions* (SSOA) with respect to the dependency relation $\ll$ iff the set

$$Cn_{\ll}(Ay) = \{L \in Lit \mid L \ll Ay\} \cup Ay$$

is non-empty and consistent.

It is said that Ay, a SSOA, is *total* (TSSOA) iff for each $A \in \mathcal{A}$ holds either $A \in Cn_{\ll}(Ay)$ or $not\ A \in Cn_{\ll}(Ay)$. □

Example 9 Let $\ll$ be $\{(a, \{not\ c\}), (not\ a, \{not\ d\})\}$. There is no TSSOA w.r.t. $\ll$. Both $\{not\ c\}$ and $\{not\ d\}$ are SSOAs w.r.t. $\ll$.

An important remark: updates of conflicting multiprograms can be viewed as a construction of TSSOAs from SSOAs while ignoring some dependencies. □

Next theorem shows that the dependency framework is relevant from the stable (answer set) semantics point of view.

Theorem 10 *X is a TSSOA w.r.t. $\ll_P$ iff $Cn_{\ll_P}(X)$ is a stable model of P.*

Let S be a stable model of P. Then there is $X \subseteq Subj$, a TSSOA w.r..t. $\ll_P$ s.t. $S = Cn_{\ll_P}(X)$. □

Semantics based on assumptions and dependencies. Consider two mappings Σ, Σ'. Let Σ assigns to each program P the set of all its stable models. Let Σ' assigns to each program P the set of all TSSOAs w.r.t. $\ll_P$. We have seen in Theorem 10 that (the semantics characterized by) Σ is equivalent to (the semantics characterized by) Σ'. So, we can speak about a semantics based on assumptions and dependencies.

Dependencies in a multiprogram. We intend to use our framework for handling conflicting dependencies in a multiprogram. Note that dependencies in a multiprogram are well defined:

Proposition 11 *Let $\mathcal{P}$ be a multiprogram. Then $\ll_{\bigcup_{i \in V} P_i}$ is well defined. It holds*

$$\bigcup_{i \in V} \ll_{P_i} \subseteq \ll_{\bigcup_{i \in V} P_i},$$

but the converse inclusion does not hold. □

We are going to introduce an approach (and some concepts) which enables to handle conflicts involving dependencies and assumptions.

Conflicts, solutions, postulates

There are essentially two possible sources of incoherence in a (multi)program $\mathcal{P}$:

1. two conflicting literals depend on a set of literals;
2. a literal L depends on a set of literals W and $not\ L \in W$; dependencies on subjective literals are crucial in our framework, so we are focused only on the case that L is an atom.

Definition 12 Let $\ll$ be a dependency relation. It is said that $\ll$ *contains a conflict* C (where $C \subseteq \ll$) iff for some $A \in \mathcal{A}$ is $C = \{(A, Y), (not\ A, Y) \mid Y \subseteq Subj\}$ or $C = \{(A, Y) \mid Y \subseteq Subj, not\ A \in Y\}$. □

We are interested in solving conflicts between dependencies generated by multiprograms. It is assumed here that the preference relation on programs is preserved also for corresponding dependency relations, i.e. if P_i is more preferred than P_j then $\ll_{P_i}$ is more preferred than $\ll_{P_j}$.

Definition 13 Let $\mathcal{P}$ be a multiprogram and $C \subseteq \ll_{\bigcup_{i \in V}}$ be a conflict.

It is said that a set of dependencies D is a *solution* of the conflict C iff $C \not\subseteq Cl(\bigcup_{i \in V} \ll_{P_i} \setminus D)$. D is called *minimal* iff there is no proper subset of D which is a solution of C.

Let D and D' be minimal solutions of C. It is said that D' is *more suitable* than D iff there is an injection $\kappa : D' \longrightarrow D$ such that $\forall d \in D'$ $((d \in \ll_{P_j} \wedge \kappa(d) \in \ll_{P_i}) \Rightarrow j \preceq i)$. If the cardinality of D and D' is the same then for at least one $d \in D'$ holds $j \prec i$.

A minimal solution D of a conflict C is called *good solution* iff there is no more suitable solution of C. □

Some comments to the definition: A solution of a conflict is focused on dependencies generated by a single program. Only elementary pieces of a chain of dependencies (dependencies from some $\ll_{P_i}$, $i \in V$) are ignored.[5] It is more suitable to ignore less preferred dependencies. Good solutions consist of minimally preferred dependencies and the principle of minimal change is obeyed. Note that in general there are different good solutions of a conflict. Another approach to alternative solutions of a conflict is presented in (Šefránek 2006a).

Our attention is focused also on conflicting assumptions. First a notion of falsified assumptions is defined. It is used in the analysis of Example 18.

Definition 14 An assumption $not\ A$, where $A \in \mathcal{A}$, is *falsified* in a dependency relation $\ll$ iff $A \ll \emptyset$, $not\ A \not\ll \emptyset$ and $\emptyset$ is a SSOA w.r.t. $\ll$.

A set of assumptions $Ay \subseteq Subj$ is falsified in $\ll$ iff it contains a literal falsified in $\ll$. □

We can now proceed to postulates. Our goal is to transfer the discussion about problems with logic program updates from examples and counterexamples to some general principles.[6] Our postulates specify which dependencies should be ignored (i.e. not considered when creating a coherent semantic view on a set of dependencies). We believe that the postulates are clear and can be continually improved, if needed (note that a translation of many formalisms to the dependency framework is possible). The construction of a coherent semantic view on a set of dependencies presented in this paper satisfies the postulates.

It is supposed in Postulates below that

- a dependency relation $\ll$ is given,
- a finite set $\mathcal{D} = \{\ll_1, \ldots, \ll_k\}$, where $\ll_i \subset \ll$, is given,
- an acyclic, transitive and irreflexive preference relation ρ on $\mathcal{D}$ is defined.

[5] Our approach does not reject or insert some rules. Its ambition is to provide a coherent view on a (possibly incoherent) MDyLP (NMKB) by *ignoring* some dependencies.

[6] An allied approach is presented in (Alferes et al.2005; Banti et al. 2005) as regards irrelevant updates (according to our terminology). We discuss the case of irrelevant updates in (Šefr´anek 2006b).

If $\ll_i, \ll_j \in \mathcal{D}$ and $\ll_i \ \rho \ \ll_j$, it is said that $\ll_j$ ($\ll_i$) is more (less) preferred as $\ll_i$ ($\ll_j$). Similarly, if $d \in \ll_j$ and $d' \in \ll_i$, it is said that d (d') is more (less) preferred than d' (d).

Postulate 1 *Let Ay be a set of assumptions falsified in $\ll$. Then all dependencies of the form $L \ll W$, where $Ay \subseteq W$, are ignored.*

Note that Postulate 2 is a generalization of two special cases (for two kinds of conflicts from Definition 12. The postulate corresponds partly to the CRP, but it also extends the possibilities of solving conflicts. Moreover, the general formulation of Postulate 2 enables also other kinds of solutions of a conflict of the form $\{L_1 \ll W, L_2 \ll W\}$ than those enabled by CRP, see Example 20 (we emphasize that bridging the gap between belief revision community and dynamic logic programming community is enabled as a consequence). Postulate 2 admits non-determinism: alternative good solutions of a conflict are possible. This non-determinism is an appropriate one in the context of stable model semantics.

Postulate 2 *Let C be a conflict. If there is a set S of good solutions of C, then one $D \in S$ is selected and ignored.*

Finally, the last postulate is introduced below. The postulate enables to distinguish more preferred TSSOAs (stable models) in order to be able solve the problems in the style of prioritized logic programming. Of course, there are different motivations behind (logic program) updates and preferential reasoning. On the other hand, *multidimensional* dynamic logic programming is aiming at distinguishing various kinds of preferences (w.r.t. time, agents, hierarchical instances, domains of knowledge, hierarchy of power, viewpoint etc. etc.) and we expect that it should (even must) be able also to select more preferred stable models in the style of prioritized logic programming. May be, different strategies of conflict solving along different dimensions in a multiprogram are needed (for a more detailed discussion see Example 21). We are proposing a first step toward that goal below. A preference relation on sets of assumptions is defined.

Example 15 If a less preferred program is $P = \{a \leftarrow not\ b\}$ and a more preferred program, its update, is $U = \{b \leftarrow not\ a\}$, we get two standard TSSOAs: $Ay_1 = \{not\ a\}$ and $Ay_2 = \{not\ b\}$ (and corresponding standard stable models).

However, we can consider Ay_1 as more preferred than Ay_2. An example from (Delgrande et al. 2003) is discussed in Example 21. □

We accept that sometimes it is useful also discriminate a preference relation on (sets of) assumptions. It enables to gain capabilities of logic programming with preferences in multidimensional dynamic logic programming, see Example 21.

Definition 16 Let be $S(not\ \ A) = \{i \in V \mid \exists r \in P_i\ not\ A \in body(r)\}$.

Let be $L, L' \in Subj$. The assumption L is *preferred at least as* the assumption L' iff for each maximal $i \in S(L')$ and each maximal $j \in S(L)$ holds either $i \preceq j$ or $i \parallel j$.

L is *more preferred* than L' iff L is preferred at least as L' and for at least one pair i, j holds $i \prec j$.

A set of subjective literals S is more preferred than the set of subjective literals S' iff each $L \in S \setminus S'$ is preferred at least as each $L' \in S' \setminus S$ and there is an $L \in S \setminus S'$ more preferred as each $L' \in S' \setminus S$. □

Definition 17 Let $\mathcal{P} = (\Pi, G)$ be a multiprogram, $G = (V, E)$, let be $i, s, t \in V$.

It is said that a set of assumptions Ay is *falsified w.r.t.* a more preferred set of assumptions X and a dependency relation $View \subseteq \ll_{\bigcup_{i \preceq s} P_i}$ iff

- X is a TSSOA w.r.t. $View$,
- there are $L_1 \in X$ and $L_2 \in Ay$ such that $not\ L_2 \in Cn_{View}(X)$,
- X is not falsified and it is also not falsified w.r.t. some Y and some $View' \subseteq \ll_{\bigcup_{i \preceq t} P_i}$, where $s \prec t$, $View \subseteq View'$, Y is a TSSOA w.r.t. $View'$. □

We will use a shorthand "wrt-falsified" supposing that the corresponding more preferred set of assumptions and the corresponding dependency relation are implicitly clear or it does not matter what set of assumptions and what dependency relation are considered.

Finally, the last postulate is introduced for the case, if a preference relation on sets of assumptions is defined.

Postulate 3 *Let σ be an acyclic, transitive and irreflexive preference relation on sets of assumptions. If Ay is wrt-falsified, then all dependencies of the form $L \ll W$, where $Ay \subseteq W$, are ignored.*

Semantics of multiprograms based on assumptions and dependencies. We outline now a preliminary characterization of semantics of a multiprogram (in our dependency framework). The semantics is characterized by a mapping from a multiprogram $\mathcal{P}$ to a set of pairs of the form $(Ay, View)$, where $View \subseteq \ll_{\bigcup_{i \in V} P_i}$ and Ay is a TSSOA w.r.t. $View$. It means that our semantics is looking for a conflict-free subset of $\ll_{\bigcup_{i \in V} P_i}$ and for a reasonable set of assumptions (while Postulates 1 – 3 are satisfied). Technical details of the semantics are motivated and elaborated below.

Semantics based on rejection of rules

We will point out some drawbacks of semantics based on rejection of rules in this section. Only examples of the form $\langle P, U\rangle$, where $P \prec U$, are used (with the only exception). Refined dynamic stable model semantics is used in the analysis of the examples, but our arguments are applicable to any semantics focused only on conflicts between the heads of rules (hence, also to the general multiprograms).

We are proceeding to an example illustrating problem of *irrelevant* updates.

Example 18 ((Eiter et al. 2002))

$$\begin{aligned} P = it\_is\_cloudy &\leftarrow it\_is\_raining \\ it\_is\_raining &\leftarrow \\ U = not\ it\_is\_raining &\leftarrow not\ it\_is\_cloudy \end{aligned}$$

$RDSM(\langle P, U\rangle) = \{\{not\ it\_is\_raining, not\ it\_is\_cloudy\}, \{it\_is\_raining, it\_is\_cloudy\}\}$.

Notice that $it\_is\_raining \ll_{P\cup U} \{not\ it\_is\_cloudy\}$, $not\ it\_is\_raining \ll_{P\cup U} \{not\ it\_is\_cloudy\}$, but the assumption $not\ it\_is\_cloudy$ is *falsified* in $\ll_{P\cup U}$ because of $it\_is\_cloudy \ll_{P\cup U} \emptyset$. Information given by U do not override the information of P (which is based on the empty set of assumptions). The only TSSOA w.r.t. $\ll_{P\cup U}$ is $\emptyset$. Consider Postulate 1.

Irrelevant updates are analyzed in a more detail in (Šefránek 2006b).

In general, troubles with all semantics based on rejection of rules are caused also by a too free choice of an interpretation involved in the fixpoint condition (1). We mean an interpretation containing falsified assumptions (default negations). Interpretations generated by falsified assumptions do not provide an appropriate candidate for a semantic characterization of a multiprogram. A remark is in place: conflicts involving assumptions did not attract an adequate attention until now. □

Incompleteness of updates based on CRP is illustrated by the next example.

Example 19 [7]

$$\begin{aligned} P &= \{obedient \leftarrow punish\} \\ U &= \{punish \leftarrow not\ obedient\}. \end{aligned}$$

There is no conflict between heads of rules. Hence, no rule can be rejected and the meaning of $P \cup U$ cannot be updated according to CRP. However, there is a kind of conflict between both programs. $P \cup U$ has no stable model, but both P and U are coherent.

There is no clear reason why to solve conflicts between heads of rules and not to solve other conflicts. Also conflicts caused by some assumptions and conflicts between dependencies on some assumptions are relevant (for non-monotonic reasoning).

The third postulate of (Katsuno and Mendelzon 1991), 3[KM] hereafter (if both ψ and μ are satisfiable then $\psi \diamond \mu$ is also satisfiable, where ψ is a knowledge base, μ is an update and $\diamond$ is an update operation) cannot be satisfied by a semantics focused only on rejection of rules with conflicting heads. □

We propose to ignore less preferred dependencies in order to cut off a dependency of an atom A on the assumptions containing $not\ A$.[8]

[7] This is a variant of an example from (Pereira and Pinto 2005).

[8] We do not exclude that reasoning based on reduction ad absurd (Pereira and Pinto 2005) is useful for knowledge based systems. However, we prefer here constructive mode of reasoning and a kind of compatibility with answer set programming paradigm.

Another of the drawbacks is that CRP is not able to recognize alternative solutions of a given inconsistency (note a striking difference w.r.t. the belief revision research).

Example 20 [9]

$$P = \{a \leftarrow; b \leftarrow\} \qquad U = \{not\ a \leftarrow b\}$$

$Rej^R(\langle P, U\rangle, \{not\ \ a, b\}) = \{a \leftarrow\}$ and $RDSM(\langle P, U\rangle) = \{\{not\ \ a, b\}\}$. However, it is not clear why $a \leftarrow$ can be rejected and $b \leftarrow$ cannot be rejected. There are two (if we respect the preference relation) maximal coherent subsets of incoherent $P \cup U$ and two corresponding stable models – besides $\{not\ \ a, b\}$ also $\{not\ b, a\}$.

Alternative solutions of this conflict using *nonmonotonic integrity constraints* are proposed in (Šefránek 2006a). However, notice that the more general formulation of Postulate 2 in present paper enables alternative solutions, too. □

Let us proceed to logic programs with preferences. Note that there is a trivial correspondence between prioritized logic programs and multidimensional dynamic logic programs. Consider first static preferences (on rules). Let a MDyLP $\mathcal{P}$ be given. If $P_i \prec P_j$ then for each $r \in P_i$ and each $r' \in P_j$ holds $r \prec r'$. Otherwise, rules are incomparable.

Conversely, let a prioritized logic program be given as a pair $(\{r_i \mid i \in I\}, \prec)$. The corresponding MDyLP we obtain as a set of programs (singletons) $P_i = \{r_i\}$ preserving $\prec$: $P_i \prec P_j$ iff $r_i \prec r_j$.

If preference relation is a dynamic one, then a dynamic preference relation on programs is needed. A possibility of such extension of MDyLP is supposed (f.ex. in (Alferes et al. 1998)), but we are not aware of a realization of the possibility. However, it is feasible and straightforward.

Example 21 [10] $P_1 = \{c \leftarrow not\ b\}, P_2 = \{a \leftarrow\}, P_3 = \{b \leftarrow a, not\ c\}$; $P_1 \prec P_3$, $P_1 \parallel P_2 \parallel P_3$.
$RDSM(\langle P, U\rangle) = \{\{a, b, not\ c\}, \{a, c, not\ b\}\}$. There are no conflicting rules in this multiprogram, hence no rule can be rejected.

The more preferred model (not only) according to (Delgrande et al. 2003) is $\{a, b, not\ c\}$.

Suppose now that $Ay_1 = \{not\ c\}$ is more preferred than $Ay_2 = \{not\ b\}$. Hence, "it is not known b" seems not to be a reasonable assumption and we can consider it as falsified by the *more preferred* set of assumptions.

In technical terms introduced in Definition 16: $S(not\ \ b) = \{1\}$ and $S(not\ \ c) = \{3\}$, hence $not\ \ c$ is more preferred than $not\ \ b$. Further, $\{not\ c\}$ is a TSSOA w.r.t. $\ll_{P_1 \cup P_2 \cup P_3}$, it is neither falsified nor wrt-falsified and finally, $b \in Cn_{\ll_{P_1 \cup P_2 \cup P_3}}(\{not\ c\})$. Therefore, $\{not\ b\}$ is falsified w.r.t. $\{not\ c\}$ and $\ll_{P_1 \cup P_2 \cup P_3}$.

[9] This example is due to Martin Bal´až. More thorough discussion of this Example is in (Šefr´anek 2006a).

[10] This is an adapted version of an example from (Delgrande et al. 2003). For a more thorough discussion see (Šefr´anek 2006a).

There is an intuitive difference between updates and preferences (see (Alferes and Pereira 2000)). However, the *multidimensional* approach of MDyLP should represent also preferential reasoning. May be, different strategies for different dimensions are needed. Moreover, there are some problems with very notion of update, if default negations are allowed (even in heads of rules). We will devote a future research to the topic of updates and revisions of NMKBs. □

A coherent semantic view on a set of dependencies

We are going to define a reasonable semantic view on a set of dependencies $\ll$. The view will be a set of dependencies $View$, which is

- coherent (in the sense defined below),
- a subset of $\ll$.

Definition 22 (Coherent dependency relation) A dependency relation $\ll$ is called *coherent* iff there is a TSSOA w.r.t. $\ll$. A dependency relation is called *incoherent* iff it is not a coherent one. □

Consequence 23 *Let a dependency relation $\ll$ be coherent. Let Ay be a TSSOA w.r.t. $\ll$. Then there is no $C \subseteq \ll$ s.t. $C = \{A \ll Ay, not\ A \ll Ay \mid A \in \mathcal{A}\}$ or $C = \{A \ll Ay \mid A \in \mathcal{A} \wedge not\ A \in Ay\}$.*

Proof: $Cn_{\ll}(Ay)$ is consistent. □

In general, $\ll_{\bigcup_{i \in V} P_i}$ can be incoherent. Our approach to semantics of MDyLP is focused on looking for assumptions which can serve as TSSOA w.r.t. a corresponding subset of given dependency relation $\ll_{\bigcup_{i \in V} P_i}$. Note that more reasonable semantic views on a set of dependencies are possible (of course, this can be expected – stable model semantics is at the background of our constructions).

There are different TSSOAs w.r.t. different subsets of $\ll_{P \cup U}$ in next example.

Example 24

$$P = \{not\ b \leftarrow not\ a \quad not\ a \leftarrow not\ b\} \qquad U = \{b \leftarrow not\ a \quad a \leftarrow not\ b\}$$

$Ay_1 = \{not\ \ a\}, Ay_2 = \{not\ \ b\}$, $View_1 = Cl((\ll_P \cup \ll_U) \setminus \{not\ b \ll_P \{not\ a\}\})$, $View_2 = Cl((\ll_P \cup \ll_U) \setminus \{not\ a \ll_P \{not\ b\}\})$.

$View_1$ is coherent (Ay_1 is a TSSOA w.r.t. $View_1$) and also $View_2$ is coherent (Ay_2 is a TSSOA w.r.t. $View_2$) □

We are aiming to construct all possible dependency (sub)relations which are coherent (w.r.t. a TSSOA), see Example 24. Note that also coherent dependency relations can be revised (until we obtain all possible pairs of the form $(Z, View)$, where Z is a TSSOA w.r.t. $View$. Hence, we are aiming to solve conflicts connected to some subsets of $Subj$.

Definition 25 If $Y \subseteq Subj$ is fixed and $C \subseteq \ll$ be a conflict (i.e. $C = \{(A, Y), (not\ A, Y) \mid Y \subseteq Subj\}$ or $C = \{(A, Y) \mid Y \subseteq Subj, not\ A \in Y\}$), then it is said that C is a conflict *of* Y. □

If we are looking for coherent subsets of a dependency relation, we are focused on solving conflicts of candidates for TSSOAs. Pairs of assumptions and dependencies are relevant for our semantics of dynamic logic programs.

We will describe a construction of a coherent dependency relation from an incoherent $\ll_{\bigcup_{i\in V} P_i}$. The constructed relation represents – in a sense – a coherent semantic view on an incoherent multiprogram. We present a nondeterministic algorithm which iteratively finds preferred minimal solutions of conflicts in $\ll_{\bigcup_{i\in V}}$ and simultaneously constructs TSSOAs, see Figure 1. The algorithm iteratively solves conflicts and modifies assumptions. Observe that Postulates of Section *Conflicts, solutions, postulates* are obeyed in the algorithm.

Let describe the behaviour of the algorithm. It is assumed that there is a set Ω containing pairs of the form $(Z, View)$, where Z are assumptions and $View$ is a dependency relation. Z is neither falsified in $\ll_{\bigcup_{i\in V} P_i}$ (nor falsified w.r.t. a more preferred set of assumptions, if a preference relation on sets of assumptions is accepted). Initially, $View$ is $\ll_{\bigcup_{i\in V} P_i}$.

If Z is a TSSOA w.r.t. $View$, the task is done. Otherwise, conflicts are solved in a REPEAT – UNTIL cycle. First a set of all good solutions of a conflict is identified. For each good solution a coherent view $View_i$ is computed. Only the first computed view is processed, alternative views are collected in Ω.

The value of variable i is 0 after the *for each*-cycle, if there is no solution of the conflict and the computation failed. If Z is not a TSSOA and the conflict has been solved in the current run of the REPEAT-UNTIL cycle and if there are no more conflicts in current $View^T$ then $Cn_{View^T}(Z)$ is consistent but not complete and the cycle is finished in the next run because of $i = 0$. Otherwise, Z is a TSSOA w.r.t. the computed $View^T$.

We have to overcome the final complication before proceeding to the semantic characterization of multiprograms. It is proposed to accept minimal sets of assumptions. Let a multiprogram $\mathcal{P}$ be given. If Y and Y' are TSSOAs w.r.t. $View$, $View' \subseteq \ll_{\bigcup_{i\in V} P_i}$, respectively, and $Y \subset Y'$, then only Y is selected as a proper semantic characterization of $\mathcal{P}$. See next Example as an illustration.

Example 26 [11]

$$P = \{a \leftarrow not\ c \qquad b \leftarrow a\} \qquad U = \{not\ a \leftarrow not\ b\}$$

$Ay_1 = \{not\ c\}$, $a \ll_{P\cup U} Ay_1, b \ll_{P\cup U} Ay_1$. $Ay_2 = \{not\ c, not\ b\}$, $View_2 = Cl((\ll_P \cup \ll_U) \setminus \{a \ll_P \{not\ c\}\})$.

[11] This Example is proposed by J.Leite.

INPUT: a pair $(Z, View)$ from Ω, where $Z \subseteq Subj$, $View \subseteq \ll_{\bigcup_{i\in V} P_i}$
OUTPUT: a pair $(Z, View^T)$, where Z is a TSSOA w.r.t. $View^T$ or the decision that it is not possible to construct an TSSOA from Z
begin
  if Z is a TSSOA w.r.t. $View$ **then** RETURN $(Z, View)$
  fi
  $View^T := View$
  REPEAT
    if $View^T$ contains a conflict C of Z **then**
      SELECT the set Π of all D, good solutions of C
    fi
    $i := 0$;
    for each $D \in \Pi$ **do**
      $i := i + 1$; $View_i^T := Cl(View^T \setminus D)$
      if $i > 1$ **then** $\Omega := \Omega \cup (Z_i^T, View_i^T)$ **fi**;
      $View^T := View_1^T$
    od
    if $i = 0$ **then** FAILURE := true
      else FAILURE := false **fi**
  UNTIL Z is a TSSOA w.r.t. $View^T$ or FAILURE
  if not FAILURE **then** RETURN $(Z^T, View^T)$
      else RETURN FAILURE **fi**
end

Figure 1: Non-deterministic algorithm removing conflicts and computing TSSOAs

There are two TSSOAs: Ay_1 w.r.t. $\ll_{P\cup U}$ and Ay_2 w.r.t. $View_2$. The second one contains more non-monotonic assumptions and we prefer to accept minimal sets of assumptions. □

Definition 27 Let $\mathcal{P}$ be a multiprogram. Let $View \subseteq \ll_{\bigcup_{i\in V} P_i}$ be coherent and Z be a TSSOA w.r.t. $View$.

It is said that Z is a good set of assumptions (GSOA) w.r.t. $View$ iff there is no TSSOA Z' w.r.t. a $View' \subseteq \ll_{\bigcup_{i\in V} P_i}$ s.t. $Z' \subset Z$. □

Definition 28 (Semantics of multiprograms) Semantics of multiprograms is a mapping Σ from multiprograms to sets of pairs of the form $(Z, View)$, where $View$ is a coherent subset of $\ll_{\bigcup_{i\in V} P_i}$ w.r.t. assumptions Z and Z is a GSOA w.r.t. $View$. □

We can create canonical programs[12] determined by all GSOAs of a multiprogram.

Definition 29 Let $\{Z_1, \ldots, Z_k\}$ be all GSOAs w.r.t. $\{View_1, \ldots, View_k\}$, respectively, of a multiprogram $\mathcal{P}$. Then a program of the form $\{L \leftarrow Z_i \mid L \in Cn_{View_i}(Z_i), L \notin Z_i, i = 1, \ldots, k\}$ is called *canonical*

[12] For similar construction see (Nov´ak draft; Šefr´anek 2004; Šefr´anek 2000).

program representing a (maximally) coherent semantic view on $\mathcal{P}$. □

Theorem 30 *Let Q be a canonical program representing a multiprogram $\mathcal{P}$. The set of all stable models of Q coincide with the set of sets $\{Cn_{View_i}(Z_i) \mid Z_i$ is a GSOA w.r.t. $View_i\}$*

Evaluation

Some important properties of the semantics based on dependency framework are presented in this section.

We consider 3[KM] as a criterion of completeness of our approach: if all programs from a multiprogram are coherent then a coherent view (on the dependency relation generated by the programs of multiprogram) is required. Solution of all conflicts generated by interactions of programs seems to be a reasonable minimal condition for maintaining coherence of multiprograms. Incoherence of a single program is a problem of another kind than solution of conflicts between programs.

An analogy of 3[KM] is satisfied in our dependency framework. We express it first for the simplest case.

Theorem 31 *Let $\langle P, U\rangle$ be a multiprogram. If $\ll_P$ and $\ll_U$ are coherent then there is also a coherent dependency relation $View \subseteq \ll_{P\cup U}$.*

Proof: Suppose that there is no Ay which is a TSSOA w.r.t. $\ll_{P\cup U}$. It means that for each Ay is $Cn_{\ll_{P\cup U}}(Ay)$ inconsistent or incomplete. Incompleteness is excluded if TSSOAs w.r.t.$\ll_U$ or $\ll_P$ are considered.

Consider Ay which is a TSSOA w.r.t. $\ll_U$ and it is not falsified. We supposed that there is a conflict C contained in $\ll_{P\cup U}$, i.e. $A \ll_{P\cup U} Ay, not\ A \ll_{P\cup U} Ay$ or $A \ll_{P\cup U} Ay, not\ A \in Ay$. Of course, the conflict is not caused by $\ll_U$, so there is $D \subseteq \ll_P$ such that $Cl(\ll_{P\cup U} \backslash D)$ does not contain C.

If Ay is falsified: let be $X = \{A \in \mathcal{A} \mid A \ll_P \emptyset, not\ A \in Ay\}$. It holds for each TSSOA Bss w.r.t. $\ll_P$ that $X \subseteq Cn_{\ll_P}(Bss)$. If Bss is falsified w.r.t. a set of assumptions Y and $\ll_{P\cup U}$ then there is nothing to prove (Y is not falsified and it is a TSSOA w.r.t. $\ll_U$, the situation can be reduced to that of the previous paragraph).

Hence, assume that Bss is not falsified w.r.t. to some set of assumptions and $\ll_{P\cup U}$. Therefore, only conflicts of the form $\{A \ll_{P\cup U} Bss, not\ A \ll_{P\cup U} Bss\}$ have to be solved (and there is a good solution of such conflicts).

Finally, conflicts removing can be repeated until a coherent subset is reached. □

Consequence 32 *Let $\mathcal{P} = \langle \mathcal{P}_\infty, \ldots, \mathcal{P}_\| \rangle$ be a dynamic logic program. If each $\ll_{P_i}$ is coherent then there is also a coherent dependency relation $View \subseteq \ll_{\bigcup_j P_j}$.*

Conflicts among incomparable programs are not solvable in a natural way, so for the general case of multiprograms we accept also the condition that the union of incomparable programs is coherent.

Theorem 33 *Let $\mathcal{P} = (\Pi, G)$ be a multiprogram. Let be $\ll_{P_i}$ coherent for each $i \in V$. Let $\mathcal{I}$ be the set of all incomparable programs in $\mathcal{P}$ (i.e., for each $P_i, P_j \in \mathcal{I}, i \neq j$ holds $i \parallel j$), let be $N = \{i \in V \mid P_i \in \mathcal{I}\}$.*

If $\ll_{\bigcup_{i\in N} P_i}$ is coherent, then there is also a coherent dependency relation $View \subseteq \ll_{\bigcup_{j\in V} P_j}$.

Our framework is immune w.r.t. tautological updates.

Proposition 34 (Tautological updates) *Let P be a program, τ be a tautology (i.e. $head(\tau) \in body(\tau)$) and $U = \{\tau\}$. Then each GSOA w.r.t. $\ll_{P\cup U}$ is a GSOA w.r.t. $\ll_P$.*

Proof is trivial - U does not generate a new dependency. □

Consequence 35 *Let $\mathcal{P}$ be a multiprogram (some of the programs in Π may be empty). Let for each $i \in V$ there is a (possibly empty) set of tautologies T_i. Let $\Sigma(\mathcal{P})$ be the set of all GSOAs w.r.t. corresponding $View \subseteq \ll_{\bigcup_{i\in V} P_i}$.*

Then the set of all GSOAs w.r.t. the corresponding $View \subseteq \ll_{\bigcup_{i\in V} P_i\cup T_i}$ is precisely $\Sigma(\mathcal{P})$.

Irrelevant updates (and unsupported cyclic updates, as a special case) are avoided in our semantics.

Rejection of conflicting rules (according to the dynamic stable model semantics) is satisfied in the dependency framework.

Proposition 36 *If $r \in Rej(\mathcal{P}, M)$ then there is a good solution of conflict $C = \{head(r) \ll_{P_i} M, not\ head(r) \ll_{P_j} M\}$, where $i \prec j$.*

Conclusions, discussion, open problems

We summarize main contributions of the paper as follows:

- some drawbacks of CRP, which did not attract a sufficient attention until now, were discussed,
- a step toward bridging the gap between the research in dynamic logic programming on the one hand and belief revision or preferences handling on the other hand is done,
- a dependency framework (close in the spirit to stable model semantics) has been developed as a basis for an analysis of conflicts contained in a MDyLP and also for their solution,
- a semantics of MDyLP based on the dependency framework has been developed,
- a set of postulates governing solution of conflicts is proposed,
- the proposed semantics is immune w.r.t. tautological, cyclic and irrelevant updates, it is complete (w.r.t 3[KM]), rejection of conflicting rules is satisfied,
- the semantics uses a construction of a coherent semantic view on a set of dependencies (a non-deterministic algorithm is presented); the construction respects postulates expressed in the paper; the semantics avoids drawbacks analyzed in the paper.

Remarks of technical nature: Only generalized logic programs are considered in the paper in order to support a simple discussion of refined dynamic stable semantics (Alferes et al.2005). However, an extension to generalized extended logic programs (GELP) is straightforward and will be presented in a forthcoming paper. Examples with pairs of programs P and U, where U is more preferred than P, are used in order to be as clear and simple as possible. However, the construction is applicable (and results hold) for the general case.

Presented approach opens some problems and topics for our ongoing or future research.

At the first place we would like to mention the problem of updates of NMKB. Second postulate of (Katsuno and Mendelzon 1991) cannot be taken literally because of the presence of default (nonmonotonic) assumptions in NMKB. We believe that our dependency framework represents a good starting point for better understanding of updates of NMKB. Similarly, the relation of updates and revisions in the framework of NMKB is an open and interesting problem including a construction of a unified view on update and revision (revisions of incomplete and imprecise knowledge about changing world).

We are aiming at a more detailed elaboration of postulates for logic program updates. An evaluation of our Kripkean semantics (Šefránek 2000; Šefránek 2004) in terms of postulates and also an evaluation and comparison of other frameworks for NMKB and defeasible reasoning (argumentation) in terms of the dependency framework is a topic for future research. Default negations in heads should be rethought. A challenging task is to use different strategies for conflict solutions along different dimensions in MDyLP. The relation of our approach to prioritized logic programs deserves a more detailed attention. A detailed analysis and comparison of the refined extension principle to irrelevant updates should be done. Satisfaction of fifth postulate of (Katsuno and Mendelzon 1991) and existence of a state condensing operator of (Leite 2003) in our framework is also an interesting open problem. We intend to investigate connections of our approach to other approaches to belief revision based on a notion of dependency (Darwiche and Pearl 1997; del Cerro and Herzig 1996) and others. Last, but not least, computational properties of our semantics have to be studied. Some of the problems mentioned above are topics of an ongoing research.

Acknowledgments. The work was supported under grants APVV-20-P04805 and VEGA 1/3112/06. I am grateful to one of the anonymous reviewers for his comments and also to Martin Baláž, Martin Homola and Jozef Šiška for many discussions about a previous version of the dependency framework.

References

Alferes, J.J., Pereira, L.M.: Reasoning with logic programming. Springer 1996

Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In: Procs. of KR'98. (1998) 98–109

Alferes, J.J., Pereira, L.M.: Updates and preferences, Proc. of JELIA 2000. Springer.

Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica **1** (2005)

Banti, F., Alferes, J.J., Brogi, A., Hitzler, P.: The well supported semantics for multidimensional dynamic logic programs. LPNMR 2005, LNCS 3662, Springer, 356-368

Darwiche, A., Pearl, J.: On the logic of iterated belief revision. Artificial Intelligence, 89, 1997

del Cerro, L.F., Herzig, A,: Belief change and dependence. Procs.of TARK VI, Morgan Kaufmann, 1996

Delgrande, J., Schaub, T., Tompits, H.: A Framework for Compiling Preferences in Logic Programs, Theory and Practice of Logic Programming 3(2), 2003, pp. 129-187

Eiter, T., Sabbatini, G., Fink, M., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming (2002) 711–767

Gärdenfors, P., Rott, H.: Belief revision. In: Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4 (Epistemic and Temporal Reasoning), Claredon Press. Oxford 1995

Homola, M., Šefránek, J., Baláž, M., Abstract dependency theory: preliminary report. http://www.computational-logic.org/content/events/iccl-ss-2005/talks/MartinHomola.ps

Katsuno, H., Mendelzon, A.O.: On the difference between updating a knowledge base and revising it. Proc. of KR'91

Konczak, K., Linke, T., Schaub, T.: Graphs and Colorings for Answer Set Programming: Abridged Report. LPNMR 2004: 127-140

Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-dimensional dynamic knowledge representation. In: LPNMR 2001, Springer, 365–378

Leite, J.A.: Evolving Knowledge Bases: Specification and Semantics. IOS Press (2003)

Novák, P.: Stable Model Semantics Algorithm: Approach Based on Relation of Blocking Between Sets of Defaults; 2004; http://peter.aronde.net/publications.html

Pereira, L.M., Pinto, A.M.: Revised Stable Models - A Semantics for Logic Programs. EPIA 2005: 29-42

Šefránek, J.: Semantic considerations on rejection. In: Procs. of NMR 2004.

Šefránek, J.: A Kripkean semantics for logic program updates. : In M. Parigot, A. Voronkov (eds.), Logic for Programming and Automated Reasoning. Springer 2000, LNAI 1955

Šefránek, J.: Nonmonotonic integrity constraints. In Fink M., Tompits, H., Woltran, S. (eds.): Proc. of 20th Workshop on Logic Programming (WLP 2006), Vienna, 2006.

Šefránek, J.: Irrelevant updates of nonmonotonic knowledge bases. Accepted as a poster for ECAI 2006.