I N F S Y S
R E S E A R C H
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME
ARBEITSBEREICH WISSENSBASIERTE SYSTEME

15TH INTERNATIONAL WORKSHOP ON NON-MONOTONIC REASONING (NMR 2014)

VIENNA, AUSTRIA, JULY 17-19, 2014

PROCEEDINGS

Sébastien Konieczny and Hans Tompits (eds.)

INFSYS RESEARCH REPORT 1843-14-01
JULY 2014

Institut für Informationssysteme
Arbeitsbereich
Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-14-01, JULY 2014

PROCEEDINGS OF THE 15TH INTERNATIONAL WORKSHOP ON NON-MONOTONIC REASONING (NMR 2014)

VIENNA, AUSTRIA, JULY 17-19, 2014

Sébastien Konieczny and Hans Tompits[1]
(Volume Editors)

[1] Editors' address: Sébastien Konieczny, CRIL-CNRS, Faculté des Sciences, Université d'Artois, 62300 Lens, France, e-mail: konieczny@cril.fr.
Hans Tompits, Institut für Informationssysteme, Arbeitsbereich Wissensbasierte Systeme, Technische Universität Wien, Favoritenstraße 9-11, 1040 Vienna, Austria, e-mail: tompits@kr.tuwien.ac.at.

Preface

This volume consists of the contributions presented at the 15th International Workshop on Non-Monotonic Reasoning (NMR 2014), which was held at the Vienna University of Technology, Austria, from July 17 to 19, 2014.

The NMR workshop series is the premier specialized forum for researchers in non-monotonic reasoning and related areas. Its aim is to bring together active researchers in the broad area of non-monotonic reasoning, including belief revision, reasoning about actions, argumentation, declarative programming, preferences, non-monotonic reasoning for ontologies, uncertainty, and other related topics.

Previous NMR workshops were held in New Paltz (New York, USA) in 1984, Grassau (Germany) in 1988, South Lake Tahoe (California, USA) in 1990, Plymouth (Vermont, USA) in 1992, Schoss Dagstuhl (Germany) in 1994, Timberline (Oregon, USA) in 1996, Trento (Italy) in 1998, Breckenridge (Colorado, USA) in 2000, Toulouse (France) in 2002, Whistler (Canada) in 2004, Lake District (UK) in 2006, Sydney (Australia) in 2008, Toronto (Canada) in 2010, and Rome (Italy) in 2012.

It is a tradition for many years that NMR is collocated with the International Conference on Principles of Knowledge Representation and Reasoning (KR) as well as with the International Workshop on Description Logics (DL)—and this year is no exception. Additionally, like for the last event in 2012, NMR and DL share an invited speaker as well as common technical sessions. A particular noteworthy fact is that all of these events are organized as part of the Vienna Summer of Logic, that also hosts FLoC 2014, the Federated Logic Conference. As that, our event is part of the probably largest gathering of logic-related events in the history of science.

We would like to thank our three invited speakers, Philippe Besnard (IRIT at Université Toulouse III Paul Sabatier), Patrick Blackburn (University of Roskilde; joint speaker with DL 2014), and Hans Rott (Universität Regensburg), as well as all the Track Chairs and Program Committee Members that helped us to organize such a great event!

For the workshop, 33 technical papers have been accepted. These technical contributions cover the full spectrum of NMR, from declarative programming, uncertainty, causality, inference, and non-monotonic logics, to description logics, belief change, and argumentation. There are also contributions dedicated to system descriptions and a special track on benchmarks for NMR.

We would like to thank all authors, reviewers, and participants for their involvement in our event, as well as all the people who helped in organizing the workshop. Particularly, we would like to thank Thomas Schmidleithner who did an exceptional job for taking care of the web-presence of NMR 2014. As well, we would also like to acknowledge the valuable asset of having the EasyChair conference management system at our disposal. Last, but not least, we thank our sponsors, KR Inc. and the Artificial Intelligence Journal, and the Kurt Gödel Society as the principal organizer of the Vienna Summer of Logic.

July 2014

Sébastien Konieczny and Hans Tompits,
NMR 2014 Workshop Chairs

Organization

Workshop Chairs

Sébastien Konieczny (*CRIL-CNRS, Université d'Artois*)
Hans Tompits (*Vienna University of Technology*)

Track Chairs

Actions, Causality, and Belief Change Track

Renata Wasserman (*Universidade de São Paulo*)

Declarative Programming Track

Tomi Janhunen (*Aalto University*)

Argumentation and Dialog Track

Paul E. Dunne (*University of Liverpool*)

Preferences, Norms, and Trust Track

Mehdi Dastani (*Utrecht University*)

NMR and Uncertainty Track

Lluis Godo (*Universitat Autonoma de Barcelona*)

Commonsense and NMR for Ontologies Track

Guilin Qi (*Southeast University China*)

Systems and Applications Track

Esra Erdem (*Sabanci University*)

Benchmarks for NMR Track

Sébastien Konieczny (*CRIL-CNRS, Université d'Artois*)

Program Committee

Marcello Balduccini (*Drexel University*)
Christoph Beierle (*FernUniversität in Hagen*)
Richard Booth (*University of Luxembourg*)
Gerhard Brewka (*University of Leipzig*)
Jan Broersen (*Utrecht University*)
Nadia Creignou (*Aix-Marseille Université*)
Mehdi Dastani (*Utrecht University*)
Marina De Vos (*University of Bath*)
James P. Delgrande (*Simon Fraser University*)
Marc Denecker (*K.U. Leuven*)
Jürgen Dix (*Clausthal University of Technology*)
Paul E. Dunne (*University of Liverpool*)
Ulle Endriss (*University of Amsterdam*)
Esra Erdem (*Sabanci University*)
Patricia Everaere (*Université de Lille 1*)
Wolfgang Faber (*University of Huddersfield*)
Michael Fink (*Vienna University of Technology*)
Martin Gebser (*Aalto University*)
Michael Gelfond (*Texas Tech University*)
Lluis Godo (*Universitat Autonoma de Barcelona*)
Guido Governatori (*NICTA*)
Sven Ove Hansson (*KTH Royal Institute of Technology*)
Andreas Herzig (*Université Toulouse III Paul Sabatier*)
Zhisheng Huang (*Vrije University Amsterdam*)
Anthony Hunter (*University College London*)
Katsumi Inoue (*National Institute of Informatics, Japan*)
Tomi Janhunen (*Aalto University*)
Gabriele Kern-Isberner (*Technische Universität Dortmund*)
Sébastien Konieczny (*Université d'Artois*)
Joohyung Lee (*Arizona State University*)
Thomas Meyer (*University of Kwazulu-Natal and CSIR Meraka Institute*)
Alessandra Mileo (*Digital Enterprise Research Institute, Galway*)
Marie-Laure Mugnier (*Université Montpellier 2*)
Nir Oren (*University of Aberdeen*)
Maurice Pagnucco (*The University of New South Wales*)
Ramon Pino Perez (*Universidad de Los Andes*)
Henri Prade (*Université Toulouse III Paul Sabatier*)
Guilin Qi (*Southeast University China*)
Francesco Ricca (*University of Calabria*)
Ken Satoh (*National Institute of Informatics and The Graduate University of Advanced Studies, Japan*)
Steven Schockaert (*Cardiff University*)
Guillermo Ricardo Simari (*Universidad Nacional del Sur*)
Terrance Swift (*CENTRIA, Universidade Nova de Lisboa*)
Eugenia Ternovska (*Simon Fraser University*)
Hans Tompits (*Vienna University of Technology*)
Francesca Toni (*Imperial College London*)
Mirek Truszczynski (*University of Kentucky*)
Serena Villata (*INRIA Sophia Antipolis*)

Kewen Wang	(*Griffith University*)
Renata Wasserman	(*Universidade de São Paulo*)
Emil Weydert	(*University of Luxembourg*)
Stefan Woltran	(*Vienna University of Technology*)

Local Organization

Hans Tompits

Thomas Schmidleithner (Webpage)
Eva Nedoma (Secretary)

Additional Referees

Jean-Franois Baget
Gerald Berger
Bart Bogaerts
Giovanni Casini
Kristijonas Cyras
Jo Devriendt
Jianfeng Du
Sarah Alice Gaggl
Antonis Kakas
Hiroyuki Kido
Ho-Pun Lam
Marius Lindauer
Marco Manna
Max Ostrowski
Chiaki Sakama
Daria Stepanova
Kazuko Takahashi
Shahab Tasharrofi
Zhe Wang
Zhiqiang Zhuang

Table of Contents

Declarative Programming 2

Belief Change

Joint NMR/DL Contributed Papers

Benchmarks

Argumentation 1

Causality and Inference

Declarative Programming 3

Systems 2

Nonmonotonic Logics

Argumentation 2

Four Floors for the Theory of Theory Change

Hans Rott
Universität Regensburg
Department of Philosophy
93040 Regensburg, Germany
hans.rott@ur.de

Abstract

The theory of theory change due to Alchourrn, Grdenfors, and Makinson ("AGM") has been widely known as being characterised by two packages of postulates. The basic package consists of six postulates and is very weak, the full package adds two further postulates and is very strong. Revisiting the three classic constructions of *partial meet contraction*, *safe contraction*, and *entrenchment-based construction*, and tracing the idea of limited discriminative powers in agents, I argue that four intermediate levels can be distinguished that play important roles within the AGM theory.

Fragments of Logic, Language, and Computation

Patrick Blackburn
University of Roskilde
Department of Philosophy and Science Studies
Centre for Culture and Identity
Universitetsvej 1, 4000 Roskilde, Denmark
patrickb@ruc.dk

Abstract

Amsterdam-style logicians view modal logic as a fragment of classical logic, and description logicians view their own formalisms in much the same way. Moreover, first-order logic itself can be viewed as a modest fragment of the higher-order logics of Frege and Russell, a fragment with useful model-theoretic properties. All in all, the fine structure of logic is a key topic in contemporary research, as the intensive study of (say) the 2-variable and various guarded fragments attest.

In this talk I want to consider the role of logical fragments in applications. I will focus on applications in natural language, as this is an area rich in non-monotonic and defeasible inference. Moreover, as my perspective is that of computational (rather than theoretical) linguistics, I am interested in efficient solutions to computational tasks - that is, in fragments of computation. Drawing on a running example involving applications of description logic and classical planning to a dialogue system, I will discuss the role of computation to provide "pragmatic glue" that lets us work with small well-explored logical fragments, while simultaneously providing the dynamics required to model various forms of non-monotonicity.

Revisiting Postulates for Inconsistency Measures

Philippe Besnard

CNRS
IRIT – Université Paul Sabatier
118 rte de Narbonne, 31062 Toulouse cedex 9, France
besnard@irit.fr

Abstract

We discuss postulates for inconsistency measures as proposed in the literature.We examine them both individually and as a collection. Although we criticize two of the original postulates, we mostly focus on the meaning of the postulates as a whole. Also and accordingly, we discuss a number of new postulates as substitutes and/or as alternative families.

Introduction

In (Hunter and Konieczny 2008; Hunter and Konieczny 2010), Hunter and Konieczny have introduced postulates for inconsistency measures over knowledge bases. Let us first make it clear that the phrase "inconsistency measure" refers to the informal meaning of a measure, not to the usual formal definition whose countable additivity requirement would leave no choice for an inconsistency measure, making all minimal inconsistent knowledge bases in each cardinality to count as equally inconsistent (unless making some *consistent* formulas to count as more *inconsistent* than others!). However, we stick with the usual range $R^+ \cup \{\infty\}$ (so, the range is totally ordered and 0 is the least element). The intuition is: The higher the amount of inconsistency in the knowledge base, the greater the number returned by the inconsistency measure.

Let us emphasize that we deal with postulates for inconsistency measures that account for a raw amount of inconsistency: E.g., it will clearly appear below that an inconsistency measure I satisfying the (Monotony) postulate due to Hunter-Konieczny precludes I to be a ratio (except for quite special cases, see (Hunter and Konieczny 2010)).

HK Postulates

Hunter and Konieczny refer to a propositional language[1] $\mathcal{L}$ for classical logic $\vdash$. Belief bases are finite sequences over $\mathcal{L}$. $\mathcal{K}_\mathcal{L}$ is comprised of all belief bases over $\mathcal{L}$, in set-theoretic form (i.e., a member of $\mathcal{K}_\mathcal{L}$ is an ordinary set[2]).

According to Hunter and Konieczny, a function I over belief bases is an inconsistency measure if it satisfies the following properties, $\forall K, K' \in \mathcal{K}_\mathcal{L}$, $\forall \alpha, \beta \in \mathcal{L}$

- $I(K) = 0$ iff $K \nvdash \bot$ (Consistency Null)
- $I(K \cup K') \geq I(K)$ (Monotony)
- If α is free[3] for K then $I(K \cup \{\alpha\}) = I(K)$ (Free Formula Independence)
- If $\alpha \vdash \beta$ and $\alpha \nvdash \bot$ then $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$ (Dominance)

We start by arguing against (Free Formula Independence) and (Dominance) in the next section. We browse in the subsequent section several consequences of HK postulates, stressing the need for more general principles in each case. We then introduce various postulates supplementing the original ones, ending with a new axiomatization. We also devote a full section to a major principle, replacement of equivalent subsets. The section preceding the conclusion can be viewed as a kind of rejoinder backing (Monotony) and (Free Formula Independence) via the main new postulate.

Objections to HK Postulates

Objection to (Dominance)

In contrapositive form, (Dominance) says:

$$\text{For } \alpha \vdash \beta, \text{ if } I(K \cup \{\alpha\}) < I(K \cup \{\beta\}) \text{ then } \alpha \vdash \bot \quad (1)$$

but it makes sense that the lefthand side holds while $\alpha \nvdash \bot$. An example is as follows. Let $K = \{a \wedge b \wedge c \wedge \cdots \wedge z\}$. Take $\beta = \neg a \vee (\neg b \wedge \neg c \wedge \cdots \wedge \neg z)$ while $\alpha = \neg a$. We may hold $I(K \cup \{\alpha\}) < I(K \cup \{\beta\})$ on the following grounds:

- The inconsistency in $I(K \cup \{\alpha\})$ is $\neg a$ vs a.
- The inconsistency in $I(K \cup \{\beta\})$ is either as above (i.e., $\neg a$ vs a) or it is $\neg b \wedge \neg c \wedge \cdots \wedge \neg z$ vs $b \wedge c \wedge \cdots \wedge z$ that may be viewed as more inconsistent than the case $\neg a$ vs a, hence, $\{a \wedge b \wedge c \wedge \cdots \wedge z\} \cup \{\neg a \vee (\neg b \wedge \neg c \wedge \cdots \wedge \neg z)\}$ can be taken as more inconsistent overall than $\{a \wedge b \wedge c \wedge \cdots \wedge z\} \cup \{\neg a\}$ thereby violating (1) because $\alpha \nvdash \bot$ here.

Objection to (Free Formula Independence)

Unfolding the definition, (Free Formula Independence) is:

$$\text{If } K' \cup \{\alpha\} \vdash \bot \text{ for no consistent subset } K' \text{ of } K \quad (2)$$
$$\text{then } I(K \cup \{\alpha\}) = I(K)$$

[1] For simplicity, we use a language based on the complete set of connectives $\{\neg, \wedge, \vee\}$.

[2] In the conclusion, we mention the case of multisets.

[3] A formula φ is free for X iff $Y \cup \{\alpha\} \vdash \bot$ for no consistent subset Y of X.

(Hunter and Konieczny 2010) has an example of a consistent free formula whose rightmost conjunct contradicts a *consistent* part of a formula of K and so does its leftmost conjunct. A different case (where no minimal inconsistent subset is a singleton set) is $K = \{a \wedge c, b \wedge \neg c\}$ and $\alpha = \neg a \vee \neg b$. Atoms a and b are compatible but $a \wedge b$ is contradicted by α, and $K \cup \{\alpha\}$ may be regarded as more inconsistent than K: (2) is failed.

Consequences of HK Postulates

Proposition 1 *(Monotony) entails*

- *if* $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\alpha, \beta\})$ *then* $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\beta\})$

Proof Assume $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\alpha, \beta\})$. However, (Monotony) ensures $I(K \cup \{\alpha, \beta\}) \geq I(K \cup \{\beta\}$. Hence the result. ■

That is, if I conforms with adjunction (roughly speaking, it means identifying $\{\alpha, \beta\}$ with $\{\alpha \wedge \beta\}$) then I respects the idea that adding a conjunct cannot make the amount of inconsistency to decrease.

Notation. $\alpha \equiv \beta$ denotes that both $\alpha \vdash \beta$ and $\beta \vdash \alpha$ hold. Also, $\alpha \equiv \beta \vdash \gamma$ is an abbreviation for $\alpha \equiv \beta$ and $\beta \vdash \gamma$ (so, $\alpha \equiv \beta \nvdash \gamma$ means that $\alpha \equiv \beta$ and $\beta \nvdash \gamma$).

Proposition 2 *(Free Formula Independence) entails*

- *if* $\alpha \equiv \top$ *then* $I(K \cup \{\alpha\}) = I(K)$ (Tautology Independence)

Proof A tautology is trivially a free formula for any K. ■

Unless $\beta \nvdash \bot$, there is however no guarantee that the following holds:

- if $\alpha \equiv \top$ then $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta\})$ ($\top$-conjunct Independence)

Proposition 3 *(Dominance) entails*

- $I(K \cup \{\alpha_1, \ldots, \alpha_n\}) = I(K \cup \{\beta_1, \ldots, \beta_n\})$ *whenever* $\alpha_i \equiv \beta_i \nvdash \bot$ *for* $i = 1..n$ (Swap)

Proof For $i = 1..n$, $\alpha_i \equiv \beta_i$ so that (Dominance) can be applied in both directions. As a consequence, for $i = 1..n$, it clearly holds that $I(K \cup \{\beta_1, \ldots, \beta_{i-1}, \alpha_i, \ldots, \alpha_n\}) = I(K \cup \{\beta_1, \ldots, \beta_i, \alpha_{i+1}, \ldots, \alpha_n\})$. ■

Proposition 3 fails to guarantee that I be independent of any consistent subset of the knowledge base being replaced by an equivalent (consistent) set of formulas:

- if $K' \nvdash \bot$ and $K' \equiv K''$ then $I(K \cup K') = I(K \cup K'')$ (Exchange)

Proposition 3 guarantees that any consistent formula of the knowledge base can be replaced by an equivalent formula without altering the result of the inconsistency measure. Clearly, postulates for inconsistency measures are expected *not* to entail $I(K \cup \{\alpha\}) = I(K \cup \{\beta\})$ for $\alpha \equiv \beta \vdash \bot$. However, some subcases are desirable: $I(K \cup \{\alpha \vee \alpha\}) = I(K \cup \{\alpha\})$, $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta \wedge \alpha\})$, and so on, in full generality (i.e., even for $\alpha \vdash \bot$) but Proposition 3 fails to ensure any of these.

Proposition 4 *(Dominance) entails*

- *if* $\alpha \wedge \beta \nvdash \bot$ *then* $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\beta\})$

Proof Apply (Dominance) to the valid inference $\alpha \wedge \beta \vdash \beta$ and the result ensues. ■

Proposition 4 means that I respects the idea that adding a conjunct cannot make amount of inconsistency to decrease, in the case of a consistent conjunction (however, one really wonders why this not guaranteed to hold in more cases?).

Proposition 5 *Due to (Dominance) and (Monotony)*

- *For* $\alpha \in K$, *if* $\alpha \nvdash \bot$ *and* $\alpha \vdash \beta$ *then* $I(K \cup \{\beta\}) = I(K)$

Proof $I(K \cup \{\alpha\}) = I(K)$ as $\alpha \in K$. By (Dominance), $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$. Therefore, $I(K) \geq I(K \cup \{\beta\})$. The converse holds due to (Monotony). ■

Proposition 5 guarantees that a consequence of a consistent formula of the knowledge base can be added without altering the result of the inconsistency measure. What about a consequence of a consistent subset of the knowledge base? Indeed, Proposition 5 is a special case of

(A_n) For $\{\alpha_1, \ldots, \alpha_n\} \subseteq K$, if $\{\alpha_1, \ldots, \alpha_n\} \nvdash \bot$ and $\{\alpha_1, \ldots, \alpha_n\} \vdash \beta$ then $I(K \cup \{\beta\}) = I(K)$

That is, Proposition 5 guarantees (A_n) only for $n = 1$ but what is the rationale for stopping there?

Example 1 *Let* $K = \{\neg b, a \wedge b, b \wedge c\}$. *Proposition 5 ensures that* $I(K \cup \{a, c\}) = I(K \cup \{a\}) = I(K \cup \{c\}) = I(K)$. *Although* $a \wedge c$ *behaves as* a *and* c *with respect to all contradictions in* K *(i.e.,* $a \wedge b$ *vs* $\neg b$ *and* $b \wedge c$ *vs* $\neg b$*), HK postulates fail to ensure* $I(K \cup \{a \wedge c\}) = I(K)$.

Replacement of Equivalent Subsets

The value of (Exchange)

Firstly, (Exchange) is not a consequence of (Dominance) and (Monotony). An example is $K_1 = \{a \wedge c \wedge e, b \wedge d \wedge \neg e\}$ and $K_2 = \{a \wedge e, c \wedge e, b \wedge d \wedge \neg e\}$. Due to (Exchange), $I(K_1) = I(K_2)$ but HK postulates do not impose equality. Next are a few results showing properties of (Exchange).

Proposition 6 *(Exchange) is equivalent to each of these:*

- *The family* $(A_n)_{n \geq 1}$
- *If* $K' \equiv K''$, $K' \nvdash \bot$ *then* $I(K \cup K') = I((K \setminus K') \cup K'')$
- *If* $K' \equiv K''$ *and* $K' \nvdash \bot$ *and* $K \cap K' = \emptyset$ *then* $I(K \cup K') = I(K \cup K'')$
- *If* $\{K_1, \ldots, K_n\}$ *is a partition of* $K \setminus K_0$ *where* K_0 *is defined as* $K_0 = \{\alpha \in K \mid \alpha \vdash \bot\}$ *such that* $K_i \nvdash \bot$ *and* $K'_i \equiv K_i$ *for* $i = 1..n$ *then* $I(K) = I(K_0 \cup K'_1 \cup \cdots \cup K'_n)$

Proof Assume (A_n) for all $n \geq 1$ and $K' \equiv K'' \nvdash \bot$. (i) Let $K' = \{\alpha_1, \ldots, \alpha_m\}$. Define $\langle K'_j \rangle_{j \geq 0}$ where $K'_0 = K \cup K''$ and $K'_{j+1} = K'_j \cup \{\alpha_{j+1}\}$. It is clear that $K'' \nvdash \bot$ and $K'' \vdash \alpha_{j+1}$ and $K'' \subseteq K'_j$. Hence, (A_n) can be applied to K'_j and this gives $I(K'_j) = I(K'_j \cup \{\alpha_{j+1}\}) = I(K'_{j+1})$. Overall, $I(K'_0) = I(K'_m)$. I.e., $I(K \cup K'') = I(K \cup K' \cup K'')$. (ii) Let $K'' = \{\beta_1, \ldots, \beta_p\}$. Consider the sequence $\langle K''_j \rangle_{j \geq 0}$ where $K''_0 = K \cup K'$ and $K''_{j+1} = K''_j \cup \{\beta_{j+1}\}$. Clearly, $K' \nvdash \bot$ and $K' \vdash \beta_{j+1}$

and $K' \subseteq K''_j$. Hence, (A_n) can be applied to K''_j and this gives $I(K''_j) = I(K''_j \cup \{\beta_{j+1}\}) = I(K''_{j+1})$. Overall, $I(K''_0) = I(K''_p)$. I.e., $I(K \cup K') = I(K \cup K' \cup K'')$. Combining the equalities, $I(K \cup K') = I(K \cup K'')$. That is, the family $(A_n)_{n\geq 1}$ entails (Exchange).
We now show that the family $(A_n)_{n\geq 1}$ is entailed by the second item in the statement of Proposition 6, denoted (Exchange$'$), which is:

$$\begin{array}{l}\text{If } K' \not\vdash \bot \text{ and } K' \equiv K''\}\\ \text{then } I(K \cup K') = I((K \setminus K') \cup K'')\end{array}$$

Let $\{\alpha_1, \ldots, \alpha_n\} \subseteq K$ such that $\{\alpha_1, \ldots, \alpha_n\} \not\vdash \bot$ and $\{\alpha_1, \ldots, \alpha_n\} \vdash \beta$. So, $\{\alpha_1, \ldots, \alpha_n\} \equiv \{\alpha_1, \ldots, \alpha_n, \beta\}$. For $K' = \{\alpha_1, \ldots, \alpha_n\}$, $K'' = \{\alpha_1, \ldots, \alpha_n, \beta\}$ (Exchange) gives $I(K) = I((K \setminus \{\alpha_1, \ldots, \alpha_n\}) \cup \{\alpha_1, \ldots, \alpha_n, \beta\} = I(K \cup \{\beta\})$.
By transitivity, we have thus shown that (Exchange) is entailed by (Exchange$'$). Since the converse is obvious, the equivalence between (Exchange), (Exchange$'$) and the family $(A_n)_{n\geq 1}$ holds.

It is clear that the third item in the statement of Proposition 6 is equivalent with (Exchange).

Consider now (Exchange$''$), the last item in the statement of Proposition 6:

$$\begin{array}{r}\text{If } \{K_1, \ldots, K_n\} \text{ is a partition of } K \setminus K_0 \text{ where}\\ K_0 = \{\alpha \in K \mid \alpha \vdash \bot\} \text{ such that}\\ K_i \not\vdash \bot \text{ and } K'_i \equiv K_i \text{ for } i = 1..n \text{ then}\\ I(K) = I(K_0 \cup K'_1 \cup \cdots \cup K'_n).\end{array}$$

(i) Assume (Exchange$'$). We now prove (Exchange$''$). Let $\{K_1, \ldots, K_n\}$ be a partition of $K \setminus K_0$ satisfying the conditions of (Exchange$''$). Trivially, $I(K) = I(K_0 \cup K \setminus K_0) = I(K_0 \cup K_1 \cup \cdots \cup K_n)$. Then, $K_i \setminus K_n = K_i$ for $i = 1..n-1$. Applying (Exchange$'$) yields $I(K_0 \cup K_1 \cup \cdots \cup K_n) = I(K_0 \cup K_1 \cup \cdots \cup K'_n)$ hence $I(K) = I(K_0 \cup K_1 \cup \cdots \cup K'_n)$. Applying (Exchange$'$) iteratively upon K_{n-1}, K_{n-2}, …, K_1 gives $I(K) = I(K_0 \cup K'_1 \cup \cdots \cup K'_n)$.
(ii) Assume (Exchange$''$). We now prove (Exchange$'$). Let $K' \not\vdash \bot$ and $K'' \equiv K'$. Clearly, $(K \cup K')_0 = K_0$ and $(K \cup K') \setminus (K \cup K')_0 = (K \setminus K_0) \cup K'$. As each formula in $K \setminus K_0$ is consistent, $K \setminus K_0$ can be partitioned into $\{K_1, \ldots, K_n\}$ such that $K_i \not\vdash \bot$ for $i = 1..n$ (take $n = 0$ in the case that $K = K_0$). Then, $\{K_1 \setminus K', \ldots, K_n \setminus K', K'\}$ is a partition of $(K \setminus K_0) \cup K'$ satisfying the conditions in (Exchange$''$). Now, $I(K \cup K') = I(K_0 \cup (K_1 \setminus K') \cup \cdots \cup (K_n \setminus K') \cup K')$. Applying (Exchange$''$) with each K_i substituting itself and K'' substituting K', we obtain $I(K \cup K') = I(K_0 \cup (K_1 \setminus K') \cup \cdots \cup (K_n \setminus K') \cup K'')$. That is, $I(K \cup K') = I((K \setminus K') \cup K'')$. ■

Proposition 7 *(Exchange) entails (Swap).*

Proof Taking advantage of transitivity of equality, it will be sufficient to prove $I(K \cup \{\beta_1, \ldots, \beta_{i-1}, \alpha_i, \ldots, \alpha_n\}) = I(K \cup \{\beta_1, \ldots, \beta_i, \alpha_{i+1}, \ldots, \alpha_n\})$ for $i = 1..n$. Due to $\alpha_i \equiv \beta_i$ and $\beta_i \not\vdash \bot$, it holds that $\{\alpha_i\} \not\vdash \bot$ and $\{\alpha_i\} \equiv \{\alpha_i, \beta_i\}$. As a consequence, (Exchange) can be applied to $K \cup \{\beta_1, \ldots, \beta_{i-1}, \alpha_{i+1}, \ldots, \alpha_n\}$ for $K' = \{\alpha_i\}$ and $K'' = \{\alpha_i, \beta_i\}$. Accordingly, $I(K \cup \{\beta_1, \ldots, \beta_{i-1}, \alpha_i, \ldots, \alpha_n\})$ is then equal to $I(((K \cup \{\beta_1, \ldots, \beta_{i-1}, \alpha_{i+1}, \ldots, \alpha_n\}) \setminus \{\alpha_i\}) \cup \{\alpha_i, \beta_i\})$ and the latter is $I(K \cup \{\beta_1, \ldots, \beta_i, \alpha_{i+1}, \ldots, \alpha_n\})$. ■

That (Exchange) entails (Swap) is natural. Surprisingly, (Exchange) also entails (Tautology Independence).

Proposition 8 *(Exchange) gives (Tautology Independence).*

Proof The non-trivial case is $\alpha \notin K$. Apply (Exchange$'$) for $K' = \{\alpha\}$ and $K'' = \emptyset$, so, $I(K \cup \{\alpha\}) = I((K \setminus \{\alpha\}) \cup \emptyset)$ ensues. I.e., $I(K \cup \{\alpha\}) = I(K)$. ■

The value of an adjunction postulate

In keeping with the meaning of the conjunction connective in classical logic, consider a dedicated postulate in the form

- $I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha \wedge \beta\})$
(Adjunction Invariancy)

Proposition 9 *(Adjunction Invariancy) entails*

- $I(K \cup \{\alpha, \beta\}) = I((K \setminus \{\alpha, \beta\}) \cup \{\alpha \wedge \beta\})$
(Disjoint Adjunction Invariancy)
- $I(K) = I(\{\bigwedge K\})$ (Full Adjunction Invariancy)
where $\bigwedge K$ *denotes* $\alpha_1 \wedge \ldots \wedge \alpha_n$ *for any enumeration* $\alpha_1, \ldots, \alpha_n$ *of* K.

Proof Let $K = \{\alpha_1, \ldots, \alpha_n\}$. Apply iteratively (Adjunction Invariancy) as $I(\{\alpha_1 \wedge \ldots \wedge \alpha_{i-1}, \alpha_i, \ldots, \alpha_n\}) = I(\{\alpha_1 \wedge \ldots \wedge \alpha_i, \alpha_{i+1}, \ldots, \alpha_n\})$ for $i = 2..n$. ■

Proposition 10 *Assuming* $I(\{\alpha \wedge (\beta \wedge \gamma)\}) = I(\{(\alpha \wedge \beta) \wedge \gamma\})$ *and* $I(\{\alpha \wedge \beta\}) = I(\{\beta \wedge \alpha\})$*, (Disjoint Adjunction Invariancy) and (Full Adjunction Invariancy) are equivalent.*

Proof Assume (Full Adjunction Invariancy). $K \cup \{\alpha, \beta\} = (K \setminus \{\alpha, \beta\}) \cup \{\alpha, \beta\}$ yields $I(K \cup \{\alpha, \beta\}) = I((K \setminus \{\alpha, \beta\}) \cup \{\alpha, \beta\})$. By (Full Adjunction Invariancy), $I((K \setminus \{\alpha, \beta\}) \cup \{\alpha, \beta\}) = I(\{\bigwedge((K \setminus \{\alpha, \beta\}) \cup \{\alpha, \beta\})\})$ and the latter can be written $I(\{\gamma_1 \wedge \ldots \wedge \gamma_n \wedge \alpha \wedge \beta\})$ for some enumeration $\gamma_1, \ldots, \gamma_n$ of $K \setminus \{\alpha, \beta\}$. I.e., $I(K \cup \{\alpha, \beta\}) = I(\{\gamma_1 \wedge \ldots \wedge \gamma_n \wedge \alpha \wedge \beta\})$. By (Full Adjunction Invariancy), $I((K \setminus \{\alpha, \beta\}) \cup \{\alpha \wedge \beta\}) = I(\{\bigwedge((K \setminus \{\alpha, \beta\}) \cup \{\alpha \wedge \beta\})\})$ that can be written $I(\{\gamma_1 \wedge \ldots \wedge \gamma_n \wedge \alpha \wedge \beta\})$ for the same enumeration $\gamma_1, \ldots, \gamma_n$ of $K \setminus \{\alpha, \beta\}$. So, $I(K \cup \{\alpha, \beta\}) = I((K \setminus \{\alpha, \beta\}) \cup \{\alpha \wedge \beta\})$. As to the converse, it is trivial to use (Disjoint Adjunction Invariancy) iteratively to get (Full Adjunction Invariancy). ■

A counter-example to the purported equivalence of (Adjunction Invariancy) and (Full Adjunction Invariancy) is as follows. Let $K = \{a, b, \neg b \wedge \neg a\}$. Obviously, $I(K \cup \{a, b\}) = I(K)$ since $\{a, b\} \subseteq K$. (Full Adjunction Invariancy) gives $I(K) = I(\{\bigwedge_{\gamma \in K} \gamma\})$ i.e. $I(K \cup \{a, b\}) = I(\{\bigwedge_{\gamma \in K} \gamma\}) = I(\{a \wedge b \wedge \neg b \wedge \neg a\})$. A different case of applying (Full Adjunction Invariancy) gives $I(K \cup \{a \wedge b\}) = I(\{\bigwedge_{\gamma \in K \cup \{a \wedge b\}} \gamma\}) = I(\{a \wedge b \wedge \neg b \wedge \neg a \wedge a \wedge b\})$. However, HK postulates do not provide grounds to infer $I(\{a \wedge b \wedge \neg b \wedge \neg a\}) = I(\{a \wedge b \wedge \neg b \wedge \neg a \wedge a \wedge b\})$ hence (Adjunction Invariancy) may fail here.

(Adjunction Invariancy) provides a natural equivalence between (Monotony) and a principle which expresses that adding a conjunct cannot make the amount of inconsistency to decrease:

Proposition 11 *Assuming (Consistency Null), (Adjunction Invariancy) yields that (Monotony) is equivalent with*

- $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\alpha\})$ (Conjunction Dominance)

Proof Assume (Monotony), a simple instance of which is $I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha, \beta\})$. (Adjunction Invariancy) gives $I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha \wedge \beta\})$. As a consequence, $I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha \wedge \beta\})$. This inequality shows that (Conjunction Dominance) holds.
Assume (Conjunction Dominance). First, consider $K \neq \emptyset$. Let $\alpha \in K$. Thus, $I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha \wedge \beta\}$ by (Conjunction Dominance). (Adjunction Invariancy) gives $I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha \wedge \beta\})$. Hence, $I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha, \beta\})$. I.e., $I(K) \leq I(K \cup \{\beta\})$ because $\alpha \in K$. For $K' \in \mathcal{K}_{\mathcal{L}}$, it is enough to iterate this finitely many times (one for every β in $K' \setminus K$) to obtain $I(K) \leq I(K \cup K')$. Now, consider $K = \emptyset$. By (Consistency Null), $I(K) = 0$ hence $I(K) \leq I(K \cup K')$. ■

(Free Formula Independence) yields (Tautology Independence) by Proposition 2 although a more general principle (e.g., ($\top$-conjunct Independence) or the like) ensuring that I be independent of tautologies is to be expected. The next result shows that (Adjunction Invariancy) is the way to get both postulates at once.

Proposition 12 *Assuming (Consistency Null), (Adjunction Invariancy) yields that ($\top$-conjunct Independence) and (Tautology Independence) are equivalent.*

Proof For $\alpha \equiv \top$, (Adjunction Invariancy) and (Tautology Independence) give $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\alpha, \beta\}) = I(K \cup \{\beta\})$. As to the converse, let $\beta \in K$. Therefore, $I(K) = I(K \cup \{\beta\}) = I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha\})$. At to the case $K = \emptyset$, it is settled by means of (Consistency Null). ■

(Adjunction Invariancy) provides for free various principles related to (idempotence, commutativity, and associativity of) conjunction as follows.

Proposition 13 *(Adjunction Invariancy) entails*

- $I(K \cup \{\alpha \wedge \alpha\}) = I(K \cup \{\alpha\})$
- $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta \wedge \alpha\})$
- $I(K \cup \{\alpha \wedge (\beta \wedge \gamma)\}) = I(K \cup \{(\alpha \wedge \beta) \wedge \gamma\})$

Proof (i) $I(K \cup \{\alpha \wedge \alpha\}) = I(K \cup \{\alpha, \alpha\}) = I(K \cup \{\alpha\})$. (ii) $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\alpha, \beta\}) = I(K \cup \{\beta, \alpha\}) = I(K \cup \{\beta \wedge \alpha\})$. (iii) $I(K \cup \{\alpha \wedge (\beta \wedge \gamma)\}) = I(K \cup \{\alpha, \beta \wedge \gamma\}) = I(K \cup \{\alpha, \beta, \gamma\}) = I(K \cup \{\alpha \wedge \beta, \gamma\}) = I(K \cup \{(\alpha \wedge \beta) \wedge \gamma\})$. ■

(Adjunction Invariancy) and (Exchange) are two principles devoted to ensuring that replacing a subset of the knowledge base with an equivalent subset does not change the value given by the inconsistency measure. The contexts that these two principles require for the replacement to be safe differ:

1. For $K' \not\vdash \bot$, (Exchange) is more general than (Adjunction Invariancy) since (Exchange) guarantees $I(K \cup K') = I(K \cup K'')$ for every $K'' \equiv K'$ but (Adjunction Invariancy) ensures it only for $K'' = \{\bigwedge K'_i \mid \mathfrak{K} = \{K'_1, .., K'_n\}\}$ where $\mathfrak{K}$ ranges over the partitions of K'.
2. For $\alpha \vdash \bot$, (Adjunction Invariancy) is more general than (Exchange) because (Adjunction Invariancy) guarantees $I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha \wedge \beta\})$ but (Exchange) does not guarantee it.

Revisiting HK Postulates

Sticking with (Consistency Null) and (Monotony)

First, (Consistency Null) or a like postulate is indispensable because there seems to be no way to have a sensible inconsistency measure that would not be able to always discriminate between consistency and inconsistency.

(Monotony) is to be kept since contradictions in classical logic (and basically all logics) are monotone (Besnard 2010) wrt information: That is, extra information cannot make a contradiction to vanish.

We will not retain (Monotony) as an explicit postulate, because it ensues from our schematic postulate (see later).

Intended postulates

(Tautology Independence) and ($\top$-conjunct Independence) are due postulates. More generally, it would make no sense, when determining how inconsistent a theory is, to take into account any inessential difference in which a formula can be written (e.g., $\alpha \vee \beta$ instead of $\beta \vee \alpha$). Define α' to be a *prenormal form* of α if α' is obtained from α by applying (posibly repeatedly) one or more of the following principles: commutativity, associativity and distribution for $\wedge$ and $\vee$, De Morgan laws, double negation equivalence. Henceforth the next[4] postulate:

- If β is a prenormal form of α, $I(K \cup \{\alpha\}) = I(K \cup \{\beta\})$ (Rewriting)

As (Monotony) essentially means that extra information cannot make amount of inconsistency to decrease, the same idea must apply to conjunction because $\alpha \wedge \beta$ cannot involve less information than α. Thus, another due postulate is:

- $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\alpha\})$ (Conjunction Dominance)

Indeed, it does not matter whether α or β or both be inconsistent: It definitely cannot be rational to hold that there is a case (even a single one) where extending K with a conjunction would result in *less* inconsistency than extending K with one of the conjuncts.

Taking care of disjunction

It is very difficult to assess how inconsistent a disjunction is, but bounds can be set. Indeed, a disjunction expresses two alternative possibilities; so, accrual across these would make little sense. That is, amount of inconsistency in $\alpha \vee \beta$ cannot exceed amount of inconsistency in either α or β, depending on which one involves a higher amount of inconsistency. Hence the following postulate.

[4] Insharp contrast to (Irrelevance of Syntax) that allows for destructive transformation from α to β when both are inconsistent, (Rewriting) takes care of inhibiting purely deductive transformations (the most important one is presumably from $\alpha \wedge \bot$ to $\bot$).

- $I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$
(Disjunct Maximality)

Two alternative formulations for (Disjunct Maximality) are as follows.

Proposition 14 *Assume $I(K \cup \{\alpha \vee \beta\}) = I(K \cup \{\beta \vee \alpha\})$. (Disjunct Maximality) is equivalent with each of*

- *if $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$ then $I(K \cup \{\alpha\}) \geq I(K \cup \{\alpha \vee \beta\})$*
- *either $I(K \cup \{\alpha \vee \beta\}) \leq I(K \cup \{\alpha\})$ or $I(K \cup \{\alpha \vee \beta\}) \leq I(K \cup \{\beta\})$*

Proof Let us prove that (Disjunct Maximality) entails the first item. Assume $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$. I.e., $I(K \cup \{\alpha\}) = \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$. Using (Disjunct Maximality), $I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$, i.e. $I(K \cup \{\alpha\})) \geq I(K \cup \{\alpha \vee \beta\})$. As to the converse direction, assume that if $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$ then $I(K \cup \{\alpha\}) \geq I(K \cup \{\alpha \vee \beta\})$. Consider the case $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) = I(K \cup \{\alpha\})$. Hence, $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$. According to the assumption, it follows that $I(K \cup \{\alpha\}) \geq I(K \cup \{\alpha \vee \beta\})$. That is, $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) \geq I(K \cup \{\alpha \vee \beta\})$. Similarly, the case $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) = I(K \cup \{\beta\})$ gives $I(K \cup \{\beta\}) \geq I(K \cup \{\beta \vee \alpha\})$. Then, $I(K \cup \{\beta\}) \geq I(K \cup \{\alpha \vee \beta\})$ in view of the hypothesis in the statement of Proposition 14. That is, $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) \geq I(K \cup \{\alpha \vee \beta\})$. Combining both cases, (Disjunct Maximality) holds.
The equivalence of (Disjunct Maximality) with the last item is due to the fact that the codomain of I is totally ordered. ■

Although it is quite unclear how to weigh inconsistencies out of a disjunction, they must weigh no more than out of both disjuncts (whether tied together by a conjunction or not), which is the reason for holding

- $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\alpha \vee \beta\})$
($\wedge$-over-$\vee$ Dominance)

and its conjunction-free counterpart

- $I(K \cup \{\alpha, \beta\}) \geq I(K \cup \{\alpha \vee \beta\})$

Proposition 15 *Assume $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta \wedge \alpha\})$. (Conjunction Dominance) and (Disjunct Maximality) entail ($\wedge$-over-$\vee$ Dominance).*

Proof Given $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta \wedge \alpha\})$, (Conjunction Dominance) gives $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\alpha\})$ and $I(K \cup \{\alpha \wedge \beta\}) \geq I(K \cup \{\beta\})$. Therefore, $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) \leq I(K \cup \{\alpha \wedge \beta\})$. In view of (Disjunct Maximality), $I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$, and it accordingly follows that $I(K \cup \{\alpha \vee \beta\}) \leq I(K \cup \{\alpha \wedge \beta\})$ holds. ■

Proposition 16 *(Monotony) and (Disjunct Maximality) entail*

- $I(K \cup \{\alpha, \beta\}) \geq I(K \cup \{\alpha \vee \beta\})$

Proof Due to (Monotony), $I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha, \beta\})$ and $I(K \cup \{\beta\}) \leq I(K \cup \{\alpha, \beta\})$. As a consequence, $\max(I(K \cup \{\alpha\}), I(K \cup \{\beta\})) \leq I(K \cup \{\alpha, \beta\})$. Then, $I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$ due to (Disjunct Maximality). $I(K \cup \{\alpha, \beta\}) \geq I(K \cup \{\alpha \vee \beta\})$ easily ensues. ■

A schematic postulate

This is to be presented in two steps.

1. (Monotony) expresses that adding information cannot result in a decrease of the amount of inconsistency in the knowledge base. Considering a notion of primitive conflicts that underlies amount of inconsistency, (Monotony) is a special case of a postulate stating that amount of inconsistency is monotone with respect to the set of primitive conflicts $\mathcal{C}(K)$ of the knowledge base K: If $\mathcal{C}(K) \subseteq \mathcal{C}(K')$ then $I(K) \leq I(K')$.
Clearly, I is to admit different postulates depending on what features are required for primitive conflicts (see Table 1).
2. Keep in mind that an inconsistency measure refers to logical content of the knowledge base, *not* other aspects whether subject matter of contradiction, source of information,... This is because an inconsistency measure is only concerned with *quantity*, i.e. amount of inconsistency (of course, it is possible for example that a contradiction be more worrying than another -and so, making more pressing *to act* (Gabbay and Hunter 1993) about it- but this has nothing to do with amount of inconsistency). Now, what characterizes logical content is uniform substitutivity. Hence a postulate called (Substitutivity Dominance) stating that renaming cannot make the amount of inconsistency to decrease: If $\sigma K = K'$ for some substitution σ then $I(K) \leq I(K')$.

Combining these two ideas, we obtain the next postulate

- If $\mathcal{C}(\sigma K) \subseteq \mathcal{C}(K')$ for some substitution σ, $I(K) \leq I(K')$
(Subsumption Orientation)

Fact 1 *Every postulate of the form*

- *$I(X) \leq I(Y)$ for all $X \in \mathcal{K}_\mathcal{L}$ and $Y \in \mathcal{K}_\mathcal{L}$ such that condition $C_{X,Y}$ holds*

or of the form

- *$I(X) = I(Y)$ for all $X \in \mathcal{K}_\mathcal{L}$ and $Y \in \mathcal{K}_\mathcal{L}$ such that condition $C_{X,Y}$ holds*

is derived from (Subsumption Orientation) and from any property of $\mathcal{C}$ ensuring that condition C holds.

Individual properties of $\mathcal{C}$ ensuring condition C for a number of postulates, including all those previously mentioned in the paper, can be found in Table 1.

(Variant Equality) in Table 1 is named after the notion of a variant (Church 1956):

- If σ and σ' are substitutions s.t. $\sigma K = K'$ and $\sigma' K' = K$ then $I(K) = I(K')$
(Variant Equality)

New system of postulates (basic and strong versions)

All the above actually suggests a new system of postulates, which consists simply of (Consistency Null) and (Subsumption Orientation). The system is parameterized by the properties imposed upon $\mathcal{C}$ in the latter. In the range induced by $\mathcal{C}$, a basic system emerges, which amounts to the next list:

Specific property for $\mathcal{C}$	*Specific postulate entailed by* (*Subsumption Orientation*)
No property needed	(Variant Equality)
No property needed	(Substitutivity Dominance)
$\mathcal{C}(K \cup \{\alpha\}) = \mathcal{C}(K)$ for $\alpha \equiv \top$	(Tautology Independence)
$\mathcal{C}(K \cup \{\alpha \wedge \beta\}) = \mathcal{C}(K \cup \{\beta\})$ for $\alpha \equiv \top$	($\top$-conjunct Independence)
$\mathcal{C}(K \cup \{\alpha\}) = \mathcal{C}(K \cup \{\alpha'\})$ for α' prenormal form of α	(Rewriting)
$\mathcal{C}(K) \subseteq \mathcal{C}(K \cup \{\alpha\})$	(Instance Low)
$\mathcal{C}(K) \subseteq \mathcal{C}(K \cup \{\alpha\})$	(Monotony)
$\mathcal{C}(K \cup \{\alpha \vee \beta\}) \subseteq \mathcal{C}(K \cup \{\alpha \wedge \beta\})$	($\wedge$-over-$\vee$ Dominance)
$\mathcal{C}(K \cup \{\alpha\}) \subseteq \mathcal{C}(K \cup \{\alpha \wedge \beta\})$	(Conjunction Dominance)
$\mathcal{C}(K \cup \{\alpha, \beta\}) = \mathcal{C}(K \cup \{\alpha \wedge \beta\})$	(Adjunction Invariancy)
$\mathcal{C}(K \cup \{\alpha \vee \beta\}) \subseteq \mathcal{C}(K \cup \{\alpha\})$ or $\mathcal{C}(K \cup \{\beta\})$	(Disjunct Maximality)
$\mathcal{C}(K \cup \{\alpha \vee \beta\}) \supseteq \mathcal{C}(K \cup \{\alpha\})$ or $\mathcal{C}(K \cup \{\beta\})$	(Disjunct Minimality)
$\mathcal{C}(K \cup K') = \mathcal{C}(K \cup K'')$ for $K'' \equiv K' \not\vdash \bot$	(Exchange)
$\mathcal{C}(K \cup \{\alpha_1, ..., \alpha_n\}) = \mathcal{C}(K \cup \{\beta_1, .., \beta_n\})$ if $\alpha_i \equiv \beta_i \not\vdash \bot$	(Swap)
$\mathcal{C}(K \cup \{\beta\}) \subseteq \mathcal{C}(K \cup \{\alpha\})$ for $\alpha \vdash \beta$ and $\alpha \not\vdash \bot$	(Dominance)
$\mathcal{C}(K \cup \{\alpha\}) = \mathcal{C}(K)$ for α free for K	(Free Formula Independence)

Table 1: Conditions for postulates derived from (Subsumption Orientation).

Basic System

$I(K) = 0$ iff $K \not\vdash \bot$ (Consistency Null)
If α' is a prenormal form of α
then $I(K \cup \{\alpha\}) = I(K \cup \{\alpha'\})$ (Rewriting)
If $\sigma K \subseteq K'$ for some substitution σ
then $I(K) \leq I(K')$ (Instance Low)
$I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$ (Disjunct Maximality)
If $\alpha \equiv \top$ then $I(K) = I(K \cup \{\alpha\})$ (Tautology Independence)
If $\alpha \equiv \top$ then $I(K \cup \{\alpha \wedge \beta\}) = I(K \cup \{\beta\})$ ($\top$-conjunct Independence)
$I(K \cup \{\alpha\}) \leq I(K \cup \{\alpha \wedge \beta\})$ (Conjunction Dominance)

At the other end of the range is the strong system below (except for (Dominance) and (Free Formula Independence), it captures all postulates listed in Table 1).

Strong System

$I(K) = 0$ iff $K \not\vdash \bot$ (Consistency Null)
If α' is a prenormal form of α
then $I(K \cup \{\alpha\}) = I(K \cup \{\alpha'\})$ (Rewriting)
If $\sigma K \subseteq K'$ for some substitution σ
then $I(K) \leq I(K')$ (Instance Low)
$I(K \cup \{\alpha \vee \beta\}) \leq \max(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$ (Disjunct Maximality)
$I(K \cup \{\alpha \vee \beta\}) \geq \min(I(K \cup \{\alpha\}), I(K \cup \{\beta\}))$ (Disjunct Minimality)
If $K'' \equiv K' \not\vdash \bot$ then $I(K \cup K') = I(K \cup K'')$ (Exchange)
$I(K \cup \{\alpha, \beta\}) = I(K \cup \{\alpha \wedge \beta\})$ (Adjunction Invariancy)

HK Postulates as (Subsumption Orientation)

Time has come to make sense[5] of the HK choice of (Free Formula Independence) together with (Monotony), by means of Theorem 1 and Theorem 2.

[5] Still not defending the choice of (Free Formula Independence).

Theorem 1 *Let $\mathcal{C}$ be such that for every $K \in \mathcal{K}_\mathcal{L}$ and for every $X \subseteq \mathcal{L}$ which is minimal inconsistent, $X \in \mathcal{C}(K)$ iff $X \subseteq K$. If I satisfies both (Monotony) and (Free Formula Independence) then I satisfies (Subsumption Orientation) restricted to its non-substitution part, namely*

$$\text{if } \mathcal{C}(K) \subseteq \mathcal{C}(K') \text{ then } I(K) \leq I(K').$$

Proof Let $\mathcal{C}(K) \subseteq \mathcal{C}(K')$. Should K be a subset of K', (Monotony) yields $I(K) \leq I(K')$ as desired. So, let us turn to $K \not\subseteq K'$. Consider $\varphi \in K \setminus K'$. If φ were not free for K, there would exist a minimal inconsistent subset X of K such that $\varphi \in X$. Clearly, $X \not\subseteq K'$. The constraint imposed on $\mathcal{C}$ in the statement of the theorem would then yield both $X \in \mathcal{C}(K)$ and $X \notin \mathcal{C}(K')$, contradicting the assumption $\mathcal{C}(K) \subseteq \mathcal{C}(K')$. Hence, φ is free for K. In view of (Free Formula Independence), $I(K) = I(K \setminus \{\varphi\})$. The same reasoning applied to all the (finitely many) formulas in $K \setminus K'$ gives $I(K) = I(K \cap K')$. However, $K \cap K'$ is a subset of K' so that using (Monotony) yields $I(K \cap K') \leq I(K')$ hence $I(K) \leq I(K')$. ■

Define $\Xi = \{X \in \mathcal{K}_\mathcal{L} \mid \forall X' \subseteq X, X' \vdash \bot \Leftrightarrow X = X'\}$. Then, $\mathcal{C}$ is said to be *governed by minimal inconsistency* iff $\mathcal{C}$ satisfies the following property

$$\text{if } \mathcal{C}(K) \cap \Xi \subseteq \mathcal{C}(K') \cap \Xi \text{ then } \mathcal{C}(K) \subseteq \mathcal{C}(K').$$

It means that those Z in $\mathcal{C}(K)$ which are not minimal inconsistent cannot override set-inclusion induced by minimal inconsistent subsets —i.e., no such Z can, individually or collectively, turn $\mathcal{C}(K) \cap \Xi \subseteq \mathcal{C}(K') \cap \Xi$ into $\mathcal{C}(K) \not\subseteq \mathcal{C}(K')$.

Theorem 2 *Let $\mathcal{C}$ be governed by minimal inconsistency and be such that for all $K \in \mathcal{K}_\mathcal{L}$ and all $X \subseteq \mathcal{L}$ which is minimal inconsistent, $X \in \mathcal{C}(K)$ iff $X \subseteq K$. I satisfies (Monotony) and (Free Formula Independence) whenever I satisfies (Subsumption Orientation) restricted to its non-substitution part, namely*

$$\text{if } \mathcal{C}(K) \subseteq \mathcal{C}(K') \text{ then } I(K) \leq I(K').$$

Proof Trivially, if $X \subseteq K$ then $X \subseteq K \cup \{\alpha\}$. By the constraint imposed on $\mathcal{C}$ in the statement of the theorem, it follows that if $X \in \mathcal{C}(K)$ then $X \in \mathcal{C}(K \cup \{\alpha\})$. Since $\mathcal{C}$ is governed by minimal inconsistency, $\mathcal{C}(K) \subseteq \mathcal{C}(K \cup \{\alpha\})$ ensues and (Subsumption Orientation) yields (Monotony). Let α be a free formula for K. By definition, α is in no minimal inconsistent subset of $K \cup \{\alpha\}$. So, $X \subseteq K$ iff $X \subseteq K \cup \{\alpha\}$ for all minimal inconsistent X. By the constraint imposed on $\mathcal{C}$ in the statement of the theorem, $X \in \mathcal{C}(K)$ iff $X \in \mathcal{C}(K \cup \{\alpha\})$ ensues for all minimal inconsistent X. In symbols, $\mathcal{C}(K) \cap \Xi = \mathcal{C}(K \cup \{\alpha\}) \cap \Xi$. Since $\mathcal{C}$ is governed by minimal inconsistency, it follows that $\mathcal{C}(K) = \mathcal{C}(K \cup \{\alpha\})$. Thus, (Free Formula Independence) holds, due to (Subsumption Orientation). ■

These theorems mean that, *if substitutivity is left aside*, (Subsumption Orientation) is equivalent with (Free Formula Independence) and (Monotony) when primitive conflicts are essentially minimal inconsistent subsets. These postulates form a natural pair *if it is assumed that* minimal inconsistent subsets must be the basis for inconsistency measuring.

Conclusion

We have proposed a new system of postulates for inconsistency measures, i.e.

$I(K) = 0$ *iff* K *is consistent* (Consistency Null)
If $\mathcal{C}(\sigma K) \subseteq \mathcal{C}(K')$ *for a substitution* σ *then* $I(K) \leq I(K')$ (Subsumption Orientation)

parameterized by the requirements imposed on $\mathcal{C}$.

Even in its strong version, the new system omits both (Dominance) and (Free Formula Independence), which we have argued against. We have investigated various postulates, absent from the HK set, giving grounds to include them in the new system. We have shown that (Subsumption Orientation) accounts for the other postulates and provides a justification for (Free Formula Independence) together with (Monotony), through focussing on minimal inconsistent subsets.

We do not hold that the new system, in basic or strong version, captures all desirable cases, we more modestly claim for improving over the original HK set. In particular, we believe that the HK postulates suffer from over-commitment to minimal inconsistent subsets. Crucially, such a comment applies to *postulates* (they would exclude all approaches that are not based upon minimal inconsistent subsets) but it does not apply to *measures* themselves: There are excellent reasons to develop a specific measure (Knight 2002) (Mu, Liu and Jin 2012) (Jabbour and Raddaoui 2013) ...

As to future work, we must mention taking seriously belief bases as multisets –giving a counterpart to the idea that e.g. $\{a \wedge b \wedge \neg a \wedge \neg b \wedge a \wedge b \wedge \neg a \wedge \neg b\}$ might be viewed as more inconsistent than $\{a \wedge b \wedge \neg a \wedge \neg b\}$.

Acknowledgments

Many thanks to Hitoshi Omori for insightful discussions.

References

Philippe Besnard. Absurdity, Contradictions, and Logical Formalisms. *Proc. of the* 22*nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI-10)*, Arras, France, October 27-29, volume 1, pp. 369-374. IEEE Computer Society, 2010.

Alonzo Church. *Introduction to Mathematical Logic*. Princeton University Press, 1956.

Dov Gabbay and Anthony Hunter. Making Inconsistency Respectable 2: Meta-Level Handling of Inconsistent Data. *Proc. of the* 2*nd European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU'93)*, M. Clarke, R. Kruse, and S. Moral (eds.), Grenada, Spain, November 8-10, Lecture Notes in Computer Science, volume 747, pp. 129-136. Springer, 1993.

John Grant. Classifications for Inconsistent Theories. *Notre Dame Journal of Formal Logic* 19(3): 435-444, 1978.

John Grant and Anthony Hunter. Measuring Inconsistency in Knowledgebases. *Journal of Intelligent Information Systems* 27(2): 159-184, 2006.

John Grant and Anthony Hunter. Analysing Inconsistent First-Order Knowledgebases, *Artificial Intelligence* 172(8-9): 1064-1093, 2008.

John Grant and Anthony Hunter. Measuring the Good and the Bad in Inconsistent Information. *Proc. of the* 22*nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, T. Walsh (ed.), Barcelona, Catalonia, Spain, July 16-22, pp. 2632-2637. AAAI Press, 2011.

Anthony Hunter and Sébastien Konieczny. On the Measure of Conflicts: Shapley Inconsistency Values. *Artificial Intelligence* 174(14): 1007-1026, 2010.

Anthony Hunter and Sébastien Konieczny. Measuring Inconsistency through Minimal Inconsistent Sets. *Proc. of the* 11*th Conference on Principles of Knowledge Representation Reasoning (KR'08)*, Sydney, Australia, September 16-19, G. Brewka and J. Lang (eds.), pp. 358-366. AAAI Press, 2008.

Saïd Jabbour and Badran Raddaoui. Measuring Inconsistency through Minimal Proofs. *Proc. of the* 12*th European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU'13)*, L. C. van der Gaag (ed.), Utrecht, The Netherlands, July 8-10, Lecture Notes in Computer Science, volume 7958, pp. 290-301. Springer, 2013.

Kevin Knight. Measuring Inconsistency. *Journal of Philosophical Logic* 31(1): 77-98, 2002.

Kedian Mu, Weiru Liu and Zhi Jin. A General Framework for Measuring Inconsistency through Minimal Inconsistent Sets. *Knowledge Information Systems* 27(1): 85-114, 2011.

Kedian Mu, Weiru Liu and Zhi Jin. Measuring the Blame of each Formula for Inconsistent Prioritized Knowledge Bases. *Journal of Logic and Computation* 22(3): 481-516, 2012.

Matthias Thimm. Inconsistency Measures for Probabilistic Logics. *Artificial Intelligence* 197: 1-24, 2013.

Nonmonotonic Reasoning as a Temporal Activity

Daniel G. Schwartz
Department of Computer Science
Florida State University
Tallahassee, FL 32303

Abstract

A *dynamic reasoning system* (DRS) is an adaptation of a conventional formal logical system that explicitly portrays reasoning as a temporal activity, with each extralogical input to the system and each inference rule application being viewed as occurring at a distinct time step. Every DRS incorporates some well-defined logic together with a controller that serves to guide the reasoning process in response to user inputs. Logics are generic, whereas controllers are application-specific. Every controller does, nonetheless, provide an algorithm for nonmonotonic belief revision. The general notion of a DRS comprises a framework within which one can formulate the logic and algorithms for a given application and prove that the algorithms are correct, i.e., that they serve to (i) derive all salient information and (ii) preserve the consistency of the belief set. This paper illustrates the idea with ordinary first-order predicate calculus, suitably modified for the present purpose, and an example. The example revisits some classic nonmonotonic reasoning puzzles (Opus the Penguin, Nixon Diamond) and shows how these can be resolved in the context of a DRS, using an expanded version of first-order logic that incorporates typed predicate symbols. All concepts are rigorously defined and effectively computable, thereby providing the foundation for a future software implementation.

1. Introduction

This paper provide a brief overview of a longer paper that has been accepted for publication, subject to revision, as (Schwartz 2013). The full text of that paper (64 pages) may be viewed in the arXiv CoRR repository at http://arxiv.org/abs/1308.5374.

The notion of a *dynamic reasoning system* (DRS) was introduced in (Schwartz 1997) for purposes of formulating reasoning involving a logic of 'qualified syllogisms'. The idea arose in an effort to devise rules for evidence combination. The logic under study included a multivalent semantics where propositions P were assigned a probabilistic 'likelihood value' $l(P)$ in the interval $[0, 1]$, so that the likelihood value plays the role of a surrogate truth value. The situation being modeled is where, based on some evidence, P is assigned a likelihood value l_1, and then later, based on other evidence, is assigned a value l_2, and it subsequently is desired to combine these values based on some rule into a resulting value l_3. This type of reasoning cannot be represented in a conventional formal logical system with the usual Tarski semantics, since such systems do not allow that a proposition may have more than one truth value; otherwise the semantics would not be mathematically well-defined. Thus the idea arose to speak more explicitly about different occurrences of the propositions P where the occurrences are separated in time. In this manner one can construct a well-defined semantics by mapping the different time-stamped occurrences of P to different likelihood/truth values.

In turn, this led to viewing a 'derivation path' as it evolves over time as representing the knowledge base, or belief set, of a reasoning agent that is progressively building and modifying its knowledge/beliefs through ongoing interaction with its environment (including inputs from human users or other agents). It also presented a framework within which one can formulate a Doyle-like procedure for nonmonotonic 'reason maintenance' (Doyle 1979; Smith and Kelleher 1988). Briefly, if the knowledge base harbors inconsistencies due to contradictory inputs from the environment, then in time a contradiction may appear in the reasoning path (knowledge base, belief set), triggering a backtracking procedure aimed at uncovering the 'culprit' propositions that gave rise to the contradiction and disabling (disbelieving) one or more of them so as to remove the inconsistency. Accordingly the overall reasoning process may be characterized as being 'nonmonotonic'.

Reasoning is nonmonotonic when the discovery and introduction of new information causes one to retract previously held assumptions or conclusions. This is to be contrasted with classical formal logical systems, which are monotonic in that the introduction of new information (nonlogical axioms) always increases the collection of conclusions (theorems). (Schwartz 1997) contains an extensive bibliography and survey of the works related to nonmonotonic reasoning as of 1997. In particular, this includes a discussion of (i) the classic paper by McCarthy and Hayes (McCarthy and Hayes 1969) defining the 'frame problem' and describing the 'situation calculus', (ii) Doyle's 'truth maintenance system' (Doyle1979) and subsequent 'reason maintenance system' (Smith and Kelleher 1988), (iii) McCarthy's 'circumscription' (McCarthy 1980), (iv) Reiter's 'default logic' (Reiter 1980), and (v) McDermott and Doyle's 'nonmonotonic

logic' (McDermott and Doyle 1980). With regard to temporal aspects, there also are discussed works by Shoham and Perlis. (Shoham 1986; 1988) explores the idea of making time an explicit feature of the logical formalism for reasoning 'about' change, and (Shoham 1993) describes a vision of 'agent-oriented programming' that is along the same lines of the present DRS, portraying reasoning itself as a temporal activity. In (Elgot-Drapkin 1988; Elgot-Drapkin et al. 1987; 1991; Elgot-Drapkin and Perlis 1990; Miller 1993; Perlis et al. 1991) Perlis and his students introduce and study the notion of 'step logic', which represents reasoning as 'situated' in time, and in this respect also has elements in common with the notion of a DRS. Additionally mentioned but not elaborated upon in (Schwartz 1997) is the so-called AGM framework (Alchourón et al. 1985; Gardenfors 1988; 1992), named after its originators. Nonmonotonic reasoning and belief revision are related in that the former may be viewed as a variety of the latter.

These cited works are nowadays regarded as the classic approaches to nonmonotonic reasoning and belief revision. Since 1997 the AGM approach has risen in prominence, due in large part to the publication (Hansson 1999), which builds upon and substantially advances the AGM framework. AGM defines a belief set as a collection of propositions that is closed with respect to the classical consequence operator, and operations of 'contraction', 'expansion' and 'revision' are defined on belief sets. (Hansson 1999) made the important observation that a belief set can conveniently be represented as the consequential closure of a finite 'belief base', and these same AGM operations can be defined in terms of operations performed on belief bases. Since that publication, AGM has enjoyed a steadily growing population of adherents. A recent publication (Fermé and Hansson 2011) overviews the first 25 years of research in this area.

The DRS framework has elements in common with AGM, but also differs in several respects. Most importantly, the present focus is on the creation of computational algorithms that are sufficiently articulated that they can effectively be implemented in software and thereby lead to concrete applications. This element is still lacking in AGM, despite Hansson's contribution regarding finite belief bases. The AGM operations continue to be given only as set-theoretic abstractions and have not yet been translated into computable algorithms.

Another research thread that has risen to prominence is the logic-programming approach to nonmonotonic reasoning known as Answer Set Programming (or Answer Set Prolog, aka AnsProlog). A major work is the treatise (Baral 2003), and a more recent treatment is (Gelfond and Kahl 2014). This line of research develops an effective approach to nonmonotonic reasoning via an adaptation of the well-known Prolog programming language. As such, this may be characterized as a 'declarative' formulation of nonmonotoniticy, whereas the DRS approach is 'procedural'. The extent to which the two systems address the same problems has yet to be explored.

A way in which the present approach varies from the original AGM approach, but happens to agree with the views expressed by (Hansson 1999, cf. pp. 15-16), is that it dispenses with two of the original 'rationality postulates', namely, the requirements that the underlying belief set be at all times (i) consistent, and (ii) closed with respect to logical entailment. The latter is sometimes called the 'omniscience' postulate, inasmuch as the modeled agent is thus characterized as knowing all possible logical consequences of its beliefs.

These postulates are intuitively appealing, but they have the drawback that they lead to infinitary systems and thus cannot be directly implemented on a finite computer. To wit, the logical consequences of even a fairly simple set of beliefs will be infinite in number. Dropping these postulates does have anthropomorphic rationale, however, since humans themselves cannot be omniscient in the sense described, and, because of this, often harbor inconsistent beliefs without being aware of this. Thus it is not unreasonable that our agent-oriented reasoning models should have these same characteristics. Similar remarks may be found in the cited pages of (Hansson 1999).

Other ways in which the present work differs from the AGM approach may be noted. First, what is here taken as a 'belief set' is neither a belief set in the sense of AGM and Hansson nor a Hansson-style belief base. Rather it consists of the set of statements that have been input by an external agent as of some time t, together with the consequences of those statements that have been derived in accordance with the algorithms provided in a given 'controller'. Second, by labeling the statements with the time step when they are entered into the belief set (either by an external agent or derived by means of an inference rule), one can use the labels as a basis for defining the associated algorithms. Third, whereas Gärdenfors, Hansson, and virtually all others that have worked with the AGM framework, have confined their language to be only propositional, the present work takes the next step to full first-order predicate logic. This is significant inasmuch as the consistency of a finite set of propositions with respect to the classical consequence operation can be determined by truth-table methods, whereas the consistency of a finite set of statements in first-order predicate logic is undecidable (the famous result due to Gödel). For this reason the present work develops a well-defined semantics for the chosen logic and establishes a soundness theorem, which in turn can be used to establish consistency. Last, the present use of a controller is itself new, and leads to a new efficacy for applications.

The notion of a controller was not present in the previous work (Schwartz 1997). Its introduction here thus fills an important gap in that treatment. The original conception of a DRS provided a framework for modeling the reasoning processes of an artificial agent to the extent that those processes follow a well-defined logic, but it offered no mechanism for deciding what inference rules to apply at any given time. What was missing was a means to provide the agent with a sense of purpose, i.e., mechanisms for pursuing goals. This deficiency is remedied in the present treatment. The controller responds to inputs from the agent's environment, expressed as propositions in the agent's language. Inputs are classified as being of various 'types', and, depending on the input type, a reasoning algorithm is applied. Some of these algorithms may cause new propositions to be entered into

the belief set, which in turn may invoke other algorithms. These algorithms thus embody the agent's purpose and are domain-specific, tailored to a particular application. But in general their role is to ensure that (i) all salient propositions are derived and entered into to the belief set, and (ii) the belief set remains consistent. The latter is achieved by invoking a Doyle-like reason maintenance algorithm whenever a contradiction, i.e., a proposition of the form $P \wedge \neg P$, is entered into the belief set.

This recent work accordingly represents a rethinking, refinement, and extension of the earlier work, aimed at (1) providing mathematical clarity to some relevant concepts that previously were not explicitly defined, (ii) introducing the notion of a controller and spelling out its properties, and (iii) illustrating these ideas with a small collection of example applications. As such the work lays the groundwork for a software implementation of the DRS framework, this being a domain-independent software framework into which can be plugged domain-specific modules as required for any given application. Note that the mathematical work delineated in (Schwartz 2013) is a necessary prerequisite for the software implementation inasmuch as this provides the formal basis for an unambiguous set of requirements specifications. While the present work employs classical first-order predicate calculus, the DRS framework can accommodate any logic for which there exists a well-defined syntax and semantics.

The following Section 2 provides a fully detailed definition of the notion of a DRS. Section 3 briefly describes the version of first-order predicate logic introduced for the present purpose and mentions a few items needed for the ensuing discussion. Section 4 illustrates the core ideas in an application to multiple-inheritance systems, showing a new approach to resolving two classic puzzles of nonmontonic reasoning, namely Opus the Penguin and Nixon Diamond.

2. Dynamic Reasoning Systems

A *dynamic reasoning system* (DRS) comprises a model of an artificial agent's reasoning processes to the extent that those processes adhere to the principles of some well-defined logic. Formally it is comprised of a 'path logic', which provides all the elements necessary for reasoning, and a 'controller', which guides the reasoning process.

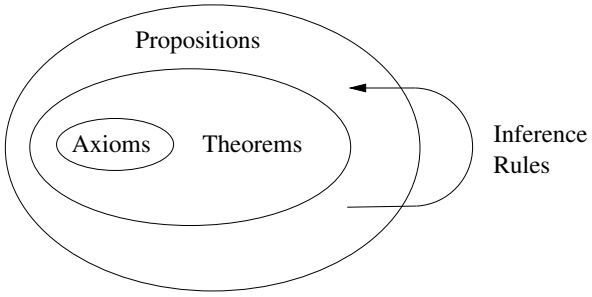


Figure 1: Classical formal logical system.

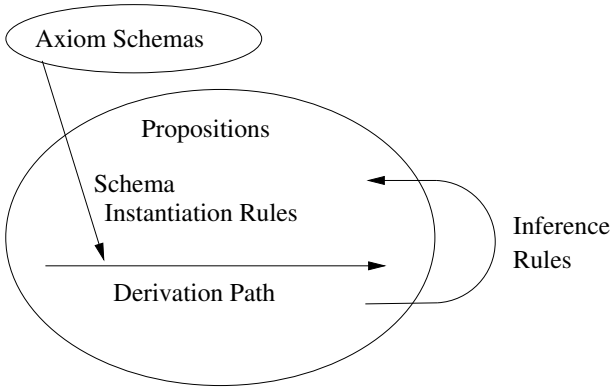


Figure 2: Dynamic reasoning system.

For contrast, and by way of introductory overview, the basic structure of a classical formal logical system is portrayed in Figure 1 and that of a DRS in Figure 2. A classical system is defined by providing a language consisting of a set of propositions, selecting certain propositions to serve as axioms, and specifying a set of inference rules saying how, from certain premises one can derive certain conclusions. The theorems then amount to all the propositions that can be derived from the axioms by means of the rules. Such systems are monotonic in that adding new axioms always serves to increase the set of theorems. Axioms are of two kinds: logical and extralogical (or 'proper', or 'nonlogical'). The logical axioms together with the inference rules comprise the 'logic'. The extralogical axioms comprise information about the application domain. A DRS begins similarly with specifying a language consisting of a set of propositions. But here the 'logic' is given in terms of a set of axioms schemas, some inference rules as above, and some rules for instantiating the schemas. The indicated derivation path serves as the belief set. Logical axioms may be entered into the derivation path by applying instantiation rules. Extralogical axioms are entered from an external source (human user, another agent, a mechanical sensor, etc.). Thus the derivation path evolves over time, with propositions being entered into the path either as extralogical axioms or derived by means of inference rules in accordance with the algorithms provided in the controller. Whenever a new proposition is entered into the path it is marked as 'believed'. In the event that a contradiction arises in the derivation path, a nonmonotonic belief revision process is invoked which leads to certain previously believed propositions becoming disbelieved, thereby removing the contradiction. A brief overview of these two components of a DRS is given in Sections 2.1 and 2.2.

2.1. Path Logic

A *path logic* consists of a language, axiom schemas, inference rules, and a derivation path, as follows.

Language: Here denoted $\mathcal{L}$, this consists of all *expressions* (or *formulas*) that can be generated from a given set σ of *symbols* in accordance with a collection of production rules (or an inductive definition, or some similar manner of

definition). As symbols typically are of different types (e.g., individual variables, constants, predicate symbols, etc.) it is assumed that there is an unlimited supply (uncountably many if necessary) of each type. Moreover, as is customary, some symbols will be *logical symbols* (e.g., logical connectives, quantifiers, and individual variables), and some will be *extralogical symbols* (e.g., individual constants and predicate symbols). It is assumed that $\mathcal{L}$ contains at least the logical connectives for expressing *negation* and *conjunction*, herein denoted $\neg$ and $\wedge$, or a means for defining these connectives in terms of the given connectives. For example, in the following we take $\neg$ and $\rightarrow$ as given and use the standard definition of $\wedge$ in terms of these.

Axiom Schemas: Expressed in some meta notation, these describe the expressions of $\mathcal{L}$ that are to serve as *logical axioms*.

Inference Rules: These must include one or more rules that enable instantiation of the axiom schemas. All other inference rules will be of the usual kind, i.e., stating that, from expressions having certain forms (premise expressions), one may infer an expression of some other form (a conclusion expression). Of the latter, two kinds are allowed: *logical rules*, which are considered to be part of the underlying logic, and *extralogical rules*, which are associated with the intended application. Note that logical axioms are expressions that are derived by applying the axiom schema instantiation rules. Inference rules may be viewed formally as mappings from $\mathcal{L}$ into itself.

The rule set may include derived rules that simplify deductions by encapsulating frequently used argument patterns. Rules derived using only logical axioms and logical rules will also be *logical rules*, and derived rules whose derivations employ extralogical rules will be additional *extralogical rules*.

Derivation Paths: These consist of a sequences of pairs $(L_0, B_0), (L_1, B_1), \ldots$, where L_t is the sublanguage of $\mathcal{L}$ that is in use at time t, and B_t is the *belief set* in effect at time t. Such a sequence is *generated* as follows. Since languages are determined by the symbols they employ, it is useful to speak more directly in terms of the set σ_t comprising the symbols that are in use at time t and then let L_t be the sublanguage of $\mathcal{L}$ that is based on the symbols in σ_t. With this in mind, let σ_0 be the logical symbols of $\mathcal{L}$, so that L_0 is the minimal language employing only logical symbols, and let $B_0 = \emptyset$. Then, given (L_t, B_t), the pair (L_{t+1}, B_{t+1}) is formed in one of the following ways:

1. $\sigma_{t+1} = \sigma_t$ (so that $L_{t+1} = L_t$) and B_{t+1} is obtained from B_t by adding an expression that is derived by application of an inference rule that instantiates an axiom schema,
2. $\sigma_{t+1} = \sigma_t$ and B_{t+1} is obtained from B_t by adding an expression that is derived from expressions appearing earlier in the path by application of an inference rule of the kind that infers a conclusion from some premises,
3. $\sigma_{t+1} = \sigma_t$ and an expression employing these symbols is added to B_t to form B_{t+1},
4. some new extralogical symbols are added to σ_t to form σ_{t+1}, and an expression employing the new symbols is added to B_t to form B_{t+1},
5. $\sigma_{t+1} = \sigma_t$ and B_{t+1} is obtained from B_t by applying a belief revision algorithm as described in the following.

Expressions entered into the belief set in accordance with either (3) or (4) will be *extralogical axioms*. A DRS can generate any number of different derivation paths, depending on the extralogical axioms that are input and the inference rules that are applied.

Whenever an expression is entered into the belief set it is assigned a *label* comprised of:

1. A *time stamp*, this being the value of the subscript $t+1$ on the set B_{t+1} formed by entering the expression into the belief set in accordance with any of the above items (1) through (4). The time stamp serves as an *index* indicating the expression's position in the belief set.
2. A *from-list*, indicating how the expression came to be entered into the belief set. In case the expression is entered in accordance with the above item (1), i.e., using a schema instantiation rule, this list consists of the name (or other identifier) of the schema and the name (or other identifier) of the inference rule if the system has more than one such rule. In case the expression is entered in accordance with above item (2), the list consists of the indexes (time stamps) of the premise expressions and the name (or other identifier) of the inference rule. In case the expression is entered in accordance with either of items (3) or (4), i.e., is a extralogical axiom, the list will consist of some code indicating this (e.g., *es* standing for 'external source') possibly together with some identifier or other information regarding the source.
3. A *to-list*, being a list of indexes of all expressions that have been entered into the belief set as a result of rule applications involving the given expression as a premise. Thus to-lists may be updated at any future time.
4. A *status indicator* having the value *bel* or *disbel* according as the proposition asserted by the expression currently is believed or disbelieved. The primary significance of this status is that only expressions that are believed can serve as premises in inference rule applications. Whenever an expression is first entered into the belief set, it is assigned status *bel*. This value may then be changed during belief revision at a later time. When an expression's status is changed from *bel* to *disbel* it is said to have been *retracted*.
5. An *epistemic entrenchment factor*, this being a numerical value indicating the strength with which the proposition asserted by the expression is held. This terminology is adopted in recognition of the work by Gärdenfors, who initiated this concept (Gardenfors 1988; 1992), and is used here for essentially the same purpose, namely, to assist when making decisions regarding belief retractions. Depending on the application, however, this value might alternatively be interpreted as a degree of belief, as a certainty factor, as a degree of importance, or some other type of value to be used for this purpose. Logical axioms always receive the highest possible epistemic entrenchment value, whatever scale or range may be employed.

6. A *knowledge category specification*, having one of the values *a priori*, *a posteriori*, *analytic*, and *synthetic*. These terms are employed in recognition of the philosophical tradition initiated by Immanuel Kant (Kant 1935). Logical axioms are designated as a priori; extralogical axioms are designated as a posteriori; expressions whose derivations employ only logical axioms and logical inference rules are designated as analytic; and expressions whose derivations employ any extralogical axioms or extralogical rules are designated as synthetic.

Thus when an expression P is entered into the belief set, it is more exactly entered as an expression-label pair (P, λ), where λ is the label. A DRS's language, axiom schemas, and inference rules comprise a *logic* in the usual sense. It is required that this logic be *consistent*, i.e., for no expression P is it possible to derive both P and $\neg P$. The belief set may become inconsistent, nonetheless, through the introduction of contradictory extralogical axioms.

In what follows, only expressions representing a posteriori and synthetic knowledge may be retracted; expressions of a priori knowledge are taken as being held unequivocally. Thus the term 'a priori knowledge' is taken as synonymous with 'belief held unequivocally', and 'a posteriori knowledge' is interpreted as 'belief possibly held only tentatively' (some a posteriori beliefs may be held unequivocally). Accordingly the distinction between knowledge and belief is somewhat blurred, and what is referred to as a 'belief set' might alternatively be called a 'knowledge base', as is often the practice in AI systems.

2.2. Controller

A *controller* effectively determines the modeled agent's *purpose* or *goals* by managing the DRS's interaction with its environment and guiding the reasoning process. With regard to the latter, the objectives typically include (i) deriving all expressions salient to the given application and entering these into the belief set, and (ii) ensuring that the belief set remains consistent. To these ends, the business of the controller amounts to performing the following operations.

1. Receiving input from its environment, e.g., human users, sensors, or other artificial agents, expressing this input as expressions in the given language $\mathcal{L}$, and entering these expressions into the belief set in the manner described above (derivation path items (3) and (4)). During this operation, new symbols are appropriated as needed to express concepts not already represented in the current L_t.
2. Applying inference rules in accordance with some extralogical objective (some plan, purpose, or goal) and entering the derived conclusions into the belief set in the manner described above (derivation path items (1) and (2)).
3. Performing any actions that may be prescribed as a result of the above reasoning process, e.g., moving a robotic arm, returning a response to a human user, or sending a message to another artificial agent.
4. Whenever necessary, applying a 'dialectical belief revision' algorithm for contradiction resolution in the manner described below.

A *contradiction* is an expression of the form $P \wedge \neg P$. Sometimes it is convenient to represent the general notion of contradiction by the falsum symbol, $\perp$. Contradiction resolution is triggered whenever a contradiction or a designated equivalent expression is entered into the belief set. We may assume that this only occurs as the result of an inference rule application, since it obviously would make no sense to enter a contradiction directly as an extralogical axiom. The contradiction resolution algorithm entails three steps:

1. Starting with the from-list in the label on the contradictory expression, backtrack through the belief set following from-lists until one identifies all extralogical axioms that were involved in the contradiction's derivation. Note that such extralogical axioms must exist, since, by the consistency of the logic, the contradiction cannot constitute analytical knowledge, and hence must be synthetic.
2. Change the belief status of one or more of these extralogical axioms, as many as necessary to invalidate the derivation of the given contradiction. The decision as to which axioms to retract may be dictated, or at least guided by, the epistemic entrenchment values. In effect, those expressions with the lower values would be preferred for retraction. In some systems, this retraction process may be automated, and in others it may be human assisted.
3. Forward chain through to-lists starting with the extralogical axiom(s) just retracted, and retract all expressions whose derivations were dependent on those axioms. These retracted expressions should include the contradiction that triggered this round of belief revision (otherwise the correct extralogical axioms were not retracted).

This belief revision algorithm is reminiscent of G. W. F. Hegel's 'dialectic', described as a process of 'negation of the negation' (Hegel 1931). In that treatment, the latter (first occurring) negation is a perceived internal conflict (here a contradiction), and the former (second occurring) one is an act of transcendence aimed at resolving the conflict (here removing the contradiction). In recognition of Hegel, the belief revision/retraction process formalized in the above algorithm will be called *Dialectical Belief Revision*.

3. First-Order Logic

The paper (Schwartz 2013) defines a notion of first-order *theory* suitable for use in a DRS, provides this with a well-defined semantics (a notion of *model*), and establishes a Soundness Theorem: a theory is consistent if it has a model. The notions of theory and semantics are designed to accommodate the notion of a belief set evolving over time, as well as inference rules that act by instantiating axiom schemas. A first-order language $\mathcal{L}$ is defined following the notations of (Hamilton 1988). This includes notations $\mathbf{A}_n^m$ as predicate symbols (here the n-th m-ary predicate symbol) and $\mathbf{a}_n$ for individual variables. Then, in the path logic, the languages at each successive time step are sublanguages of $\mathcal{L}$. The semantics follows the style of (Shoenfield 1967). The axiom schemas of (Hamilton 1988) are adopted. The inference rules are those of (Hamilton 1988) together with some

rules for axiom schema instantiation. The formalism is sufficiently different from the classical version that new proofs of all relevant propositions must be restated in this context and proven correct. The treatment also establishes the validity of several derived inference rules that become useful in later examples, including:

Hypothetical Syllogism From $P \rightarrow Q$ and $Q \rightarrow R$ infer $P \rightarrow R$, where P, Q, R are any formulas.

Aristotelian Syllogism From $(\forall x)(P \rightarrow Q)$ and $P(a/x)$, infer $Q(a/x)$, where P, Q are any formulas, x is any individual variable, and a is any individual constant.

Subsumption From $(\forall x)(\alpha(x) \rightarrow \beta(x))$ and $(\forall x)(\beta(x) \rightarrow \gamma(x))$, infer $(\forall x)(\alpha(x) \rightarrow \gamma(x))$, where α, β, γ are any unary predicate symbols, and x is any individual variable.

Contradiction Detection From P and $\neg P$ infer $\perp$, where P is any formula.

Conflict Detection From $(\forall x)\neg(P \wedge Q)$, $P(a/x)$, and $Q(a/x)$ infer $\perp$, where P, Q are any formulas, x is any individual variable, and a is any individual constant.

4. Example: Multiple Inheritance with Exceptions

The main objective of (Schwartz 1997) was to show how a DRS framework could be used to formulate reasoning about property inheritance with exceptions, where the underlying logic was a probabilistic 'logic of qualified syllogisms'. This work was inspired in part by the frame-based systems due to (Minsky 1975) and constitutes an alternative formulation of the underlying logic (e.g., as discussed by (Hayes 1980)).

What was missing in (Schwartz 1997) was the notion of a controller. There a reasoning system was presented and shown to provide intuitively plausible solutions to numerous 'puzzles' that had previously appeared in the literature on nonmonotonic reasoning, e.g., Opus the Penguin (Touretsky 1984), Nixon Diamond (Touretsky et al. 1987), and Clyde the Elephant (Touretsky et al. 1987). But there was nothing to guide the reasoning processes—no means for providing a sense of purpose for the reasoning agent. The present work fills this gap by adding a controller. Moreover, it deals with a simpler system based on first-order logic and remands further exploitation of the logic of qualified syllogisms to a later work. The kind of DRS developed in this section will be termed a *multiple inheritance system* (MIS).

For this application the language $\mathcal{L}$ discussed in Section 3 is expanded by including some *typed predicate symbols*, namely, some unary predicate symbols $\mathbf{A}_1^{(k)}, \mathbf{A}_2^{(k)}, \ldots$ representing *kinds* of things (any objects), and some unary predicate symbols $\mathbf{A}_1^{(p)}, \mathbf{A}_2^{(p)}, \ldots$ representing *properties* of things. The superscripts k and p are applied also to generic denotations. Thus an expression of the form $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(p)}(x))$ represents the proposition that all αs have property β. These new predicate symbols are used here purely as syntactical items for purposes of defining an extralogical 'specificity principle' and some associated extralogical graphical structures and algorithms. Semantically they are treated exactly the same as other predicate symbols.

A *multiple-inheritance hierarchy* H will be a directed graph consisting of a set of *nodes* together with a set of *links* represented as ordered pairs of nodes. Nodes may be either *object* nodes, *kind* nodes, or *property* nodes. A link of the form (object node, kind node) will be an *object-kind* link, one of the form (kind node, kind node) will be a *subkind-kind* link, and one of the form (kind node, property node) will be a *has-property* link. There will be no other types of links. Object nodes will be labeled with (represent) individual constant symbols, kind nodes will be labeled with (represent) kind-type unary predicate symbols, and property nodes will be labeled with (represent) property-type unary predicate symbols or negations of such symbols. In addition, each property type predicate symbol with bear a numerical subscript, called an *occurrence index*, indicating an occurrence of that symbol in a given hierarchy H. These indexes are used to distinguish different occurrences of the same property-type symbol in H. An object-kind link between an individual constant symbol a and a predicate symbol $\alpha^{(k)}$ will represent the formula $\alpha^{(k)}(a)$, a subkind-kind link between a predicate symbol $\alpha^{(k)}$ and a predicate symbol $\beta^{(k)}$ will represent the formula $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(k)}(x))$, and a has-property link between a predicate symbol $\alpha^{(k)}$ and a predicate symbol $\beta_1^{(p)}$ will represent the formula $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta_1^{(p)}(x))$.

Given such an H, there is defined on the object nodes and the kind nodes a *specificity relation* $>_s$ (read 'more specific than') according to: (i) if $(\mathrm{node}_1, \mathrm{node}_2)$ is either an object-kind link or a kind-kind link, then $\mathrm{node}_1 >_s \mathrm{node}_2$, and (ii) if $\mathrm{node}_1 >_s \mathrm{node}_2$ and $\mathrm{node}_2 >_s \mathrm{node}_3$, then $\mathrm{node}_1 >_s \mathrm{node}_3$. We shall also have a dual *generality relation* $>_g$ (read 'more general than') defined by $\mathrm{node}_1 >_g \mathrm{node}_2$ iff $\mathrm{node}_2 >_s \mathrm{node}_1$. It follows that object nodes are maximally specific and minimally general. It also follows that H may have any number of maximally general nodes, and in fact that it need not be connected. A maximally general node is a *root* node. A *path* in a hierarchy H (not to be confused with the path in a path logic) will be a sequence $\mathrm{node}_1, \ldots, \mathrm{node}_n$ wherein, node_1 is a root node and, for each $i = 1, \ldots, n-2$, the pair $(\mathrm{node}_{i+1}, \mathrm{node}_i)$ is a subkind-kind link, and, the pair $(\mathrm{node}_n, \mathrm{node}_{n-1})$ is either a subkind-kind link or an object-kind link. Note that property nodes do not participate in paths as here defined.

It is desired to organize a multiple inheritance hierarchy as a directed acyclic graph (DAG) without redundant links with respect to the object-kind and subkind-kind links (i.e., here ignoring has-property links), where, as before, by a redundant link is meant a direct link from some node to an ancestor of that node other than the node's immediate ancestors (i.e., other than its parents). More exactly, two distinct paths will form a *redundant pair* if they have some node in common beyond the first place where they differ. This means that they comprise two distinct paths to the common node(s). A path will be simply *redundant* (or *redundant in* H) if it is a member of a redundant pair. A path contains a *loop* if it has more than one occurrence of the same node. Provisions are

made in the following algorithms to ensure that hierarchies with loops or redundant paths are not allowed. As is customary, the hierarchies will be drawn with the upward direction being from more specific to less (less general to more), so that roots appear at the top and objects appear at the bottom. Kind-property links will extend horizontally from their associated kind nodes.

In terms of the above specificity relation on H, we can assign an *address* to each object and kind node in the following manner. Let the addresses of the root nodes, in any order, be $(1), (2), (3), \ldots$. Then for the node with address (1), say, let the next most specific nodes in any order have the addresses $(1,1), (1,2), (1,3), \ldots$; let the nodes next most specific to the one with address $(1,1)$ have addresses $(1,1,1), (1,1,2), (1,1,3), \ldots$; and so on. Thus an address indicates the node's position in the hierarchy relative to some root node. Inasmuch as an object or kind node may be more specific than several different root nodes, the same node may have more than one such address. Note that the successive initial segments of an address are the addresses of the nodes appearing in the path from the related root node to the node having that initial segment as its address. Let $>$ denote the usual lexicographic order on addresses. We shall apply $>$ also to the nodes having those addresses. It is easily verified that, if $\mathrm{node}_1 > \mathrm{node}_2$ and the node_2 address is an initial segment of the node_1 address, then $\mathrm{node}_1 >_s \mathrm{node}_2$, and conversely. For object and kind nodes, we shall use the term *specificity rank* (or just *rank*) synonymously with 'address'.

Since, as mentioned, it is possible for any given object or kind node to have more than one address, it thus can have more than one rank. Two nodes are comparable with respect to the specificity relation $>_s$, however, only if they appear on the same path, i.e., only if one node is an ancestor of the other, in which case only the rank each has acquired due to its being on that path will apply. Thus, if two nodes are comparable with respect to their ranks by the relation $>_s$, there is no ambiguity regarding the ranks being compared.

Having thus defined specificity ranks for object and kind nodes, let us agree that each property node inherits the rank of the kind node to which it is linked. Thus for property nodes the rank is not an address.

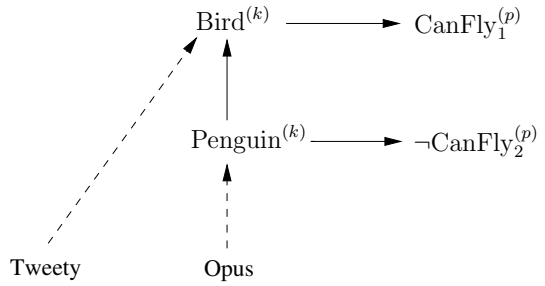


Figure 3: Tweety the Bird and Opus the Penguin as an MIS.

An example of such a hierarchy is shown in Figure 3. Here 'Tweety' and 'Opus' may be taken as names for the individual constants $\mathbf{a}_1$ and $\mathbf{a}_2$, and '$\mathrm{Bird}^{(k)}$', '$\mathrm{Penguin}^{(k)}$', and '$\mathrm{CanFly}^{(p)}$' can be taken as names, respectively, for the unary predicate symbols $\mathbf{A}_1^{(k)}$, $\mathbf{A}_2^{(k)}$, and $\mathbf{A}_1^{(p)}$. [Note: The superscripts are retained on the names only to visually identify the types of the predicate symbols, and could be dropped without altering the meanings.] The links represent the formulas

$$(\forall x)(\mathrm{Penguin}^{(k)}(x) \rightarrow \mathrm{Bird}^{(k)}(x))$$
$$(\forall x)(\mathrm{Bird}^{(k)}(x) \rightarrow \mathrm{CanFly}_1^{(p)}(x))$$
$$(\forall x)(\mathrm{Penguin}^{(k)}(x) \rightarrow \neg\mathrm{CanFly}_2^{(p)}(x))$$
$$\mathrm{Bird}^{(k)}(\mathrm{Tweety})$$
$$\mathrm{Penquin}^{(k)}(\mathrm{Opus})$$

The subscripts 1 and 2 on the predicate symbol $\mathrm{CanFly}^{(p)}$ in the graph distinguish the different occurrences of this symbol in the graph, and the same subcripts on the symbol occurrences in the formulas serve to correlate these with their occurrences in the graph. Note that these are just separate occurrences of the same symbol, however, and therefore have identical semantic interpretations. Formally, $\mathrm{CanFly}_1^{(p)}$ and $\mathrm{CanFly}_2^{(p)}$ can be taken as standing for $\mathbf{A}_{1_1}^{(p)}$ and $\mathbf{A}_{1_2}^{(p)}$ with the lower subscripts being regarded as extralogical notations indicating different occurrences of $\mathbf{A}_1^{(p)}$.

This figure reveals the rationale for the present notion of multiple-inheritance hierarchy. The intended interpretation of the graph is that element nodes and kind nodes inherit the properties of their parents, with the exception that more specific property nodes take priority and block inheritances from those that are less specific. Let us refer to this as the *specificity principle*. In accordance with this principle, in Figure 3 Tweety inherits the property CanFly from Bird, but Opus does not inherit this property because the inheritance is blocked by the more specific information that Opus is a Penguin and Penguins cannot fly.

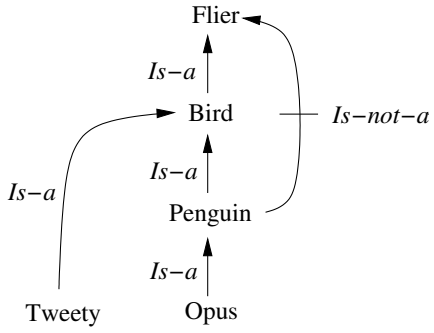


Figure 4: Tweety the Bird and Opus the Penguin, original version.

Figure 3 constitutes a rethinking of the well-known example of Opus the penguin depicted in Figure 4 (adapted from (Touretsky1984)). The latter is problematic in that, by one reasoning path one can conclude that Opus is a flier, and by another reasoning path that he is not. This same contradiction is implicit in the formulas introduced above,

since if one were to apply the axioms and rules of first-order logic discussed in Section 3, one could derive both $\mathrm{CanFly}^{(p)}(\mathrm{Opus})$ and $\neg\mathrm{CanFly}^{(p)}(\mathrm{Opus})$, in which case the system would be inconsistent.

Formal Specification of an Arbitrary MIS

We are now in a position to define the desired kind of DRS. For the path logic, let the language be the one described above, obtained from the $\mathcal{L}$ of Section 3 by adjoining the additional unary kind-type and property-type predicate symbols, let the axiom schemas and inference rules be those discussed in Section 3 together with Aristotelian Syllogism and Contradiction Detection. In this case, derivation paths will consist of triples (L_t, B_t, H_t), where these components respectively are the (sub)language (of $\mathcal{L}$), belief set, and multiple inheritance hierarchy at time t. In accordance with Section 2, let L_0 be the minimal sublanguage of $\mathcal{L}$ consisting of all formulas that can be built up from the atomic formula $\bot$, and let $B_0 = \emptyset$. In addition, let $H_0 = \emptyset$.

The MIS controller is designed to enforce the above specificity principle. Contradictions can arise in an MIS that has inherently contradictory root nodes in its multiple inheritance hierarchy. An example of this, the famous Nixon Diamond (Touretsky 1987), will be discussed. The purpose of the MIS controller will be (i) to derive and enter into the belief set all object classifications implicit in the multiple inheritance hierarchy, i.e., all formulas of the form $\alpha^{(k)}(a)$ that can be derived from formulas describing the hierarchy (while observing the specificity principle), and (ii) to ensure that the belief set remains consistent. Item (i) thus defines what will be considered the *salient information* for an MIS. Also, the MIS controller is intended to maintain the multiple inheritance hierarchy as a DAG without redundant paths with respect to just the object and kind nodes. Formulas that can be input by the users may have one of the forms (i) $\alpha^{(k)}(a)$, (ii) $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(k)}(x))$, (iii) $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(p)}(x))$, and (iv) $(\forall x)(\alpha^{(k)}(x) \rightarrow \neg\beta^{(p)}(x))$. It will be agreed that the epistemic entrenchment value for all input formulas is 0.5.

We may now define some algorithms that are to be executed in response to each type of user input. There will be eight types of events. Event Types 1, 6, 7 and 8 correspond to user inputs, and the others occur as the result of rule applications. In all such events it is assumed that, if the formula provided to the controller already exists and is active in the current belief set, its input is immediately rejected. In each event, assume that the most recent entry into the derivation path is (L_t, B_t, H_t). For the details of the algorithms, please see (Schwartz 2013).

Event Type 1: A formula of the form $\alpha^{(k)}(a)$ is provided to the controller by a human user.

Event Type 2: A formula of the form $\alpha^{(k)}(a)$ is provided to the controller as a result of an inference rule application (Aristotelian Syllogism).

Event Type 3: A formula of the form $\alpha^{(p)}(a)$ is provided to the controller as a result of an inference rule application (Aristotelian Syllogism).

Event Type 4: A formula of the form $\neg\alpha^{(p)}(a)$ is provided to the controller as a result of an inference rule application (Aristotelian Syllogism).

Event Type 5: The formula $\bot$ is provided to the controller as the result of an application of Contradiction Detection.

Event Type 6: A formula of the form $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(k)}(x))$ is provided to the controller by a human user.

Event Type 7: A formula of the form $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(p)}(x))$ is provided to the controller by a human user.

Event Type 8: A formula of the form $(\forall x)(\alpha^{(k)}(x) \rightarrow \neg\beta^{(p)}(x))$ is provided to the controller by a human user.

Main Results

That an MIS controller produces all relevant salient information as prescribed above can be summarized as a pair of theorems.

Theorem 5.1. The foregoing algorithms serve to maintain the hierarchy with respect to the object and kind nodes as a directed acyclic graph without redundant links.

Theorem 5.2. After any process initiated by a user input terminates, the resulting belief set will contain a formula of the form $\alpha^{(k)}(a)$ or $\alpha^{(p)}(a)$ or $\neg\alpha^{(p)}(a)$ iff the formula is derivable from the formulas corresponding to links in the inheritance hierarchy, observing the specificity principle.

That the algorithms serve to preserve the consistency of the belief set is established as:

Theorem 5.3. For any derivation path in an MIS, the belief set that results at the conclusion of a process initiated by a user input will be consistent with respect to the formulas of the forms $\alpha^{(k)}(a)$, $(\forall x)(\alpha^{(k)}(x) \rightarrow \beta^{(p)}(x))$, and $\alpha^{(p)}(a)$.

Illustration 1

Some of the algorithms associated with the foregoing events can be illustrated by considering the inputs needed to create the inheritance hierarchy shown in Figure 3. This focuses on the process of property inheritance with exceptions. Let us abbreviate 'Bird', 'Penguin', and 'CanFly', respectively, by 'B', 'P', and 'CF'. In accordance with the definition of derivation path in Section 2.1, the language L_0 will consist only of the formula $\bot$, and the belief set $B_0 = \emptyset$. In accordance with the definition of an MIS, $H_0 = \emptyset$. We consider inputs of the afoermentioned formulas, with each input comprising a type of event initiating a particular reasoning algorithm. These inputs and event types are:

$(\forall x)(\mathrm{P}^{(k)}(x) \rightarrow \mathrm{B}^{(k)}(x))$, Type 6
$(\forall x)(\mathrm{B}^{(k)}(x) \rightarrow \mathrm{CF}_1^{(p)}(x))$, Type 7
$(\forall x)(\mathrm{P}^{(k)}(x) \rightarrow \neg\mathrm{CF}_2^{(p)}(x))$, Type 8
$\mathrm{B}^{(k)}(\mathrm{Tweety})$, Type 1
$\mathrm{P}^{(k)}(\mathrm{Opus})$, Type 1

The specificity principle is invoked during the last event. This results in the following belief set (omitting formula labels):

$(\forall x)(\mathrm{P}^{(k)}(x) \rightarrow \mathrm{B}^{(k)}(x))$
$(\forall x)(\mathrm{B}^{(k)}(x) \rightarrow \mathrm{CF}_1^{(p)}(x))$
$(\forall x)(\mathrm{P}^{(k)}(x) \rightarrow \neg\mathrm{CF}_2^{(p)}(x))$
$\mathrm{B}^{(k)}(\mathrm{Tweety})$
$\mathrm{CF}_1^{(p)}(\mathrm{Tweety})$
$\mathrm{P}^{(k)}(\mathrm{Opus})$
$\mathrm{B}^{(k)}(\mathrm{Opus})$
$\neg\mathrm{CF}_2^{(p)}(\mathrm{Opus})$

Thus is is seen that, in this example, the algorithms serve to derive all salient information, i.e., all formulas of the forms $\alpha^{(k)}(a)$, $\alpha^{(p)}(a)$, and $\alpha^{(p)}(a)$ that are implicit in the graph, while at the same time correctly enforcing the specificity principle. It may also be observed that the belief set is consistent.

Illustration 2

This considers an application of Contradiction Detection. The classic Nixon Diamond puzzle (cf. Touretsky et al. 1987) is shown in Figure 5. Here a contradiction arises because, by the reasoning portrayed on the left side, Nixon is a pacifist, whereas, by the reasoning portrayed on the right, he is not. The resolution of this puzzle in the context of an MIS can be described in terms of the multiple inheritance hierarchy shown in Figure 6.

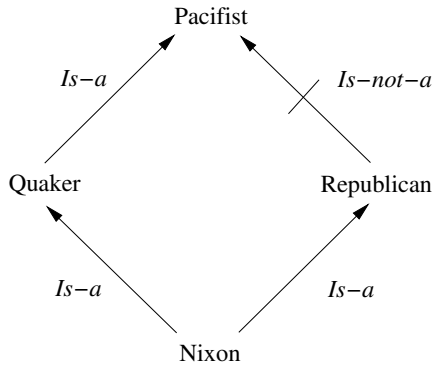


Figure 5: Nixon Diamond, original version.

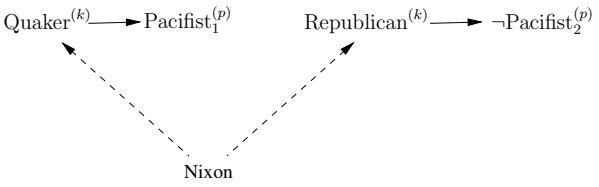


Figure 6: Nixon Diamond as an MIS.

The links in Figure 6 represent the formulas

$(\forall x)(\mathrm{Quaker}^{(k)}(x) \rightarrow \mathrm{Pacifist}_1^{(p)}(x))$
$(\forall x)(\mathrm{Republican}^{(k)}(x) \rightarrow \neg\mathrm{Pacifist}_2^{(p)}(x))$
$\mathrm{Quaker}^{(k)}(\mathrm{Nixon})$
$\mathrm{Republican}^{(k)}(\mathrm{Nixon})$

The action of the algorithms may be traced similarly as in Illustration 1. Let 'Quaker', 'Republican' and 'Pacifist' denote the predicate symbols $\mathbf{A}_1^{(k)}$, $\mathbf{A}_2^{(k)}$ and $\mathbf{A}_1^{(p)}$, and abbreviate these by 'Q', 'R' and 'P'. Let 'Nixon' denote the individual constant $\mathbf{a}_1$. L_0, B_0, and H_0 will be as before. The inputs and their event types are:

$(\forall x)(\mathrm{Q}^{(k)}(x) \rightarrow \mathrm{P}_1^{(p)}(x))$, Type 7.
$(\forall x)(\mathrm{R}^{(k)}(x) \rightarrow \neg\mathrm{P}_1^{(p)}(x))$, Type 8.
$\mathrm{Q}^{(k)}(\mathrm{Nixon})$, Type 1.
$\mathrm{R}^{(k)}(\mathrm{Nixon})$, Type 1.

These lead to the following belief set (again omitting formual labels):

$(\forall x)(\mathrm{Q}^{(k)}(x) \rightarrow \mathrm{P}_1^{(p)}(x))$
$(\forall x)(\mathrm{R}^{(k)}(x) \rightarrow \neg\mathrm{P}_2^{(p)}(x))$
$\mathrm{Q}^{(k)}(\mathrm{Nixon})$.
$\mathrm{P}_1^{(p)}(\mathrm{Nixon})$
$\mathrm{R}^{(k)}(\mathrm{Nixon}$
$\neg\mathrm{P}_2^{(p)}(\mathrm{Nixon})$
$\perp$

At this point Dialectical Belief Revision is invoked. All the formulas that were input by the user are candidates for belief change. Suppose that the formula $(\forall x)(\mathrm{R}^{(k)}(x) \rightarrow \neg\mathrm{P}_2^{(p)}(x))$, is chosen. Then the procedure forward chains through to lists, starting with this formula, and changes to *disbel* the status first of $\neg\mathrm{P}_2^{(p)}(\mathrm{Nixon})$, and then of $\perp$. This results in a belief set with these three formulas removed (disbelieved) leaving only the left side of the hierarchy in Figure 6. Thus again all salient information is derived and the resulting belief set is consistent.

Further well-known puzzles that can be resolved similarly within an MIS are the others discussed in (Schwartz 1997), namely, Bosco the Blue Whale (Stein 1992), Suzie the Platypus (Stein 1992), Clyde the Royal Elephant (Touretsky et al. 1987), and Expanded Nixon Diamond (Touretsky et al. 1987).

References

Alchourón, C. E.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic* 50(2):510–530.

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

Delgrande, J. P., and Farber, W., eds. 2011. *Logic Programming and Nonmonotonic Reasoning 11th International Conference, LPNMR 2011*. Lecture notes in Computer Science, Volume 66452011, Springer Verlag.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence* 12:231–272.

Elgot-Drapkin, J. J. 1988. *Step Logic: Reasoning Situated in Time*. PhD thesis, University of Maryland, College Park. Technical Report CS-TR-2156 and UMIACS-TR-88-94.

Elgot-Drapkin, J. J.; Miller, M.; and Perlis, D. 1987. Life on a desert island: ongoing work on real-time reasoning. In F.M. Brown, ed., *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pp. 349–357, Los Altos, CA: Morgan Kaufmann.

Elgot-Drapkin, J. J.; Miller, M.; and Perlis, D. 1991. Memory, reason, and time: the step-logic approach. In R. Cummins and J. Pollock, eds, *Philosophy and AI: Essays at the Interface*, pp. 79–103. MIT Press.

Elgot-Drapkin, J. J., and Perlis, D. 1990. Reasoning situated in time I: basic concept. *Journal of Experimental and Theoretical Artificial Intelligence* 2(1):75–98.

Gelfond, M. and Kahl, Y., *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Anwer Set Programming Approach*, Cambridge University Press, 2014.

Hayes, P. J. 1980. The logic of frames. In D. Metzing, ed., *Frame Conceptions and Text Understanding*, Berlin: Walter de Gruyter, pp. 46–61.

Fermé, E., and Hansson, S. O. 2011. AGM 25 years: twenty-five years of research in belief change. *J. Philos Logic*, 40:295–331.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. Cambridge, MA: MIT Press/Bradford Books.

Gärdenfors, P., ed. 1992. *Belief Revision*. Cambridge University Press.

Ginsberg, M. L., ed. 1987. *Readings in Nonmonotonic Reasoning*. Los Altos, CA: Morgan Kaufmann.

Hamilton, A. G. 1988. *Logic for Mathematicians, Revised Edition*, Cambridge University Press.

Hansson, S.O. 1999. *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Dordercht, Kluwer Academic Publishers.

Hegel, G.W.F. 1931. *Phenomenology of Mind*. J.B. Baillie, trans, 2nd edition. Oxford: Clarendon Press.

Kant, I. 1935 *Critique of Pure Reason*. N.K. Smith, trans. London, England: Macmillan.

McCarthy, J. 1980. Circumscription—a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 171–172. Reprinted in (Ginsberg 1987), pp. 145–152.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. Stanford University. Reprinted in (Ginsberg 1987), pp. 26–45, and in V. Lifschitz, ed., *Formalizing Common Sense: Papers by John McCarthy*, Norwood, NJ: Ablex, 1990, pp. 21–63.

McDermott, D., and Doyle, J. 1980. Non-monotonic logic–I. *Artificial Intelligence* 13:41–72. Reprinted in (Ginsberg 1987), pp. 111–126.

Miller, M. J. 1993. *A View of One's Past and Other Aspects of Reasoned Change in Belief*. PhD thesis, University of Maryland, College Park, Department of Computer Science, July. Technical Report CS-TR-3107 and UMIACS-TR-93-66.

Minsky, M. 1975. A framework for representing knowledge. In P. Winston, ed., *The Psychology of Computer Vision*, New York: McGraw-Hill, pp. 211–277. A condensed version has appeared in D. Metzing, ed., *Frame Conceptions and Text Understanding*, Berlin: Walter de Gruyter, Berlin, 1980, pp. 1–25.

Perlis, D.; Elgot-Drapkin, J. J.; and Miller, M. 1991. Stop the world—I want to think. In K. Ford and F. Anger, eds., *International Journal of Intelligent Systems: Special Issue on Temporal Reasoning, Vol. 6*, pp. 443–456. Also Technical Report CS-TR-2415 and UMIACS-TR-90-26, Department of Computer Science, University of Maryland, College Park, 1990.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13(1-2):81–132. Reprinted in (Ginsberg 1987), pp. 68–93.

Schwartz, D. G. 1997. Dynamic reasoning with qualified syllogisms. *Artificial Intelligenc* 93:103–167.

Schwartz, D .G. 2013. Dynamic reasoning systems. *ACM Transactions on Computational Intelligence*, accepted subject to revision February 7, 2014.

Shoenfield, J. R. 1967. *Mathematical Logic*, Association for Symbolic Logic.

Shoham, Y. 1986. Chronological ignorance: time, nonmonotonicity, necessity, and causal theories. *Proceedings of the American Association for Artificial Intelligence, AAAI'86*, Philadelphia, PA, pp. 389–393.

Shoham, Y. 1988. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. Cambridge, MA: MIT Press.

Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence* 60:51–92.

Smith, B., and Kelleher, G., eds. 1988. *Reason Maintenance Systems and Their Applications*. Chichester, England:Ellis Horwood.

Stein, L. A. 1992. Resolving ambiguity in nonmonotonic inheritance hierarchies. *Artificial Intelligence* 55(2-3).

Touretzky, D. 1984. Implicit ordering of defaults in inheritance systems. *Proceedings of the Fifth National Conference on Artificial Intelligence, AAAI'84*, Austin, TX, Los Altos, CA: Morgan Kaufmann, pp. 322–325. Reprinted in (Ginsberg 1987), pp. 106–109, and in G. Shafer and J. Pearl, eds., *Readings in Uncertain Reasoning*, San Mateo, CA: Morgan Kaufmann, 1990, pp. 668–671.

Touretzky, D. S.; Horty, J .E.; and Thomason, R.H. 1987. A clash of intuitions: the current state of nonmonotonic multiple inheritance systems. *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'87*, Milan, Italy. pp. 476–482.

Probabilistic Inductive Logic Programming Based on Answer Set Programming*

Matthias Nickles[♯♭] **and Alessandra Mileo**[♯]
{matthias.nickles,alessandra.mileo}@deri.org
[♯] INSIGHT/DERI Galway
National University of Ireland, Galway
[♭] Department of Information Technology
National University of Ireland, Galway

Abstract

We propose a new formal language for the expressive representation of probabilistic knowledge based on Answer Set Programming (ASP). It allows for the annotation of first-order formulas as well as ASP rules and facts with probabilities and for learning of such weights from data (parameter estimation). Weighted formulas are given a semantics in terms of soft and hard constraints which determine a probability distribution over answer sets. In contrast to related approaches, we approach inference by optionally utilizing so-called streamlining XOR constraints, in order to reduce the number of computed answer sets. Our approach is prototypically implemented. Examples illustrate the introduced concepts and point at issues and topics for future research.

Keywords: *Uncertainty Reasoning, Answer Set Programming, Probabilistic Inductive Logic Programming, Statistical Relational Learning, SAT*

1 Introduction

Reasoning in the presence of uncertainty and relational structures (such as social networks and Linked Data) is an important aspect of knowledge discovery and representation for the Web, the Internet Of Things, and other potentially heterogeneous and complex domains. Probabilistic logic programing, and the ability to learn probabilistic logic programs from data, can provide an attractive approach to uncertainty reasoning and statistical relational learning, since it combines the deduction power and declarative nature of logic programming with probabilistic inference abilities traditionally known from less expressive graphical models such as Bayesian and Markov networks. A very successful type of logic programming for nonmonotonic domains is *Answer Set Programming* (ASP) (Lifschitz 2002; Gelfond and Lifschitz 1988). Since statistical-relational approaches to probabilistic reasoning often rely heavily on the propositionalization of first-order or other relational information, ASP appears to be an ideal basis for probabilistic logic programming, given its expressiveness and the existence of highly optimized grounders and solvers. However, despite the successful employment of conceptually related approaches in the area of SAT for probabilistic inference tasks, only a small number of approaches to probabilistic knowledge representation or probabilistic inductive logic programming under the stable model semantics exist so far, of which some are rather restrictive wrt. expressiveness and parameter estimation techniques. We build upon these and other existing approaches in the area of probabilistic (inductive) logic programming in order to provide a new ASP-based probabilistic logic programming language (with first-order as well as ASP basic syntax) for the representation of probabilistic knowledge. Weights which directly represent probabilities can be attached to arbitrary formulas, and we show how this can be used to perform probabilistic inference and how weights of hypotheses can be inductively learned from given relational examples. To the best of our knowledge, this is the first ASP-based approach to probabilistic (inductive) logic programming which does not impose restrictions on the annotation of ASP-rules and facts as well as FOL-style formulas with probabilities.

The remainder of this paper is organized as follows: the next section presents relevant related approaches. Section 3 introduces syntax and semantics of our new language. Section 4 presents our approach to probabilistic inference (including examples), and Section 5 shows how formula weights can be learned from data. Section 6 concludes.

2 Related Work

Being one of the early approaches to the logic-based representation of uncertainty sparked by Nilsson's seminal work (Nilsson 1986), (Halpern 1990) presents three different probabilistic first-order languages, and compares them with a related approach by Bacchus (Bacchus 1990). One language has a *domain-frequency* (or *statistical*) semantics, one has a possible worlds semantics (like our approach), and one bridges both types of semantics. While those languages as such are mainly of theoretical relevance, their types of semantics still form the backbone of most practically relevant contemporary approaches.

Many newer approaches, including Markov Logic Networks

*This work is an extended and revised version of A. Mileo, M. Nickles: Probabilistic Inductive Answer Set Programming by Model Sampling and Counting. First International Workshop on Learning and Nonmonotonic Reasoning (LNMR 2013), Corunna, Spain, 2013.

(see below), require a possibly expensive grounding (propositionalization) of first-order theories over finite domains. A recent approach which does not fall into this category but employs the principle of maximum entropy in favor of performing extensive groundings is (Thimm and Kern-Isberner 2012). However, since ASP is predestined for efficient grounding, we do not see grounding necessarily as a shortcoming. *Stochastic Logic Programs* (SLPs) (Muggleton 2000) are an influential approach where sets of rules in form of range-restricted clauses can be labeled with probabilities. Parameter learning for SLPs is approached in (Cussens 2000) using the EM-algorithm. Approaches which combine concepts from Bayesian network theory with relational modeling and learning are, e.g., (Friedman et al. 1999; Kersting and Raedt 2000; Laskey and Costa 2005). Probabilistic Relational Models (PRM) (Friedman et al. 1999) can be seen as relational counterparts to Bayesian networks In contrast to these, our approach does not directly relate to graphical models such as Bayesian or Markov Networks but works on arbitrary possible worlds which are generated by ASP solvers. ProbLog (Raedt, Kimmig, and Toivonen 2007) allows for probabilistic facts and definite clauses, and approaches to probabilistic rule and parameter learning (from interpretations) also exist for ProbLog. Inference is based on weighted model counting, which is similarly to our approach, but uses Boolean satisfiability instead of stable model search. ProbLog builds upon the very influential Distribution Semantics introduced for PRISM (Sato and Kameya 1997), which is also used by other approaches, such as Independent Choice Logic (ICL) (Poole 1997). Another important approach outside the area of ASP are *Markov Logic Networks* (MLN) (Richardson and Domingos 2006), which are related to ours. A MLN consists of first-order formulas annotated with weights (which are not probabilities). MLNs are used as "templates" from which Markov networks are constructed, i.e., graphical models for the joint distribution of a set of random variables. The (ground) Markov network generated from the MLN then determines a probability distribution over possible worlds. MLNs are syntactically similar to the logic programs in our framework (in our framework, weighted formulas can also be seen as soft or hard constraints for possible worlds), however, in contrast to MLN, we allow for probabilities as formula weights. Our initial approach to weight learning is closely related to certain approaches to MLN parameter learning (e.g., (Lowd and Domingos 2007)), as described in Section 5.

Located in the field of nonmonotonic logic programming, our approach is also influenced by P-log (Baral, Gelfond, and Rushton 2009) and abduction-based rule learning in probabilistic nonmonotonic domains (Corapi et al. 2011). With P-log, our approaches shares the view that answer sets can be seen as possible worlds in the sense of (Nilsson 1986). However, the syntax of P-log is quite different from our language, by restricting probabilistic annotations to certain syntactical forms and by the concept of independent experiments, which simplifies the implementation of their framework. In distinction from P-log, there is no particular coverage for causality modeling in our framework. (Corapi et al. 2011) allows to associate probabilities with abducibles and to learn both rules and probabilistic weights from given data (in form of literals). In contrast, our present approach does not comprise rule learning. However, our weight learning algorithm allows for learning from any kind of formulas and for the specification of virtually any sort of hypothesis as learning target, not only sets of abducibles. Both (Corapi et al. 2011) and our approach employ gradient descent for weight learning. Other approaches to probabilistic logic programming based on the stable model semantics for the logic aspects include (Saad and Pontelli 2005) and (Ng and Subrahmanian 1994). (Saad and Pontelli 2005) appears to be a powerful approach, but restricts probabilistic weighting to certain types of formulas, in order to achieve a low computational reasoning complexity. Its probabilistic annotation scheme is similar to that proposed in (Ng and Subrahmanian 1994). (Ng and Subrahmanian 1994) provides both a language and an in-depth investigation of the stable model semantics (in particular the semantics of non-monotonic negation) of probabilistic deductive databases.

Our approach (and ASP in general) is closely related to SAT solving, #SAT and constraint solving. ASP formulas in our language are constraints for possible worlds (legitimate models). As (Sang, Beame, and Kautz 2005) shows, Bayesian networks can be "translated" into a weighted model counting problem over propositional formulas, which is related to our approach to probabilistic inference, although details are quite different. Also, the XOR constraining approach (Gomes, Sabharwal, and Selman 2006) employed for sampling of answer sets (Section 4) has originally been invented for the sampling of propositional truth assignments.

3 Probabilistic Answer Set Programming with PrASP

Before we turn to probabilistic inference and parameter estimation, we introduce our new language for probabilistic non-monotonic logic programming, called Probabilistic Answer Set Programming (*PrASP*).

Syntax: Just add probabilities

To remove unnecessary syntax restrictions and because we will later require certain syntactic modifications of given programs which are easier to express in First-Order Logic (FOL) notation, we allow for FOL statements in our logic programs, using the F2LP conversion tool (Lee and Palla 2009). More precisely, a *PrASP program* consists of ground or non-ground formulas in unrestricted first-order syntax annotated with numerical *weights* (provided by some domain expert or learned from data). Weights directly represent probabilities. If the weights are removed, and provided finite variable domains, any such program can be converted into an equivalent answer set program by means of the transformation described in (Lee and Palla 2009).

Let Φ be a set of function, predicate and object symbols and $\mathcal{L}(\Phi)$ a first-order language over Φ and the usual connectives (including both strong negation "-" and default negation "not") and first-order quantifiers.

Formally, a PrASP program is a non-empty finite set $\{([p], f_i)\}$ of PrASP *formulas* where each formula $f_i \in$

$\mathcal{L}(\Phi)$ is annotated with a *weight* $[p]$. A weight directly represents a probability (provided it is probabilistically sound). If the weight is omitted for some formula of the program, weight $[1]$ is assumed. The weight p of $[p]$ f is denoted as $w(f)$. Weighted formulas can intuitively seen as constraints which specify which possible worlds are indeed possible, and with which probability.

Let Λ^- denote PrASP program Λ stripped of all weights. Weights need to be probabilistically sound, in the sense that the system of inequalities (1) - (4) in Section 3 must have at least one solution (however, in practice this does not need to be strictly the case, since the constraint solver employed for finding a probability distribution over possible worlds can find approximate solutions often even if the given weights are inconsistent).

In order to translate conjunctions of unweighted formulas in first-order syntax into disjunctive programs with a stable model semantics, we further define transformation $lp : \mathcal{L}(\Phi) \cup dLp(\Phi) \rightarrow dLp(\Phi)$, where $dLp(\Phi)$ is the set of all disjunctive programs over Φ. The details of this transformation can be found in (Lee and Palla 2009)[1]. Applied to rules and facts in ASP syntax, lp simply returns these. This allows to make use of the wide range of advanced possibilities offered by contemporary ASP grounders in addition to FOL syntax (such as aggregates), although when defining the semantics of programs, we consider only formulas in FOL syntax.

Semantics

The probabilities attached to formulas in a PrASP program induce a probability distribution over answer sets of an ordinary answer set program which we call the *spanning program* associated with that PrASP program. Informally, the idea is to transform a PrASP program into an answer set program whose answer sets reflect the nondeterminism introduced by the probabilistic weights: each annotated formula might hold as well as not hold (unless its weight is [0] or [1]). Of course, this transformation is lossy, so we need to memorize the weights for the later computation of a probability distribution over possible worlds. The important aspect of the spanning program is that it programmatically generates a set of possible worlds in form of answer sets.

Technically, the spanning program $\rho(\Lambda)$ of PrASP program Λ is a disjunctive program obtained by transformation $lp(\Lambda')$. We generate Λ' from Λ by removing all weights and transforming each formerly weighted formula f into a disjunction $f|not\ f$, where not stands for default negation and $|$ stands for the disjunction in ASP (so probabilities are "default probabilities" in our framework). Note that $f|not\ f$ doesn't guarantee that answer sets are generated for weighted formula f. By using ASP choice constructs such as aggregates and disjunctions, the user can basically generate as many answer sets (possible worlds) as desired.

[1]The use of the translation into ASP syntax requires either an ASP solver which can deal directly with disjunctive logic programs (such as claspD) or a grounder which is able to shift disjunctions from the head of the respective rules into the bodies, such as gringo (Gebser, Kaufmann, and Schaub 2012).

Formulas do not need to be ground - as defined in Section 3, they can contain existentially as well as universally quantified variables in the FOL sense (although restricted to finite domains).

As an example, consider the following simple *ground* PrASP program (examples for PrASP programs with variables and first-order style quantifiers are presented in the next sections):

```
[0.7] q <- p.
[0.3] p.
[0.2] -p & r.
```

The set of answer sets (which we take as possible worlds) of the spanning program of this PrASP program is $\{\{p, q\}, \{-p, r\}, \{\}, \{p\}\}$.

The semantics of a PrASP program Λ and single PrASP formulas is defined in terms of a probability distribution over a set of possible worlds (in form of answer sets of $\rho(\Lambda)$) in connection with the stable model semantics. This is analogously to the use of *Type 2 probability structures* (Halpern 1990) for first-order probabilistic logics with probabilities, but restricted to finite domains of discourse.

Let $M = (D, \Theta, \pi, \mu)$ be a probability structure where D is a finite discrete domain of objects, Θ is a non-empty set of possible worlds, π a function which assigns to the symbols in Φ (see Section 3) predicates, functions and objects over/from D, and μ a discrete probability function over Θ. Each possible world is a Herbrand interpretation over Φ. Since we will use answer sets as possible worlds, defining $\Gamma(a)$ to be the set of all answer sets of answer set program a will become handy. For example, given $\rho(\Lambda)$ as (uncertain) knowledge, the set of worlds deemed possible according to existing belief $\rho(\Lambda)$ is $\Gamma(\rho(\Lambda))$ in our framework.

We define a (non-probabilistic) satisfaction relation of possible worlds and unannotated programs as follows: let Λ^- be is an unannotated program. Then $(M, \theta) \models_\Theta \Lambda^-$ *iff* $\theta \in \Gamma(lp(\Lambda^-))$ and $\theta \in \Theta$ (from this it follows that Θ induces its own closed world assumption - any answer set which is not in Θ is not satisfiable wrt. $\models_\Theta$). The probability $\mu(\{\theta\})$ of a possible world θ is denoted as $Pr(\theta)$ and sometimes called "weight" of θ. For a disjunctive program ψ, we analogously define $(M, \theta) \models_\Theta \psi$ *iff* $\theta \in \Gamma(\psi)$ and $\theta \in \Theta$.

To do groundwork for the computation of a probability distribution over possible worlds Θ which are "generated" and weighted by some given background knowledge in form of a PrASP program, we define a (non-probabilistic) satisfaction relation of possible worlds and unannotated formulas: let ϕ be a PrASP formula (without weight) and θ be a possible world. Then $(M, \theta) \models_\Lambda \phi$ *iff* $(M, \theta) \models_\Theta \rho(\Lambda) \cup lp(\phi)$ *and* $\Theta = \Gamma(\rho(\Lambda))$ (we say formula ϕ is *true in possible world* θ). Sometimes we will just write $\theta \models_\Lambda \phi$ if M is given by the context. A notable property of this definition is that it does not restrict us to single ground formulas. Essentially, an unannotated formula ϕ can be any answer set program specified in FOL syntax, even if its grounding consists of multiple sentences. Observe that Θ restricts $\models_\Lambda$ to answer sets of $\rho(\Lambda)$. For convenience, we will abbreviate $(M, \theta) \models_\Lambda \phi$ as $\theta \models_\Lambda \phi$.

$Pr(\phi)$ denotes the probability of a formula ϕ, with

$Pr(\phi) = \mu(\{\theta \in \Theta : (M, \theta) \models_\Lambda \phi\})$. Note that this holds both for annotated and unannotated formulas: even if it has a weight attached, the probability of a PrASP formula is defined by means of μ and only indirectly by its manually assigned weight (weights are used below as constraints for the computation of a probabilistically consistent μ). Further observe that there is no particular treatment for conditional probabilities in our framework; $Pr(a|b)$ is simply calculated as $Pr(a \wedge b)/Pr(b)$.

While our framework so far is general enough to account for probabilistic inference using unrestricted programs and query formulas (provided we are given a probability distribution over the possible answer sets), this generality also means a relatively high complexity in terms of computability for inference-heavy tasks which rely on the repeated application of operator $\models_\Lambda$, even if we would avoid the transformation lp and restrict ourselves to the use of ASP syntax.

The obvious question now, addressed before for other probabilistic logics, is how to compute μ, i.e., how to obtain a probability distribution over possible worlds (which tells us for each possible world the probability with which this possible world is the actual world) from a given annotated program Λ in a sound and computationally inexpensive way.

Generally, we can express the search for probability distributions in form of a number of constraints which constitute a system of linear inequalities (which reduce to linear equalities for point probabilities as weights). This system typically has multiple or even infinitely many solutions (even though we do not allow for probability intervals) and computation can be costly, depending on the number of possible worlds according to $\rho(\Lambda)$.

We define the parameterized probability distribution $\mu(\Lambda, \Theta)$ over a set Θ of answer sets as the solution (for all $Pr(\theta_i)$) of the following system of linear equations and an inequality (if precisely one solution exists) or as the solution with maximum entropy (Thimm and Kern-Isberner 2012), in case multiple solutions exist [2]. We require that the given weights in a PrASP program are chosen such that the following constraint system has at least one solution.

$$\sum_{\theta_i \in \Theta : \theta_i \models_\Lambda f_1} Pr(\theta_i) = w(f_1) \quad (1)$$

$$\cdots$$

$$\sum_{\theta_i \in \Theta : \theta_i \models_\Lambda f_n} Pr(\theta_i) = w(f_n) \quad (2)$$

$$\sum_{\theta_i \in \Theta} \theta_i = 1 \quad (3)$$

$$\forall \theta_i \in \Theta : 0 \leq Pr(\theta_i) \leq 1 \quad (4)$$

At this, $\Lambda = \{f_1, ..., f_n\}$ is a PrASP program.

The *canonical probability distribution* $\mu(\Lambda)$ of Λ is defined as $\mu(\Lambda, \Gamma(\rho(\Lambda)))$. In the rest of the paper, we refer to $\mu(\Lambda)$ when we refer to the probability distribution over the answer sets of the spanning program of a given PrASP program Λ.

[2] Since in this case the number of solutions of the system of linear equations is infinite, de facto we need to choose the maximum entropy solution of some finite subset. In the current prototype implementation, we generate a user-defined number of random solutions derived from a solution computed using a constrained variant of Singular Value Decomposition and the null space of the coefficient matrix of the system of linear equations (1)-(3).

4 Inference

Given possible world weights ($\mu(\Lambda)$), probabilistic inference becomes a model counting task where each model has a weight: we can compute the probability of any query formula ϕ by summing up the probabilities (weights) of those possible worlds (models) where ϕ is true. To make this viable even for larger sets of possible worlds, we optionally restrict the calculation of $\mu(\Lambda)$ to a number of answer sets sampled near-uniformly at random from the total set of answer sets of the spanning program, as described in Section 4.

Adding a sampling step and computing probabilities

All tasks described so far (solving the system of (in)equalities, counting of weighted answer sets) become intractable for very large sets of possible worlds. To tackle this issue, we want to restrict the application of these tasks to a sampled subset of all possible worlds. Concretely, we want to find a way to sample (near-)uniformly from the total set of answer sets *without* computing a very large number of answer sets. While this way the set of answer sets cannot be computed using only a single call of the ASP solver but requires a number of separate calls (each with different sampling constraints), the required solver calls can be performed *in parallel*. However, a shortcoming of the sampling approach is that there is currently no way to pre-compute the size of the minimally required set of samples.

Guaranteeing near-uniformity in answer set sampling looks like a highly non-trivial task, since any set of answers obtained from ASP solvers as a subset of the total set of answer sets is typically not uniformly distributed but strongly biased in hardly foreseeable ways (due to various interplaying heuristics applied by modern solvers), so we could not simply request any single answer set from the solver.

However, we can make use of so-called *XOR constraints* (a form of streamlining constraints in the area of SAT solving) for near-uniform sampling (Gomes, Sabharwal, and Selman 2006) to obtain samples from the space of all answer sets, within arbitrarily narrow probabilistic bounds, using any off-the-shelf ASP solver. Compared to approaches which use Markov Chain Monte Carlo (MCMC) methods to sample from some given distribution, this method has the advantage that the sampling process is typically faster and that it requires only an off-the-shelf ASP solver (which is in the ideal case employed only once per sample, in order to obtain a single answer set). However, a shortcoming is that we are not doing Importance Sampling this way - the probability of a possible world is not taken into account but computed later from the samples.

Counting answer sets could also be achieved using XOR constraints, however, this is not covered in this paper, since it does not comprise weighted counting, and we could normally not use an unweighted counting approach directly.

XOR constraints were originally defined over a set of propositional variables, which we identify with a set of ground atoms $V = \{a_1, ..., a_n\}$. Each XOR constraint is represented by a subset D of $V \cup \{true\}$. D is satisfied by some model if an *odd* number of elements of D are satisfied by this model (i.e., the constraint acts like a parity of D). In ASP syntax, an XOR constraint can be represented for example as `:- #even{ a1, ..., an }` (Gebser et al. 2011).
In our approach, XOR constraints are independently at random drawn from a probability distribution $\mathbb{X}(|V|, 0.5)$ over the set V of all possible XOR constraints over all ground atoms of the ground answer set program resulting from $\rho(\Lambda)$. $\mathbb{X}(|V|, 0.5)$ is defined such that each XOR constraint is drawn from this distribution independently at random with probability 0.5 and includes $true$ with probability 0.5. In effect, any given XOR constraint is drawn with probability $2^{-(|V|+1|)}$ (see (Gomes, Sabharwal, and Selman 2006) for details). Since adding an XOR constraint to an answer set program eliminates any given answer set with probability 0.5, it cuts the set of answer sets in half in expectation. Iteratively adding a small number of XOR constraints to an answer set program therefore reduces the number of answer sets to a small number also. If this process results in a single answer set, the remaining answer set is drawn near-uniformly from the original set of answer sets, as shown in (Gomes, Sabharwal, and Selman 2006).
Since for answer set programs the costs of repeating the addition of constraints until precisely a single answer set remains appears to be higher than the costs of computing somewhat too many models, we just estimate the number of required constraints and choose randomly from the resulting set of answer sets. The following way of answer set sampling using XOR constraints has been used before in Xorro (a tool which is part of the *Potassco* set of ASP tools (Gebser et al. 2011)) in a very similar way.

Function ***sample***: $\psi \mapsto \gamma$

Given any disjunctive program ψ, the following procedure computes a random sample γ from the set of all answer sets of ψ:
$\psi_g \leftarrow$ ground(ψ)
$ga \leftarrow atoms(\psi_g)$
$xors \leftarrow$ XOR constraints $\{xor_1, ..., xor_n\}$ over ga, drawn from $\mathbb{X}(|V|, 0.5)$
$\psi' \leftarrow \psi \cup xors$
$\gamma \leftarrow$ an answer set selected randomly from $\Gamma(\psi')$

At this, the number of constraints n is set to a value large enough to produce one or a very low number of answer sets ($log_2(|ga|)$ in our experiments).

We can now compute $\mu(\Lambda, \Theta')$ (i.e., $Pr(\theta)$ for each $\theta \in \Theta'$) for a set of samples Θ' obtained by multiple (ideally parallel) calls of *sample* from the spanning program $\rho(\Lambda)$ of PrASP program Λ, and subsequently sum up the weights of those samples (possible worlds) where the respective query formula (whose marginal probability we want to compute) is true. Precisely, we approximate $Pr(\phi)$ for a (ground or non-ground) query formula ϕ using:

$$Pr(\phi) \approx \sum_{\{\theta' \in \Theta' : \theta' \models_\Lambda \phi\}} Pr(\theta') \quad (5)$$

for a sufficiently large set Θ' of samples.
Conditional probabilities $Pr(a|b)$ can simply be computed as $Pr(a \wedge b)/Pr(b)$.

If sampling is not useful (i.e., if the total number of answer sets Θ is moderate), inference is done in the same way, we just set $\Theta' = \Theta$. Sampling using XOR constraints costs time too (mainly because of repeated calls of the ASP solver), and making this approach more efficient is an important aspect of future work (see Section 6).

As an example for inference using our current implementation, consider the following PrASP formalization of a simple coin game:

```
coin(1..3).
[0.6] coin_out(1,heads).
[[0.5]] coin_out(N,heads) :- coin(N), N != 1.
1{coin_out(N,heads), coin_out(N,tails)}1
         :- coin(N).
n_win :- coin_out(N,tails), coin(N).
win :- not n_win.
```

At this, the line starting with `[[0.5]]...` is syntactic sugar for a set of weighted rules where variable N is instantiated with all its possible values (i.e.,
`[0.5] coin_out(2,heads) :- coin(2), 2 != 1`
and
`[0.5] coin_out(3,heads) :- coin(3), 3 != 1`).
It would also be possible to use `[0.5]` as annotation of this rule, in which case the weight 0.5 would specify the probability of the whole non-ground formula instead.
Our prototypical implementation accepts query formulas in format `[?] a` (computes the marginal probability of a) and `[?|b] a` (computes the conditional probability $Pr(a|b)$). E.g.,

```
[?] coin_out(1,tails).
[?] coin_out(1,heads) | coin_out(1,tails).
[?] coin_out(1,heads) & coin_out(2,heads)
        & coin_out(3,heads).
[?] win.
[?|coin_out(1,heads) & coin_out(2,heads)
          coin_out(3,heads)] win.
```

...yields the following result

```
[0.3999999999999999] coin_out(1,tails).
[1] coin_out(1,heads) | coin_out(1,tails).
[0.15] coin_out(1,heads) & coin_out(2,heads)
       & coin_out(3,heads).
[0.15] win.
[1|coin_out(1,heads) & coin_out(2,heads)
   & coin_out(3,heads)] win.
```

In this example, use of sampling does not make any difference due to its small size. An example where a difference can be observed is presented in Section 5. This example also demonstrates that FOL and logic programming / ASP syntax can be freely mixed in background knowledge and queries.
Another simple example shows the use of FOL-style variables and quantifiers mixed with ASP-style variables:

```
p(1). p(2). p(3).
#domain p(X).
[0.5] v(1).
[0.5] v(2).
[0.5] v(3).
[0.1] v(X).
```

With this, the following query:

```
[?] v(X).
#domain p(Z).
[?] ![Z]: v(Z).
[?] ?[Z]: v(Z).
```

...results in:

```
[0.1] ![Z]: v(Z).
[0.8499999999999989] ?[Z]: v(Z).
```

The result of query `[?] ![Z]: v(Z)` with universal quantifier `![Z]` is $Pr(\forall z.v(z)) = 0.1$, which is also the result of the equivalent queries `[?] v(1) & v(2) & v(3)` and `[?] v(X).` In our example, this marginal probability was directly given as weight in the background knowledge. In contrast to `X`, variable `Z` is a variable in the sense of first-order logic (over a finite domain).
The result of `?[Z]: v(Z)` is $Pr(\exists z.v(z))$ (i.e., `?[Z]:` represents the existential quantifier) and could likewise be calculated manually using the inclusion-exclusion principle as $Pr(v(1) \vee v(2) \vee v(3)) = Pr(v(1)) + Pr(v(2)) + Pr(v(3)) - Pr(v(1) \wedge v(2)) - Pr(v(1) \wedge v(3)) - Pr(v(2) \wedge v(3)) + Pr(v(1) \wedge v(2) \wedge v(3)) = 0.85$.
Of course, existential or universal quantifiers can also be used as sub-formulas and in PrASP programs.

An alternative approach: conversion into an equivalent non-probabilistic answer set program

An alternative approach to probabilistic inference without computing μ and without counting of weighted possible worlds, would be to find an unannotated first-order program Λ' which reflects the desired probabilistic nondeterminism (choice) of a given PrASP program Λ. Instead of defining probabilities of possible worlds, Λ' has answers sets whose frequency (number of occurrences within the total set of answer sets) reflects the given probabilities in the original (annotated) program. To make this idea more intuitive, imagine that each possible world corresponds to a room. Instead of encountering a certain room with a certain frequency, we create further rooms which have all, from the viewpoint of the observer, the same look, size and furniture. The number of these rooms reflects the probability of this type of room. E.g., to ensure probability $\frac{1}{3}$ of some literal p, Λ' is created in a way such that p holds in one third of all answer sets of Λ'. This task can be considered as an elaborate variant of the generation of the (much simpler) spanning program $\rho(\Lambda)$.

Finding Λ' could be formulated as an (intractable) rule search problem (plus subsequently the conversion into ASP syntax and a simple unweighted model counting task): find a non-probabilistic program Λ' such that for each annotated formula $[p]f$ in the original program the following holds (under the provision that the given weights are probabilistically sound):

$$\frac{|\{m : m \in \Gamma(\Lambda'), m \models f\}|}{|\Gamma(\Lambda')|} = p. \quad (6)$$

Unfortunately, the direct search approach to this would be obviously intractable.

However, in the special case of mutually independent formulas we can omit the rule learning task by conditioning each formula in Λ by a nondeterministic choice amongst the truth conditions of a number of "helper atoms" h_i (which will later be ignored when we count the resulting answer sets), in order to "emulate" the respective probability specified by the weight. If (and only if) the formulas are mutually independent, the obtained Λ' is isomorphic to the original probabilistic program. In detail, conditioning means to replace each formula $[w]\ f$ by formulas $1\{h_1, ..., h_n\}1$, $f \leftarrow h_1|...|h_m$ and $not\ f \leftarrow not\ (h_1|...|h_m)$, where the h_i are new names (the aforementioned "helper atoms"), $\frac{m}{n} = w$ and $m < n$ (remember that we allow for weight constraints as well as FOL syntax).

In case the transformation accurately reflects the original uncertain program, we could now calculate marginal probabilities simply by determining the percentage of those answer sets in which the respective query formula is true (ignoring any helper atoms introduced in the conversion step), with no need for computing $\mu(\Lambda)$.
As an example, consider the following program:

```
coin(1..10).
[0.6] coin_out(1,heads).
[[0.5]] coin_out(N,heads) :- coin(N), N != 1.

1{coin_out(N,heads), coin_out(N,tails)}1
  :- coin(N).
n_win :- coin_out(N,tails), coin(N).
win :- not n_win.
```

Since coin tosses are mutually independent, we can transform it into the following equivalent un-annotated form (the $\texttt{hpatom}_n$ are the "helper atoms". Rules are written as disjunctions):

```
coin(1..10).
1{hpatom1,hpatom2,hpatom3,hpatom4,hpatom5}1.
(coin_out(1,heads))
 | -(hpatom1|hpatom2|hpatom3).
not (coin_out(1,heads))
 | (hpatom1|hpatom2|hpatom3).
1{hpatom6,hpatom7}1.
(coin_out(10,heads)) | -(hpatom6).
not (coin_out(10,heads)) | (hpatom6).
1{hpatom8,hpatom9}1.
(coin_out(9,heads)) | -(hpatom8).
not (coin_out(9,heads)) | (hpatom8).
1{hpatom10,hpatom11}1.
(coin_out(8,heads)) | -(hpatom10).
not (coin_out(8,heads)) | (hpatom10).
1{hpatom12,hpatom13}1.
(coin_out(7,heads)) | -(hpatom12).
not (coin_out(7,heads)) | (hpatom12).
1{hpatom14,hpatom15}1.
(coin_out(6,heads)) | -(hpatom14).
```

```
not (coin_out(6,heads)) | (hpatom14).
1{hpatom16,hpatom17}1.
(coin_out(5,heads)) | -(hpatom16).
not (coin_out(5,heads)) | (hpatom16).
1{hpatom18,hpatom19}1.
(coin_out(4,heads)) | -(hpatom18).
not (coin_out(4,heads)) | (hpatom18).
1{hpatom20,hpatom21}1.
(coin_out(3,heads)) | -(hpatom20).
not (coin_out(3,heads)) | (hpatom20).
1{hpatom22,hpatom23}1.
(coin_out(2,heads)) | -(hpatom22).
not (coin_out(2,heads)) | (hpatom22).
1{coin_out(N,heads), coin_out(N,tails)}1
  :- coin(N).
n_win :- coin_out(N,tails), coin(N).
win :- not n_win.
```

Exemplary query results:

```
[0.001171875] win.
[0.998828125] not win.
[0.6] coin_out(1,heads).
[0.5] coin_out(2,heads).
```

What is remarkable here is that no equation solving task (computation of $\mu(\Lambda)$) is required to compute these results. However, this does not normally lead to improved inference speed, due to the larger amount of time required for the computation of models.

5 Weight Learning

Generally, the task of parameter learning in probabilistic inductive logic programming is to find probabilistic parameters (weights) of logical formulas which maximize the likelihood given some data (learning examples) (Raedt and Kersting 2008). In our case, the hypothesis H (a set of formulas without weights) is provided by an expert, optionally together with some PrASP program as background knowledge B. The goal is then to discover weights w of the formulas H such that $Pr(E|H_w \cup B)$ is maximized given example formulas $E = e_1, e_2, ...$ Formally, we want to compute

$$argmax_w(Pr(E|H_w \cup B)) = argmax_w(\prod_{e_i \in E} Pr(e_i|H_w \cup B)) \tag{7}$$

(Making the usual i.i.d. assumption regarding the individual examples in E. H_w denotes the hypothesis weighted with weight vector w.)

This results in an optimization task which is related but not identical to weight learning for, e.g., MLNs and (Corapi et al. 2011). In MLNs, typically a database (possible world) is given whose likelihood should be maximized, e.g. using a generative approach (Lowd and Domingos 2007) by gradient descent. Another related approach distinguishes a priori between evidence atoms X and query atoms Y and seeks to maximize the likelihood $Pr(Y|X)$, again using gradient descent (Huynh and Mooney 2008). At this, cost-heavy inference is avoided as far as possible, e.g., by optimization of the pseudo-(log-)likelihood instead ot the (log-)likelihood or by approximations of costly counts of true formula groundings in a certain possible world (the basic computation in MLN inference). In contrast, the current implementation of PrASP learns weights from any formulas and not just literals (or, more precisely as for MLNs: atoms, where negation is implicit using a closed-world assumption). Furthermore, the maximization targets are different ($Pr(possible\ world)$ or $Pr(Y|X)$) vs. $Pr(E|H_w \cup B)$).

Regarding the need to reduce inference when learning, PrASP parameter estimation should in principle make no exception, since inference can still be costly even when probabilities are inferred only approximately by use of sampling. However, in our preliminary experiments we found that at least in relatively simple scenarios, there is no need to resort to inference-free approximations such as pseudo-(log-)likelihood. The pseudo-(log-)likelihood approach presented in early works on MLNs (Richardson and Domingos 2006) would also require a probabilistic ground formula independence analysis in our case, since in PrASP there is no obvious equivalent to Markov blankets.

Note that we assume that the example data is non-probabilistic and fully observable.

Let $H = \{f_1, ..., f_n\}$ be a given set of formulas and a vector $w = (w^1, ..., w^n)$ of (unknown) weights of these formulas. Using the Barzilai and Borwein method (Barzilai and Borwein 1988) (a variant of the gradient descent approach with possibly superlinear convergence), we seek to find w such that $Pr(E|H_w \cup B)$ is maximized (H_w denotes the formulas in H with the weights w such that each f_i is weighted with w^i). Any existing weights of formulas in the background knowledge ar not touched, which can significantly reduce learning complexity if H is comparatively small. Probabilistic or unobservable examples are not considered.

The learning algorithm (Barzilai and Borwein 1988) is as follows:

Repeat for $k = 0, 1, ...$ until convergence:
Set $s_k = \frac{1}{\alpha_k} \nabla(Pr(E|H_{w_k} \cup B))$
Set $w_{k+1} = w_k + s_k$
Set $y_k = \nabla(Pr(E|H_{w_{k+1}} \cup B)) - \nabla(Pr(E|H_{w_k} \cup B))$

Set $\alpha_{k+1} = \frac{s_k^T y_k}{s_k^T s_k}$

At this, the initial gradient ascent step size α_0 and the initial weight vector w_0 can be chosen freely. $Pr(E|H_w \cup B)$ denotes $\prod_{e_i \in E} Pr(e_i|H_w \cup B)$ inferred using vector w as weights for the hypothesis formulas, and

$$\nabla(Pr(E|H_w \cup B)) = \tag{8}$$

$$(\frac{\partial}{\partial w^1} Pr(E|H_w \cup B), ..., \frac{\partial}{\partial w^n} Pr(E|H_w \cup B)) \tag{9}$$

Since we usually cannot practically express $Pr(E|H_w \cup B)$ in dependency of w in closed form, at a first glance, the above formalization appears to be not very helpful. However, we can still resort to numerical differentiation and approximate

$$\nabla(Pr(E|H_w \cup B)) = \tag{10}$$

$$(\lim_{h \to 0} \frac{Pr(E|H_{(w^1+h,...,w^n)} \cup B) - Pr(E|H_{(w^1,...,w^n)} \cup B)}{h}, \tag{11}$$

$$..., \lim_{h \to 0} \frac{Pr(E|H_{(w^1,...,w^n+h)} \cup B) - Pr(E|H_{(w^1,...,w^n)} \cup B)}{h}) \quad (12)$$

by computing the above vector (dropping the limit operator) for a sufficiently small h (in our prototypical implementation, $h = \sqrt{\epsilon} w_i$ is used, where ϵ is an upper bound to the rounding error using the machine's double-precision floating point arithmetic).
This approach has the benefit of allowing in principle for any maximization target (not just E). In particular, any unweighted formulas (unnegated and negated facts as well as rules) can be used as (positive) examples.

As a small example both for inference and weight learning using our preliminary implementation, consider the following fragment of a an nonmonotonic indoor localization scenario, which consists of estimating the position of a person, and determining how this person moves a certain number of steps around the environment until a safe position is reached:

```
[0.6] moved(1).
[0.2] moved(2).
point(1..100).
1{atpoint(X):point(X)}1.
distance(1) :- moved(1).
distance(2) :- moved(2).
atpoint(29) | atpoint(30) | atpoint(31)
   | atpoint(32) | atpoint(33)
   | atpoint(34) | atpoint(35) | atpoint(36)
   | atpoint(37) -> selected.
safe :- selected, not exception.
exception :- distance(1).
```

The spanning program of this example has 400 answer sets. Inference of
$Pr(safe|distance(2))$ and $Pr(safe|distance(1))$ without sampling requires ca. 2250 ms using our current unoptimized prototype implementation. If we increase the number of points to 1000, inference is tractable only by use of sampling (see Section 4).
To demonstrate how the probability of a certain hypothesis can be learned in this simple scenario, we remove `[0.6] moved(1)` from the program above (with 100 points) and turn this formula (without the weight annotation) into a hypothesis. Given example data `safe`, parameter estimation results in $Pr(moved(1)) \approx 0$, learned in ca. 3170 ms using our current prototype implementation.

6 Conclusions

With this introductory paper, we have presented a novel framework for uncertainty reasoning and parameter estimation based on Answer Set Programming, with support for probabilistically weighted formulas in background knowledge, hypotheses and queries. While our current framework certainly leaves room for future improvements, we believe that we have already pointed out a new venue towards more practicable probabilistic inductive answer set programming with a high degree of expressiveness. Ongoing work is focusing on performance improvements, theoretical analysis (in particular regarding minimum number of samples wrt. inference accuracy), empirical evaluation and on the investigation of viable approaches to PrASP structure learning.

Acknowledgments
This work is supported by the EU FP7 CityPulse Project under grant No. 603095. http://www.ict-citypulse.eu

References

Bacchus, F. 1990. l_p, a logic for representing and reasoning with statistical knowledge. *Computational Intelligence* 6:209–231.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic reasoning with answer sets. *Theory Pract. Log. Program.* 9(1):57–144.

Barzilai, J., and Borwein, J. M. 1988. Two point step size gradient methods. *IMA J. Numer. Anal.*

Corapi, D.; Sykes, D.; Inoue, K.; and Russo, A. 2011. Probabilistic rule learning in nonmonotonic domains. In *Proceedings of the 12th international conference on Computational logic in multi-agent systems*, CLIMA'11, 243–258. Berlin, Heidelberg: Springer-Verlag.

Cussens, J. 2000. Parameter estimation in stochastic logic programs. In *Machine Learning*, 2001.

Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 1999. Learning probabilistic relational models. In *In IJCAI*, 1300–1309. Springer-Verlag.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The potsdam answer set solving collection. *AI Commun.* 24(2):107–124.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th Int'l Conference on Logic Programming*, volume 161.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Near-uniform sampling of combinatorial spaces using xor constraints. In *NIPS*, 481–488.

Halpern, J. Y. 1990. An analysis of first-order logics of probability. *Artificial Intelligence* 46:311–350.

Huynh, T. N., and Mooney, R. J. 2008. Discriminative structure and parameter learning for markov logic networks. In *25th Int. Conf. on*, 416–423.

Kersting, K., and Raedt, L. D. 2000. Bayesian logic programs. In *Proceedings of the 10th International Conference on Inductive Logic Programming*.

Laskey, K. B., and Costa, P. C. 2005. Of klingons and starships: Bayesian logic for the 23rd century. In *Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence*.

Lee, J., and Palla, R. 2009. System f2lp - computing answer sets of first-order formulas. In Erdem, E.; Lin, F.; and

Schaub, T., eds., *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, 515–521. Springer.

Lifschitz, V. 2002. Answer set programming and plan generation. *AI* 138(1):39–54.

Lowd, D., and Domingos, P. 2007. Efficient weight learning for markov logic networks. In *In Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, 200–211.

Muggleton, S. 2000. Learning stochastic logic programs. *Electron. Trans. Artif. Intell.* 4(B):141–153.

Ng, R. T., and Subrahmanian, V. S. 1994. Stable semantics for probabilistic deductive databases. *Inf. Comput.* 110(1):42–83.

Nilsson, N. J. 1986. Probabilistic logic. *Artificial Intelligence* 28(1):71–87.

Poole, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence* 94:7–56.

Raedt, L. D., and Kersting, K. 2008. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, 1–27.

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2462–2467.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2):107–136.

Saad, E., and Pontelli, E. 2005. Hybrid probabilistic logic programming with non-monotoic negation. In *In Twenty First International Conference on Logic Programming*. Springer Verlag.

Sang, T.; Beame, P.; and Kautz, H. A. 2005. Performing bayesian inference by weighted model counting. In *AAAI*, 475–482.

Sato, T., and Kameya, Y. 1997. Prism: a language for symbolic-statistical modeling. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97*, 1330–1335.

Thimm, M., and Kern-Isberner, G. 2012. On probabilistic inference in relational conditional logics. *Logic Journal of the IGPL* 20(5):872–908.

A Plausibility Semantics for Abstract Argumentation Frameworks

Emil Weydert

Individual and Collective Reasoning Group
ILIAS-CSC, University of Luxembourg

Abstract

We propose and investigate a simple plausibility-based extension semantics for abstract argumentation frameworks based on generic instantiations by default knowledge bases and the ranking construction paradigm for default reasoning.[1]

1 Prologue

The past decade has seen a flourishing of abstract argumentation theory, a coarse-grained high-level form of defeasible reasoning introduced by Dung [Dung 95]. It is characterized by a top-down perspective which ignores the logical fine structure of arguments and focuses instead on logical (conflict, support, ...) or extra-logical (preferences, ...) relations between given black box arguments so as to identify reasonable argumentative positions. One way to address the complexity of enriched argument structures carrying interacting relations, and to identify the best approaches for evaluating Dung's basic attack frameworks as well as more sophisticated argumentation systems, is to look for deeper unifying semantic foundations allowing us to improve, compare, and judge existing proposals, or to develop new ones.

A major issue is to what extent an abstract account can adequately model concrete argumentative reasoning in the context of a sufficiently expressive, preferably defeasible logic. The instantiation of abstract frameworks by more fine-grained logic-based argument constructions and configurations is therefore an important tool for justifying or criticising abstract argumentation theories. Most of this work is however based on the first generation of nonmonotonic formalisms, like Reiter's default logic or logic programming. While these are closer to classical logic and the original spirit of Dung's approach, it is well known that they fail to model plausible implication. In fact, they are haunted by counterintuitive behaviour and violate major desiderata for default reasoning encoded in benchmark examples and rationality postulates [Mak 94]. For instance, the only way to deal even with simple instances of specificity reasoning are opaque ad hoc prioritization mechanisms.

[1] This is an improved - polished and partly revised - version of my ECSQARU 2013 paper. It adds a link to structured argumentation, refines the semantic instantiation concept, and discusses attacks between inference pairs.

The goal of the present work is therefore to supplement existing instantiation efforts with a simple ranking-based semantic model which interprets arguments and attacks by conditional knowledge bases. The well-behaved ranking construction semantics for default reasoning [Wey 96, 98, 03] can then be exploited to specify a new extension semantics for Dung frameworks which allows us to directly evaluate the plausibility of argument collections. Its occasionally unorthodox behaviour may shed a new light on basic argumentation-theoretic assumptions and concepts.

We start with an introduction to default reasoning based on the ranking construction paradigm. After a short look at abstract argumentation theory, we show how to interpret abstract argumentation frameworks by instantiating the arguments and characterizing the attacks with suitable sets of conditionals describing constraints over ranking models. Based on the concept of generic instantiations, i.e. using minimal assumptions, and plausibility maximization, we then specify a natural ranking-based extension semantics. We conclude with a simple algorithm, some instructive examples, and the discussion of several important properties.

2 Ranking-based default reasoning

We assume a basic language L closed under the usual propositional connectives, together with a classical satisfaction relation $\models$ inducing a monotonic entailment relation $\vdash \subseteq 2^L \times L$. The model sets of $(L, \models)$ are denoted by $[\![\varphi]\!] = \{m \mid m \models \varphi\}$, resp. $[\![\Sigma]\!] = \cap_{\varphi\in\Sigma}[\![\varphi]\!]$ for $\Sigma \subseteq L$. $\mathcal{B}_L$ is the boolean proposition algebra over $\mathbb{B}_L = \{[\![\varphi]\!] \mid \varphi \in L\}$. Let $Cn(\Sigma) = \{\psi \mid \Sigma \vdash \psi\}$.

Default inference is an important instance of nonmonotonic reasoning concerned with drawing reasonable but potentially defeasible conclusions from knowledge bases of the form $\Sigma \cup \Delta$, where $\Sigma \subseteq L$ is a set of assumptions or facts, e.g. encoding knowledge about a specific state of affairs in the domain language L, and $\Delta \subseteq L(\twoheadrightarrow, \rightsquigarrow)$ is a collection of conditionals expressing strict or exception-tolerant implicational information over L, which is used to guide defeasible inference. $L(\twoheadrightarrow, \rightsquigarrow) = \{\varphi \twoheadrightarrow \psi \mid \varphi, \psi \in L\} \cup \{\varphi \rightsquigarrow \psi \mid \varphi, \psi \in L\}$ is the corresponding flat conditional language on top of L. In the following we will focus on finite Σ and Δ. $\Delta^{\rightarrow} = \{\varphi \rightarrow \psi \mid \varphi \twoheadrightarrow \psi, \varphi \rightsquigarrow \psi\}$ collects the material implications corresponding to the conditionals in Δ.

The strict implication $\varphi \twoheadrightarrow \psi$ states that φ necessarily

implies ψ, forcing us to accept ψ given φ. The default implication $\varphi \rightsquigarrow \psi$ tells us that φ plausibly/normally implies ψ, and only recommends the acceptance of ψ given φ. The actual impact of a default depends of course on the context $\Sigma \cup \Delta$ and the chosen nonmonotonic inference concept $\mid\!\sim$, which will be discussed later.

We can distinguish two perspectives in default reasoning: the autoepistemic/context-based one, and the plausibilistic/quasi-probabilistic one. The former is exemplified by Reiter's default logic, where defaults are usually modeled by normal default rules of the form $\varphi : \psi/\psi$ (if φ, and it is consistent that ψ, then ψ). A characteristic feature is that the conclusions are obtained by intersecting suitable equilibrium sets, known as extensions.

The alternative is to use default conditionals interpreted by some preferential or valuational semantics, e.g. System Z [Pea 90, Leh 92], or probabilistic ME-based accounts [GMP 93] (ME = maximum-entropy). For historical reasons and technical convenience (closeness to classical logic), the first approach has received most attention, especially in the context of argumentation. However, this ignores the fact that the conditional semantic paradigm has a much better record when it comes to the natural handling of benchmark examples and the satisfaction of rationality postulates [Mak 94]. It therefore seems promising to investigate whether such semantic-based accounts can also help to instantiate and evaluate abstract argumentation frameworks.

Our default conditional semantics for interpreting argumentation frameworks is based on the simplest plausibility measure concept able to reasonably handle independence and conditionalization, namely Spohn's ranking functions [Spo 88, 12], or more generally, ranking measures [Wey 94]. These are quasi-probabilistic belief valuations expressing the degree of surprise or implausibility of propositions. Integer-valued ranking functions were originally introduced by Spohn to model the iterated revision of graded plain belief. We will consider $[0, \infty]_{real}$-valued ranking measures[2], where ∞ expresses doxastic impossibility.

Definition 2.1 (Ranking measures)
A map $R : \mathcal{B}_L \rightarrow ([0, \infty], 0, \infty, +, \geq)$ is called a real-valued ranking measure iff $R([\![\mathbf{T}]\!]) = 0$, $R([\![\mathbf{F}]\!]) = R(\emptyset) = \infty$, and for all $A, B \in \mathbb{B}_L$, $R(A \cup B) = \min_{\leq}\{R(A), R(B)\}$. $R(.|.)$ is the associated conditional ranking measure defined by $R(B|A) = R(A \cap B) - R(A)$ if $R(A) \neq \infty$, else $R(B|A) = \infty$. R_0 is the uniform ranking measure, i.e. $R_0(A) = 0$ for $A \neq \emptyset$. If $\mathcal{B} = \mathcal{B}_L$, we will use the abbreviation $R(\varphi) := R([\![\varphi]\!])$.

For instance, the order of magnitude reading interprets ranking measure values $R(A)$ as exponents of infinitesimal probabilities $P(A) = p_A \varepsilon^{R(A)}$, which explains the parallels with probability theory. The monotonic semantics of our conditionals $\twoheadrightarrow, \rightsquigarrow$ is based on the satisfaction relation $\models_{rk}$. The corresponding truth conditions are

- $R \models_{rk} \varphi \twoheadrightarrow \psi$ iff $R(\varphi \wedge \neg\psi) = \infty$.
- $R \models_{rk} \varphi \rightsquigarrow \psi$ iff $R(\varphi \wedge \psi) + 1 \leq R(\varphi \wedge \neg\psi)$.

[2] Although for us, rational values would actually be sufficient.

That is, we assume that a strict implication $\varphi \twoheadrightarrow \psi$ states that $\varphi \wedge \neg\psi$ is doxastically impossible.

Note that we may replace $\varphi \twoheadrightarrow \psi$ by $\varphi \wedge \neg\psi \rightsquigarrow \mathbf{F}$, i.e. it would be actually enough to consider $L(\rightsquigarrow)$. We use $\leq$ with a threshold because this provides more discriminatory power and also guarantees the existence of minima for relevant ranking construction procedures. The exchangeability of arbitrary $r, r' \neq 0, \infty$ by automorphisms allows us to focus, by convention, on the threshold 1. For $\Delta \cup \{\delta\} \subseteq L(\twoheadrightarrow, \rightsquigarrow)$, we set

$$[\![\Delta]\!]_{rk} = \{R \mid R \models_{rk} \Delta\},\ \Delta \vdash_{rk} \delta \text{ iff } [\![\Delta]\!]_{rk} \subseteq [\![\delta]\!]_{rk}.$$

$\vdash_{rk}$ is monotonic and verifies the axioms and rules of preferential conditional logic and disjunctive rationality (threshold semantics: no rational monotony) for $\rightsquigarrow$ [KLM 90].

But it is important to understand that the central concept in default reasoning is not some monotonic conditional logic for $L(\twoheadrightarrow, \rightsquigarrow)$, but a nonmonotonic meta-level inference relation $\mid\!\sim$ over $L \cup L(\twoheadrightarrow, \rightsquigarrow)$ specifying which conclusions $\psi \in L$ can be plausibly inferred from finite $\Sigma \cup \Delta \subseteq L \cup L(\twoheadrightarrow, \rightsquigarrow)$. We write $\Sigma \cup \Delta \mid\!\sim \psi$, or alternatively $\Sigma \mid\!\sim_\Delta \psi$, and set $C_\Delta^{\mid\!\sim}(\Sigma) = \{\psi \mid \Sigma \mid\!\sim_\Delta \psi\}$.

The ranking semantics for plausibilistic default reasoning is based on nonmonotonic ranking choice operators $\mathcal{I}$ which map each finite $\Delta \subseteq L(\twoheadrightarrow, \rightsquigarrow)$ to a collection $\mathcal{I}(\Delta) \subseteq [\![\Delta]\!]_{rk}$ of preferred ranking models of Δ. A corresponding ranking-based default inference notion $\mid\!\sim^{\mathcal{I}}$ can then be specified by

$$\Sigma \mid\!\sim_\Delta^{\mathcal{I}} \psi \text{ iff for all } R \in \mathcal{I}(\Delta), R \models_{rk} \wedge\Sigma \rightsquigarrow \psi.$$

Similarly, we can also define a monotonic inference concept characterizing the strict consequences.

$$\Sigma \vdash_\Delta^{\mathcal{I}} \psi \text{ iff for all } R \in \mathcal{I}(\Delta), R \models_{rk} \wedge\Sigma \twoheadrightarrow \psi.$$

If $\mathcal{I}(\Delta) = [\![\Delta]\!]_{rk}$, $\mid\!\sim_\Delta^{\mathcal{I}}$ is, modulo cosmetic details, equivalent to preferential entailment (System P) [KLM 90]. If $\leq_{pt}$ describes the pointwise comparison of ranking measures, i.e. $R \leq_{pt} R'$ iff for all $A \in \mathbb{B}_L$ $R(A) \leq R'(A)$, then $\mathcal{I}(\Delta) = \{Min_{\leq_{pt}}[\![\Delta]\!]_{rk}\}$ essentially characterizes System Z [Pea 90]. Because these approaches fail to adequately deal with inheritance to exceptional subclasses, we introduced and developed the construction paradigm for default reasoning [Wey 96, 98, 03], which is a powerful strategy for specifying reasonable $\mathcal{I}$ based on Spohn's Jeffrey-conditionalization for ranking measures. The resulting default inference notions are well-behaved and show nice inheritance features. The essential idea is that defaults do not only specify ranking constraints, but also admissible construction steps to generate them. In particular, for each default $\varphi \rightsquigarrow \psi$, we are allowed to uniformly shift upwards (make less plausible/increase the ranks of) the $\varphi \wedge \neg\psi$-worlds, which amounts to strengthen belief in the corresponding material implication $\varphi \rightarrow \psi$. If W is finite, this is analogous to specifying the rank of a world by adding a weight ≥ 0 for each default it violates. More formally, we define a shifting transformation $R \rightarrow R + r[\rho]$ such that for each ranking measure R, $\chi, \rho \in L$, and $r \in [0, \infty]$, we set

$$(R + r[\rho])(\chi) = min\{R(\chi \wedge \rho) + r, R(\chi \wedge \neg\rho)\}.$$

This corresponds to uniformly shifting ρ by r.

Definition 2.2 (Constructibility)
Let $\Delta = \{\varphi_i \rightsquigarrow / \twoheadrightarrow \psi_i \mid i \leq n\} \subseteq L(\twoheadrightarrow, \rightsquigarrow)$. *A ranking measure* R' *is said to be constructible from* R *over* Δ, *written* $R' \in Constr(\Delta, R)$, *iff there are* $r_i \in [0, \infty]$ *s.t.* $R' = R + \Sigma_{i \leq n} r_i[\varphi_i \wedge \neg\psi_i]$.[3]

For instance, we obtain a well-behaved robust default inference relation, System J [Wey 96], just by setting $\mathcal{I}_J(\Delta) = Constr(\Delta, R_0) \cap [\![\Delta]\!]_{rk}$. To implement shifting minimization, we may strengthen System J by allowing proper shifting ($r_i > 0$) only if the targeted ranking constraint interpreting a default $\varphi_i \rightsquigarrow \psi_i$ is realized as an equality constraint $R(\varphi_i \wedge \psi_i) + 1 = R(\varphi_i \wedge \neg\psi_i)$. Otherwise, the shifting wouldn't seem to be justified in the first place.

Definition 2.3 (Justifiable constructibility)
R *is called a justifiably constructible model of* Δ, *written* $R \in \mathcal{I}_{jj}(\Delta)$ *iff* $R \models_{rk} \Delta$, $R = R_0 + \Sigma_{i \leq n} r_i[\varphi_i \wedge \neg\psi_i]$, *and for each* $r_j > 0$, $R(\varphi_j \wedge \psi_j) + 1 = R(\varphi_j \wedge \neg\psi_j)$.

It follows from a standard property of entropy maximization (ME) that the order-of-magnitude translation of ME, in the context of a nonstandard model of the reals with infinitesimals [GMP 93, Wey 95], to the ranking level always produces a canonical justifiably constructible ranking model R_{me}. We set $\mathcal{I}_{me}(\Delta) = \{R^{\Delta}_{me}\}$. Hence, if $\Delta \not\vdash_{rk} \mathbf{F}$, $R^{\Delta}_{me} \in \mathcal{I}_{jj}(\Delta) \neq \emptyset$. If $\mathcal{I}_{jj}(\Delta)$ is a singleton, we have therefore $\mid\!\sim^{jj} = \mid\!\sim^{me}$. This holds for instance for minimal core default sets Δ [GMP 93], where no doxastically possible $\varphi_i \wedge \neg\psi_i$, i.e. $\Delta \not\vdash_{rk} \varphi_i \wedge \neg\psi_i \rightsquigarrow \mathbf{F}$, is covered by other $\varphi_j \wedge \neg\psi_j$. However, because of its fine-grained quantitative character, $\mid\!\sim^{me}$ is actually representation-dependent, i.e. the solution depends on how we describe a problem in L, it is not invariant under boolean automorphisms of $\mathcal{B}_L$. Fortunately, there are two other natural representation-independent ways to pick up a canonical justifiably constructible model.

- System JZ is based on on a natural canonical hierarchical ranking construction in the tradition of System Z and ensures justifiable constructibility [Wey 98, 03]. It constitutes a uniform way to implement the minimal information philosophy at the ranking level.
- System JJR is based on the fusion of the justifiably constructible ranking models of Δ, i.e. $\mathcal{I}_{jjr}(\Delta) = \{R^{\Delta}_{jjr}\}$, where for all $A \in \mathbb{B}_L$, $R^{\Delta}_{jjr}(A) = Min_{\leq_{pt}} \mathcal{I}_{jj}(\Delta)$. $\mid\!\sim^{jjr}$ may be of particular interest because its canonical ranking model is at least as plausible as every justifiably constructible one.

Note that for non-canonical $\mathcal{I}_{jj}(\Delta)$, it is possible that $R^{\Delta}_{jjr} \notin \mathcal{I}_{jj}(\Delta)$. We have $\mid\!\sim^{jj} \subset \mid\!\sim^{me}, \mid\!\sim^{jz}, \mid\!\sim^{jjr}$. Fortunately, for the generic default sets we will use to interpret abstract argumentation frameworks, all four turn out to be equivalent. To conclude this section, let us consider a simple example with a single JJ-model.

Big birds example:
Non-flying birds are not inferred to be small.

[3] Similar ideas can be found in [BSS 00, KI 01].

$\{B, \neg F\} \cup \{B \rightsquigarrow S, B \rightsquigarrow F, \neg S \rightsquigarrow \neg F\} \not\mid\!\sim S$
The canonical JJ/ME/JZ/JJR-model is then
$R = R_0 + 1[\neg F] + 2[\neg S \wedge F]$. But $R \not\models_{rk} B \wedge \neg F \rightsquigarrow S$ because $R(B \wedge \neg F \wedge S) = R(B \wedge \neg F \wedge \neg S) = 1$

3 Abstract argumentation

The idea of abstract argumentation theory, launched by Dung [Dun 95], has been to replace the traditional bottom-up strategy, which models and exploits the logical fine structure of arguments, by a top-down perspective, where arguments become black boxes evaluated only based on knowledge about specific logical or extra-logical relationships connecting them. It is interesting to see that such a coarse-grained relational analysis often seems sufficient to determine which collections of instantiated arguments are reasonable. In addition to possible conceptual and computational gains, the abstract viewpoint offers furthermore a powerful methodological tool for general argumentation-theoretic investigations.

An abstract argumentation framework in the original sense of Dung is a structure of the form $\mathcal{A} = (\mathbb{A}, \rhd)$, where $\mathbb{A}$ is a collection of abstract entities representing arguments, and $\rhd$ is a possibly asymmetric binary attack relation modeling conflicts between arguments. To grasp the expressive complexity of real-world argumentation, several authors have extended this basic account to include further inferential or cognitive relations, like support links, preferences, valuations, or collective attacks. Our general definition[4] [Wey 11] for the first-order context is as follows.

Definition 3.1 (Hyperframeworks) *A general abstract argumentation framework, or hyperframework (HF), is just a structure of the form* $\mathcal{A} = (\mathbb{A}, (\mathcal{R}_i)_{i \in I}, (\mathcal{P}_j)_{j \in J})$, *where* $\mathbb{A}$ *is the domain of arguments, the* $\mathcal{R}_i$ *are conflictual, and the* $\mathcal{P}_i$ *non-conflictual relations over* $\mathbb{A}$. $B \subseteq \mathbb{A}$ *is said to be conflict-free iff it does not instantiate a conflictual relation.*

For instance, standard Dung frameworks $(\mathbb{A}, \rhd)$ carry one conflictual and no non-conflictual relations. The general inferential task in abstract argumentation is to identify reasonable evaluations of the arguments described by $\mathcal{A}$, e.g. to find out which sets of arguments describe acceptable argumentative positions. These are called extensions. In Dung's scenario, the extensions are $E \subseteq \mathbb{A}$ obeying suitable acceptability conditions in the context of $\mathcal{A}$, the minimal requirement being the absence of internal conflicts. For instance, E is admissible iff it is conflict-free and each attacker of an $a \in E$ is attacked by some $b \in E$. E is grounded/preferred iff it is minimally/maximally admissible, it is stage iff $E \cup \rhd''E$ is maximal, semi-stable if it is also admissible, and stable iff $\mathbb{A} - E = \rhd''E$. Here $\rhd''E$ is the relational image of E, i.e. the set of $a \in \mathbb{A}$ attacked by some $b \in E$. In concrete decision contexts, we may however also want to exploit finer-grained assessments of arguments, like prioritizations or classifications. This suggests a more general semantic perspective [Wey 11].

Definition 3.2 (Hyperextensions) *A hyperframework semantics is a map* $\mathcal{E}$ *associating with each hyperframework*

[4] A bit of an overkill for this paper, but we couldn't resist.

$\mathcal{A} = (\mathbb{A}, (\mathcal{R}_i)_{i\in I}, (\mathcal{P}_j)_{j\in J})$ of a given signature a collection $\mathcal{E}(\mathcal{A})$ of distinguished evaluation structures expanding $\mathcal{A}$, of the form $(\mathcal{A}, In^{\mathcal{A}}, (\mathcal{F}_h)_{h\in H})$. $In^{\mathcal{A}}$ is here a conflict-free subset of $\mathbb{A}$. The elements of $\mathcal{E}(\mathcal{A})$ are called hyperextensions of $\mathcal{A}$.

$In^{\mathcal{A}}$ plays here the role of a classical extension, whereas the $\mathcal{F}_h$ ($h \in H$) express more sophisticated structures over arguments, e.g. a posteriori plausibility orderings, value predicates, or completions of framework relations considered partial. If $H = \emptyset$, we are back to Dung.

4 Concretizing arguments

Ideally, abstract argumentation frameworks should be reconstructible as actual abstractions of logic-based argumentation scenarios. Such an anchoring seems required to develop, evaluate, and apply the abstract models in a suitable way. In a first step, this amounts to instantiate the abstract arguments from the framework domain by logical entities representing concrete arguments, and to interpret the abstract framework structure by specific inferential or evaluational relationships fitting the conceptual intentions the abstract level tries to capture. In what follows we will sketch a natural hierarchy of instantiation layers, passing from more concrete, deep instantiations, to more abstract, shallow ones, with a focus on the intermediate level.

Structured instantiations:
We start with logic-based structured argumentation over a defeasible conditional logic $\mathcal{L}_\delta = (L \cup L(\twoheadrightarrow, \rightsquigarrow), \vdash^\delta, \mid\!\sim^\delta)$, with $(L, \vdash)$ as a classical Tarskian background logic. For the moment, we do not impose any further a priori conditions on $\mathcal{L}_\delta$. But eventually we will turn to specific ranking-based default formalisms. In the context of $\mathcal{L}_\delta$, a concrete defeasible argument a for a claim $\psi_a \in L$, exploiting some given general knowledge base $\Sigma \cup \Delta$, is modeled by a finite rooted defeasible inference tree $\mathcal{T}_a$ whose nodes s are tagged by local claims $\eta_s \in L \cup L(\twoheadrightarrow, \rightsquigarrow)$ such that

- the root node is tagged by ψ_a,
- the leaf nodes are tagged by $\eta_s \in \Sigma_a \cup \Delta_a \cup \Lambda$, where $\Lambda = \{\mathbf{T}\} \cup \{\varphi \twoheadrightarrow \varphi, \varphi \rightsquigarrow \varphi \mid \varphi \in L\}$ (basic tautologies),
- the non-leaf nodes are tagged by $\eta_s \in L$ s.t. $\Gamma_s \mid\!\sim^\delta \eta_s$ where Γ_s is the set of claims from the children of s.

$\Sigma_a \cup \Delta_a$ is the contingent premise set of a, the premises being the claims of the leaf nodes. Within concrete arguments, the local justification steps, e.g. from Γ_s to η_s, are typically assumed to be elementary, like instances of modus ponens. To handle reasoning by cases, which holds for plausible implication, we may also apply the *disjunctive modus ponens* for $\twoheadrightarrow$ and $\rightsquigarrow$, e.g.

$$\Gamma_s = \{\varphi_1 \vee \ldots \vee \varphi_n, \varphi_1 \rightsquigarrow \psi_1, \ldots, \varphi_n \twoheadrightarrow \psi_n\}$$

$$\mid\!\sim^\delta \psi_1 \vee \ldots \vee \psi_n (= \eta_s).$$

If $\Gamma_s \subseteq L(\twoheadrightarrow)$, we can replace $\mid\!\sim^\delta$ by $\vdash^\delta$ and obtain a strict inference step. For our purposes we may ignore the exact nature of the justification steps. Note that the correctness of local inference steps does not entail the global correctness of the argument a. Consider for instance $\Sigma_a \cup \Delta_a = \{\varphi\} \cup \{\varphi \rightsquigarrow \psi, \psi \rightsquigarrow \neg\varphi\}$, which is consistent w.r.t. $\mid\!\sim^\delta = \mid\!\sim^{\mathcal{I}}$.

$$\{\varphi\} \cup \{\varphi \rightsquigarrow \psi\} \mid\!\sim^\delta \psi \text{ and } \{\psi\} \cup \{\psi \rightsquigarrow \neg\varphi\} \mid\!\sim^\delta \neg\varphi,$$

but $\Sigma_a \cup \Delta_a \not\mid\!\sim^\delta \neg\varphi$. This example looks odd because accepting the whole argument would require the acceptance of all its claims, which is blocked by $\varphi, \neg\varphi \vdash \mathbf{F}$. In fact, a natural requirement for an acceptable argument a would be that it satisfies

Material consistency: $\Sigma_a \cup \Delta_a^{\rightarrow} \not\vdash \mathbf{F}$.

This means that the factual premises and the material implications corresponding to the conditional premises are classically consistent. Note that this condition is strictly stronger than $\Sigma_a \cup \Delta_a \not\mid\!\sim^\delta \mathbf{F}$ because we typically have $\{\mathbf{T} \rightsquigarrow \varphi, \neg\varphi\} \not\mid\!\sim^\delta \mathbf{F}$ whereas $\{\mathbf{T} \rightarrow \varphi, \neg\varphi\} \vdash \mathbf{F}$. However, in practice, without omniscience w.r.t. propositional logic, it may not be clear whether these global conditions are actually satisfied. Real arguments may well be inconsistent in the strong sense.

In structured argumentation, an argument tree has two functions: first, to describe and offer a prima facie justification for a claim, and secondly, to specify target points where other arguments may attack. It is essentially a computational tool which is intended to help identifying - or even defining - inferential relationships within a suitable defeasible conditional logic $\mathcal{L}_\delta$, and to help specifying attack relations to determine reasonable argumentative positions.

But what can we say about the semantic content of an argument represented by such a tree? What is an agent committed to if he accepts or believes a given argument, or a whole collection of arguments? Which tree attributes have to be known to specify this content? What is the meaning of attacks between arguments?

Conditional instantiations:
Our basic idea is that, whatever the requirements for argumentation trees in the context of $\mathcal{L}_\delta$, and whatever the content of an argument a represented by such a tree $\mathcal{T}_a$, it should only depend on the collection of local claims $\{\eta_s \mid s \text{ node of } \mathcal{T}_a\}$, and more specifically, on the choice of the main claim ψ_a, the premise claims $\Sigma_a \cup \Delta_a$, and the intermediate claims Ψ_a. In fact, because the acceptance of a structured argument includes the acceptance of all its subarguments, we have to consider the main claims of the subarguments as well. So we can assume that the content of $\mathcal{T}_a$ is fixed by the triple $(\Sigma_a \cup \Delta_a, \Psi_a, \psi_a)$. An agent accepting a obviously has to be committed to all the elements of the base $\Sigma_a \cup \Delta_a \cup \Psi_a \cup \{\psi_a\}$.

To be fully acceptable w.r.t. $\mathcal{L}_\delta$, the structured argument also has to be globally correct in the sense that all its local claims are actually defeasibly entailed by $\Sigma_a \cup \Delta_a$. In particular, $\Sigma_a \cup \Delta_a \mid\!\sim^\delta \psi$ for each $\psi \in \Psi_a \cup \{\psi_a\}$. This requirement should also hold for each subarguments b of a. But note that, because of defeasibility, this does not exclude that the premises $\Sigma_b \cup \Delta_b$ of a subargument b could implicitly infer the negation of a local claim ψ_x external to b, as long as this conflicting inference is eventually overridden by the full premise set $\Sigma_a \cup \Delta_a$. It follows that the strengthening of a subargument by choosing a stronger claim could

undermine global correctness. But if the intermediate claims are always inferred and therefore implicitly present, we may actually drop Ψ_a and just consider for each globally correct argument a the finite inference pair $(\Sigma_a \cup \Delta_a, \psi_a)$.

Given a pure Dung framework $\mathcal{A} = (\mathbb{A}, \rhd)$ and the defeasible conditional logic $\mathcal{L}_\delta$, a structured instantiation I_{str} of $\mathcal{A}$ maps each $a \in \mathbb{A}$ to a globally correct argument tree $\mathcal{T}_a$ over $\mathcal{L}_\delta$. On the most general level, we do not want to impose a priori further restrictions beyond inferential correctness. In practice one may however well decide to focus on specific argument trees, e.g. those using specific justification steps. Each $I_{str}(a)$ specifies a correct inference pair $I_{log}(a) = (\Sigma_a \cup \Delta_a, \psi_a)$, which we call a conditional logical instantiation of a over $\mathcal{L}_\delta$. I_{log} specifies the intended logical content of an argument on the syntactic level. Note that it depends on the tree concept whether we can obtain all the correct inference pairs.

In monotonic argumentation, the consistency and minimality of the premise sets are standard assumptions. But within defeasible argumentation, a more liberal perspective may be preferable. For instance, on the structured level, we want to allow arguments claiming $\mathbf{F}$. The reductio ad absurdum principle then offers a possibility to attack arguments from within. Consequently, we also have to accept instantiating inference pairs whose conclusion is $\mathbf{F}$. On the other hand, material consistency, the existence of models of Σ_a which do not violate any conditional in Δ_a, is a natural requirement in the context of argumentation theory. But we can replace it by a qualified version, restricted to those instances where $\Sigma_a \cup \Delta_a$ is actually consistent.

What about minimality? First, it may obviously fail for inference pairs obtained by flattening argument trees. Of course, we could consider an additional minimization step where we replace each $(\Sigma_a \cup \Delta_a, \psi_a)$ by all those (Φ, ψ_a) with $\Phi \subseteq \Sigma_a \cup \Delta_a$ and which are minimal s.t. $\Phi \mid\!\sim^\delta \psi_a$. Although this may be computationally costly, it could be theoretically appealing. However, minimality could also be questioned because by adding premises, a conclusion may successively get accepted, rejected, and accepted again, letting the character of the inferential support change between different levels of specifity, which calls for a discrimination between the corresponding inference pairs. Proponents of minimality object that these types of support could, perhaps, also be reproduced by suitable minimal (Φ, ψ_a). However, this assumption is not sustainable for ranking-based semantics for argumentation, because here the results may change if we restrict ourselves to minimal premise sets. In fact, shrinking $\Sigma_a \cup \Delta_a$ to Φ may actually increase the set of possible attacks. In particular, we could have attacks on all the minimal $\Phi \mid\!\sim^\delta \psi_a$, but none on $\Sigma_a \cup \Delta_a \mid\!\sim^\delta \psi_a$. Hence premise minimality may fail.

Shallow instantiations:
Let us recall our task: exploiting a ranking semantics for default reasoning to provide a plausibilistic semantics for abstract argumentation. But inference pairs, which populate the conditional logical instantiation level, are still rather complex and opaque objects. To model argumentation frameworks and their semantics, we would here have to deal with sets of sets of conditionals, whose inferential interactions may furthermore be hard to assess. We therefore prefer to start with simpler entities and to seek more abstraction.

Consider the main goal of an agent: to extract from argument configurations suitable beliefs, expressed in the domain language L, whose plausibility is semantically modeled by ranking measures over $\mathbb{B}_L$. Given an inference pair $(\Sigma_a \cup \Delta_a, \psi_a)$ representing the full conditional logical content of an argument a, in addition to the main claim ψ_a, there are three relevant collections of formulas: $\Sigma_a, C_\vdash(\Delta_a \cup, \Sigma_a), C_{\mid\sim}(\Delta_a \cup \Sigma_a)$ which represent resp. the premises, the strict, and the defeasible consequences. If the language is finitary, this gives us four L-formulas representing the relevant propositional L-content.

- $\varphi_a = \wedge\Sigma_a$ (premise content).
- $\theta_a = \wedge C_\vdash(\Delta_a \cup \Sigma_a)$ (strict content).
- $\delta_a = \wedge C_{\mid\sim}(\Delta_a \cup \Sigma_a)$ (defeasible content).
- ψ_a (main claim).

We have $\delta_a \vdash \theta_a \vdash \varphi_a$, and $\delta_a \vdash \psi_a$ by inferential correctness. δ_a specifies the strongest possible claim based on the information made available by the argument. For our semantic modeling purposes, we will assume that $\psi_a = \delta_a$. If we abstract away from the representational details, we arrive at our central concept: the *shallow semantic instantiation* of a extracted from the conditional logical instantiation $I_{log}(a)$.

$$I_{sem}(a) = (\llbracket\varphi_a\rrbracket, \llbracket\theta_a\rrbracket, \llbracket\psi_a\rrbracket).$$

In the following, we will sloppily denote $I_{sem}(a)$ by $(\varphi_a, \theta_a, \psi_a)$. One should emphasize that these propositional semantic profiles are not intended to grasp the full nature of arguments, but only to reflect certain characteristics exploitable by suitable argumentation semantics. We observe that each proposition triple (φ, θ, ψ) with $\psi \vdash \theta \vdash \varphi$ can become a shallow instantiation. In fact, if $I_{log}(a) = (\{\varphi, \varphi \twoheadrightarrow \theta, \varphi \rightsquigarrow \psi\}, \psi)$, for standard $\mid\!\sim^\delta$, we obtain $I_{sem}(a) = (\varphi, \theta, \psi)$. In terms of ranking constraints, this gives us $R(\varphi \wedge \neg\theta) = \infty$ and $R(\varphi \wedge \psi) + 1 \leq R(\varphi \wedge \neg\psi)$.

5 Concretizing attacks

One argument attacks another argument if accepting the first interferes with the inferential structure or the goal of the second one. To avoid a counterattack, the premises of the attacked argument should also not affect the inferential success of the attacker, otherwise the presupposition of the attack could be undermined. In the following we will investigate attack relations between conditional logical resp. shallow semantic instantiations of abstract arguments. We start with the former. Let $I_{log}(a) = (\Sigma_a \cup \Delta_a, \psi_a)$ and $I_{log}(b) = (\Sigma_b \cup \Delta_b, \psi_b)$ be two correct inference pairs for $\mathcal{L}_\delta$. We distinguish two scenarios: unilateral and mutual attack. The idea is to say that $(\Sigma_a \cup \Delta_a, \psi_a)$ unilaterally attacks $(\Sigma_b \cup \Delta_b, \psi_b)$ iff the premises of both arguments together with ψ_a enforce the strict rejection of ψ_b, i.e.

$$\Sigma_b \cup \Delta_b \cup \Sigma_a \cup \Delta_a \cup \{\psi_a\} \vdash^\delta \neg\psi_b,$$

whereas the defeasible inference of ψ_a from the premises is preserved, i.e.

$$\Sigma_a \cup \Delta_a \cup \Sigma_b \cup \Delta_b \mid\!\sim^{\delta} \psi_a.$$

On the other hand, $(\Sigma_a \cup \Delta_a, \psi_a)$ and $(\Sigma_b \cup \Delta_b, \psi_b)$ attack each other iff they strictly reject each other's claims, i.e.

$$\Sigma_b \cup \Delta_b \cup \Sigma_a \cup \Delta_a \cup \{\psi_a\} \vdash^{\delta} \neg\psi_b, \text{ and}$$

$$\Sigma_a \cup \Delta_a \cup \Sigma_b \cup \Delta_b \cup \{\psi_b\} \vdash^{\delta} \neg\psi_a.$$

This holds for instance if their premise sets, resp. their claims, are classically inconsistent. This definition provides one of the strongest possible natural attack relations for inference pairs. Note that we have a self-attack iff the premise set is inconsistent, i.e. $\Sigma_a \cup \Delta_a \vdash^{\delta} \mathbf{F}$. To exploit the powerful semantics of ranking-based default reasoning, in what follows we will assume that $\mid\!\sim^{\delta} = \mid\!\sim^{\mathcal{I}}$, where $\mathcal{I}$ is a ranking choice function.

How can we exploit the above approach to define attacks between shallow instantiations, e.g. $I_{sem}(a) = (\varphi_a, \theta_a, \psi_a)$ and $I_{sem}(b) = (\varphi_b, \theta_b, \psi_b)$? The corresponding inference pairs are $I_{log}(a) = (\{\varphi_a, \varphi_a \twoheadrightarrow \theta_a, \varphi_a \rightsquigarrow \psi_a\}, \psi_a)$ and $I_{log}(b) = (\{\varphi_b, \varphi_b \twoheadrightarrow \theta_b, \varphi_b \rightsquigarrow \psi_b\}, \psi_b)$. For an unilateral attack from $I_{log}(a)$ on $I_{log}(b)$, we must have

$$\mathcal{I}(\Delta_a \cup \Delta_b) \models_{rk} \varphi_a \wedge \varphi_b \wedge \psi_a \twoheadrightarrow \neg\psi_b, \text{ and}$$

$$\mathcal{I}(\Delta_a \cup \Delta_b) \models_{rk} \varphi_a \wedge \varphi_b \rightsquigarrow \psi_a.$$

This is, for instance, automatically realized if $\psi_a \vdash \neg\psi_b$, $\varphi_a \vdash \varphi_b$, and we have logical independence elsewhere. For a bilateral attack, we may just drop the condition $\varphi_a \vdash \varphi_b$. However, we do not have to presuppose that all the attacks result from the logical structure induced by the instantiation. In fact, in addition to the instantiation-intrinsic attack relationships, there could be further attack links derived from a separate conditional knowledge base reflecting other known attacks.

From a given Dung framework $\mathcal{A} = (\mathbb{A}, \rhd)$ and a shallow instantiation $I = I_{sem}$, if we adopt the ranking semantic perspective and the above attack philosophy, we can induce a collection of conditionals specifying ranking constraints. For any $a \in \mathbb{A}$, the shallow inference pair supplies $\varphi_a \twoheadrightarrow \theta_a$ (alternatively, $\varphi_a \wedge \neg\theta_a \rightsquigarrow \mathbf{F}$) and $\varphi_a \rightsquigarrow \psi_a$. For every attack $a \rhd b$, we get at least $\psi_a \wedge \psi_b \rightsquigarrow \mathbf{F}$. Note that this is a consequence of choosing maximal claims at the instantiation level. For each unilateral attack $a \rhd b$ we must add $\varphi_a \wedge \varphi_b \rightsquigarrow \psi_a$ to preserve the inferential impact of a in the context of b. The resulting default base is

$$\begin{aligned}\Delta^{\mathcal{A},I} = &\{\varphi_a \rightsquigarrow \psi_a, \varphi_a \twoheadrightarrow \theta_a \mid a \in \mathbb{A}\}\\ &\cup \{\psi_a \wedge \psi_b \rightsquigarrow \mathbf{F} \mid a \rhd b \text{ or } b \rhd a\}\\ &\cup \{\varphi_a \wedge \varphi_b \rightsquigarrow \psi_a \mid a \rhd b, b \not\rhd a\}.\end{aligned}$$

We observe that for each 1-loop, we get $\psi_a \rightsquigarrow \mathbf{F}$ and $\varphi_a \rightsquigarrow \psi_a$, hence $\Delta^{\mathcal{A},I} \vdash^{\mathcal{I}} \neg\varphi_a$. The doxastic impossibility of φ_a illustrates the paradoxical character of self-attacking arguments. The belief states compatible with an instantiated framework $\mathcal{A}, I$ are here represented by the ranking models of $\Delta^{\mathcal{A},I}$.

Conversely, we can identify for each instantiation I of $\mathbb{A}$ and each collection of ranking measures $\mathcal{R} \models_{rk} \Delta^{\mathbb{A},I} = \{\varphi_a \rightsquigarrow \psi_a, \varphi_a \twoheadrightarrow \theta_a \mid a \in \mathbb{A}\}$ the attacks supported by all the $R \in \mathcal{R}$. Let $\rhd_I^{\mathcal{R}}$ be the resulting attack relation, that is, for each $a, b \in \mathbb{A}$

$$a \rhd_I^{\mathcal{R}} b \text{ iff for all } R \in \mathcal{R}, R \models_{rk} \psi_a \wedge \psi_b \rightsquigarrow \mathbf{F} \text{ and}$$

$$(R \models_{rk} \varphi_a \wedge \varphi_b \rightsquigarrow \psi_a \text{ or } R \not\models_{rk} \varphi_a \wedge \varphi_b \rightsquigarrow \psi_b).$$

The second disjunct is the result of an easy simplification. If a or b are self-reflective, we have $a \rhd_I^{\mathcal{R}} b$ because conditionals always hold if the premises are doxastically impossible. Because in this paper we will mainly consider canonical ranking choice functions, we are going to focus on $\mathcal{R} = \{R\}$, setting $\rhd_I^R = \rhd_I^{\mathcal{R}}$.

Definition 5.1 (Ranking instantiation models) *Let the notation be as usual and $\mathbb{A}^+ = \{a \in \mathbb{A} \mid a \not\rhd a\}$. (R, I) is called a ranking instantiation model (more sloppily, a ranking model) of $\mathcal{A}$ iff*

$$R \models_{rk} \Delta^{\mathbb{A},I} = \{\varphi_a \rightsquigarrow \psi_a, \varphi_a \twoheadrightarrow \theta_a \mid a \in \mathbb{A}\},$$

and for all $a, b \in \mathbb{A}^+$, $a \rhd b$ iff $a \rhd_I^R b$. Let $\mathcal{R}^{\mathcal{A}}$ be the collection of all the ranking instantiation models of $\mathcal{A}$.

That is, over the non-loopy arguments, the semantic-based attack relation $\rhd_I^R$ specified by R, I has to correspond exactly to the abstract attack relation $\rhd$. The collection of ranking instantiation models is not meant to change if we add or drop attack links between self-reflective and other arguments because the details are absorbed by the impossible joint contexts. If $\mathcal{A}$ and $\mathcal{A}'$ share the same 1-loops and the same attack structure over the other arguments, $\mathcal{R}^{\mathcal{A}} = \mathcal{R}^{\mathcal{A}'}$.

It also important to observe, and we will come back to this, that each $\mathcal{A} = (\mathbb{A}, \rhd)$ admits many ranking instantiation models (R, I), obtained by varying the ranking values or the proposition triples associated with the abstract arguments.

What can we say about classical types of attack? If we focus on the actual semantic content, rebuttal is characterized by incompatible defeasible consequents, and undermining by a defeasible consequent conflicting with an antecedent. In the ranking context, these two types of attacks can be modeled by constraints expressing necessities. The ranking characterizations are as follows. Recall that $\psi_a \vdash \varphi_a$, $\psi_b \vdash \varphi_b$.

a **rebuts** b: $R(\psi_a \wedge \psi_b) = \infty$, e.g. if $\psi_a \vdash \neg\psi_b$.
a **undermines** b: $R(\psi_a \wedge \varphi_b) = \infty$, e.g. if $\psi_a \vdash \neg\varphi_b$.

In our simple semantic reading, undermining entails rebuttal because $\psi_b \vdash \varphi_b$. There are four qualitative attack configurations involving two arguments: $\varphi_a \wedge \varphi_b$ being compatible with neither, one, or both of ψ_a, ψ_b. If a asymmetrically undermines b, we have $R(\psi_a \wedge \varphi_b) = \infty$ and $R(\psi_b \wedge \varphi_a) \neq \infty$, hence $R(\varphi_a \wedge \varphi_b) \neq \infty$. This implies $R \models_{rk} \varphi_a \wedge \varphi_b \wedge \psi_b \twoheadrightarrow \neg\psi_a$ and $R \not\models_{rk} \varphi_a \wedge \varphi_b \rightsquigarrow \psi_a$, i.e. $b \rhd_I^R a$ and $a \not\rhd_I^R b$ according to our attack semantics. It follows that undermining has no obvious ranking semantic justification if we stipulate that the defeasible claim entails the antecedent. Also note that rebuttal is entailed by symmetric and asymmetric attacks.

6 Ranking extensions

Ranking instantiation models offer new possibilities to identify reasonable argumentative positions. Let (R, I) be a model of the framework $\mathcal{A} = (\mathbb{A}, \rhd)$. In the context of (R, I), a minimal requirement for aceptable argument sets

$S \subseteq \mathbb{A}$ are coherent premises, i.e. the doxastic possibility of the joint strict contents $\varphi_S = \wedge_{a\in S}(\varphi_a \wedge \theta_a)$ w.r.t. R, which means $R(\varphi_S) \neq \infty$. This excludes self-attacks, but not conflicts within S. $S = \emptyset$ is by definition coherent because $\varphi_\emptyset = \mathbf{T}$. Given that evidence φ_a should not be rejected without good reasons, the maximally coherent $S \subseteq \mathbb{A}$ are of particular interest and constitute suitable background contexts when looking for extensions. Each $E \subseteq S$ then specifies a proposition

$$\psi_{S,E} := \varphi_S \wedge \wedge_{a\in E}\psi_a \wedge \wedge_{a\in\mathbb{A}-E}\neg\psi_a.$$

$\psi_{S,E}$ characterizes those worlds verifying the strict content of the $a \in S$ and exactly the defeasible content of the $a \in E$. Because $a \rhd^R_I b$ implies $R(\psi_a \wedge \psi_b) = \infty$, any conflict $a \rhd b$ in E makes $\psi_{S,E}$ impossible. Note however that $R(\psi_{S,E}) = \infty$ may also result from non-binary conflicts involving multiple arguments, or a specific choice of logically dependent φ_a, ψ_a. What are the most reasonable extension candidates $E \subseteq S \subseteq \mathbb{A}$ according to (R, I)? One idea is to focus on those E which induce the most plausible $\psi_{S,E}$ relative to φ_S among all their maximal coherent supersets S.

Definition 6.1 (Ranking extensions) *Let (R, I) be a ranking instantiation model of $\mathcal{A} = (\mathbb{A}, \rhd)$. $E \subseteq \mathbb{A}$ is called a ranking-extension of $\mathcal{A}$ w.r.t. (R, I) iff there is a maximal coherent $S \subseteq \mathbb{A}$ with $E \subseteq S$ and $R(\psi_{S,E}|\varphi_S) = 0$.*

Observe that if $S = \emptyset$ is the maximally coherent subset of $\mathbb{A}$, then $R(\varphi_a) = \infty$ for each $a \in \mathbb{A}$ and $E = \emptyset$ is the only ranking extension. While the above definition looks rather decent, a cause of concern may be the great diversity of ranking models (R, I) available for any given $\mathcal{A}$. Consider for instance $\mathcal{A} = (\{p, q, r\}, \{(p, q), (q, r)\})$, i.e. $p \rhd q \rhd r$. $\mathcal{A}$ together with a shallow instantiation I then induces ranking constraints described by the conditionals in

$$\Delta^{\mathcal{A},I} = \{\psi_p \wedge \psi_q \rightsquigarrow \mathbf{F}, \psi_q \wedge \psi_r \rightsquigarrow \mathbf{F}, \varphi_p \wedge \varphi_q \rightsquigarrow \psi_p, \varphi_q \wedge \varphi_r \rightsquigarrow \psi_q, \varphi_p \rightsquigarrow \psi_p, \varphi_q \rightsquigarrow \psi_q, \varphi_r \rightsquigarrow \psi_r\}.$$

If we assume that each set $\{\varphi_x, \psi_x\}$ is logically independent from all the other $\{\varphi_y, \psi_y\}$, then $\Delta^{\mathcal{A},I}$ admits a unique justifiably constructible model, which therefore automatically must be the JZ- and JJR-model: $R^{\mathcal{A},I}_{jz}$. In this example it is obtained by minimally shifting the violation areas of the conditionals.

$$R^{\mathcal{A},I}_{jz} = R_0 + \infty[\psi_p \wedge \psi_q] + \infty[\psi_q \wedge \psi_r] + 1[\varphi_p \wedge \varphi_q \wedge \neg\psi_p] + 1[\varphi_q \wedge \varphi_r \wedge \neg\psi_q] + 1[\varphi_p \wedge \neg\psi_p] + 1[\varphi_q \wedge \neg\psi_q] + 1[\varphi_r \wedge \neg\psi_r].$$

Given that $S = \mathbb{A}$ is coherent, there are eight extension candidates. For the doxastically possible alternatives, $R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\{p,r\}}) = 2 < 3 = R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\{p\}}) = R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\{q\}}) < 4 = R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\{r\}}) < 5 = R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\emptyset}) < \infty$. Because $R^{\mathcal{A},I}_{jz}(\varphi_S) = 2$, we get $R^{\mathcal{A},I}_{jz}(\psi_{\mathbb{A},\{p,r\}}|\varphi_S) = 0$. The unique ranking extension is therefore $\{p, r\}$, which is also the standard Dung solution.

However, without any further constraints on the extension generating ranking instantiation model (R, I), we could pick up as an alternative $R = R^{\mathcal{A},I}_{jz} + \infty[\psi_p \wedge \psi_r \wedge \varphi_q]$ such that $R(\psi_p \wedge \psi_r \wedge \varphi_q) = \infty$, resp. an I enforcing $\psi_p \wedge \psi_r \wedge \varphi_q \vdash F$. In both scenarios, the minima would then become $R(\psi_{\mathbb{A},\{p\}}) = R(\psi_{\mathbb{A},\{q\}}) = 3$, imposing the ranking extensions $\{p\}, \{q\}$. Because of $R(\psi_{\mathbb{A},\{p,r\}}) = \infty$, the standard extension $\{p, r\}$ would necessarily be rejected. But this violates a hallmark of argumentation, namely the unconditional support of unattacked arguments, like p. This shows that we have to control the choice of ranking instantiation models to implement a reasonable ranking extension semantics.

The idea is now to choose on one hand, as our doxastic background, a well-justified canonical ranking measure model of the default base $\Delta^{\mathcal{A},I}$, e.g. the JZ-model $R^{\mathcal{A},I}_{jz}$, and on the other hand, implementing Ockham's razor, the simplest instantiations of the given framework $\mathcal{A}$. In particular, we stipulate that the instantiations of individual arguments should by default be logically independent. We emphasize that the goal here is just to interpret abstract argumentation frameworks with a minimal amount of additional assumptions, not to adequately model specific real-world arguments.

We can satisfy these desiderata by using disjoint vocabularies for instantiating different abstract arguments, and by relying on elementary instances of the defeasible modus ponens for the corresponding inference pairs. That is, we introduce for each $a \in \mathbb{A}$ independent propositional atoms X_a, Y_a, and set $I_{log}(a) = (\{X_a\} \cup \{X_a \rightsquigarrow Y_a\}, Y_a)$. The corresponding shallow semantic instantiation is then $I(a) = I_{sem}(a) = (\varphi_a, \varphi_a, \psi_a) = (X_a, X_a, X_a \wedge Y_a)$. We call I a *generic instantiation*. Up to boolean isomorphy, it is completely characterized by the cardinality of $\mathbb{A}$.

If we fix a generic instantiation I, then the unique justifiably constructible ranking model of $\Delta^{\mathcal{A},I}$ is ($a \lhd/\rhd b$: $a \rhd b$ or $b \rhd a$)

$$\begin{aligned} R^{\mathcal{A}}_{jz} &= R_0 + \Sigma_{a\not\rhd a} 1[\varphi_a \wedge \neg\psi_a] + \Sigma_{a\rhd a}\infty[\varphi_a \wedge \neg\psi_a] + \\ &\quad \Sigma_{a\rhd b} 1[\varphi_a \wedge \varphi_b \wedge \neg\psi_a] + \Sigma_{a\lhd/\rhd b}\infty[\psi_a \wedge \psi_b] \\ &= R_0 + \Sigma_{a\not\rhd a} 1[X_a \wedge \neg Y_a] + \Sigma_{a\rhd a}\infty[X_a \wedge \neg Y_a] + \\ &\quad \Sigma_{a\rhd b} 1[X_a \wedge X_b \wedge \neg Y_a] + \Sigma_{a\lhd/\rhd b}\infty[X_a \wedge Y_a \wedge X_b \wedge Y_b]. \end{aligned}$$

Because the $\{X_a, Y_a\}$ are logically independent for distinct a, and the defaults expressing an attack $a \rhd b$ just concern $X_a \wedge X_b$, only those X_a with $a \rhd a$ become impossible. In fact, $\{\varphi_a \rightsquigarrow \psi_a, \psi_a \wedge \psi_a \rightsquigarrow \mathbf{F}\} \vdash_{rk} \varphi_a \rightsquigarrow \mathbf{F}$. Hence, in line with intuition, the ranking instantiation model $(R^{\mathcal{A}}_{jz}, I)$ will trivialize exactly the self-defeating arguments. Assuming genericity, $\mathbb{A}^+ = \{a \in \mathbb{A} \mid a \not\rhd a\}$ is then the unique maximal coherent subset of $\mathbb{A}$. We are now ready to specify our ranking-based evaluation semantics. Note that all the generic I are essentially equivalent.

JZ-evaluation semantics (JZ-extensions):
$\mathcal{E}_{jz}(\mathcal{A}) = \{E \subseteq \mathbb{A} \mid E$ ranking extension w.r.t. $(R^{\mathcal{A},I}_{jz}, I)$ for any/all generic $I\}$.

There is actually a simple algorithm to identify the JZ-extensions using extension weights.

Definition 6.2 (Extension weight) *For each argumentation framework $\mathcal{A} = (\mathbb{A}, \rhd)$, the extension weight function $r_{\mathcal{A}} : 2^{\mathbb{A}} \to [0, \infty]$ is defined as follows: If E is conflict-free,*

$r_{\mathcal{A}}(E) = |\mathbb{A}^+ - E| + |\{a \in \mathbb{A}^+ - E \mid \exists b \in \mathbb{A}^+ (a \rhd b \wedge b \not/ \rhd a)\}|$, *if not,* $r_{\mathcal{A}}(E) = \infty$.

It is not too difficult to see that $r_{\mathcal{A}}(E) = R_{jz}^{\mathcal{A},I}(\psi_{\mathbb{A}^+,E})$. Hence, $E \in \mathcal{E}_{jz}(\mathcal{A})$ iff $r_{\mathcal{A}}(E) = min\{r_{\mathcal{A}}(X) \mid X \subseteq \mathbb{A}\}$. That is, the JZ-extensions are those where the sum of the number of non-reflective non-extension arguments and the number of one-sided attacks starting from them is minimal.

7 Examples and properties

To get a better understanding of the ranking extension semantics and its relation with other extension concepts, let us first take a look at how it handles some basic examples. Because of its uncommon semantic perspective and its partly quantitative character, we will observe some unorthodox behaviour. Under instantiation genericity, it is enough to compare $R^{\mathcal{A},I}(\psi_{\mathbb{A}^+,E})$ for $E \subseteq \mathbb{A}^+$, or to focus on 1-loop-free frameworks. For each instance, we specify the domain $\mathbb{A}$ and the full attack relation $\rhd$. $\psi_{\mathbb{A}^+,\{x_1 \ldots x_n\}}$ is abbreviated by $\psi_{x_1,\ldots,x_n}$ resp. $\psi_{\emptyset}$.

Simple reinstatement: $\{a, b, c\}$ with $a \rhd b \rhd c$.

The grounded extension $\{a, c\}$ is the canonical result put forward by any standard acceptability semantics. The unique JJ-model, i.e. the JZ-model R of $\Delta^{\mathcal{A},I}$, satisfies $R(\psi_a) = R(\psi_b) = 3, R(\psi_c) = 4, R(\psi_{a,c}) = 2$, and $R(\psi_{\emptyset}) = 5$. The other candidates all get rank ∞. Because $R(\psi_{a,c})$ is minimal, $\{a, c\}$ is the only JZ-ranking extension, i.e. $\mathcal{E}_{jz}(\mathcal{A}) = \{\{a, c\}\}$.

3-loop: $\{a, b, c\}$ with $a \rhd b \rhd c \rhd a$.

Semantics under the admissibility dogm reject $\{a\}, \{b\}, \{c\}$, only $\emptyset$ is admissible. But the JZ-model R verifies $R(\psi_a) = R(\psi_b) = R(\psi_c) = 4 < 5 = R(\psi_{\emptyset})$. Because all the alternatives are set to ∞, our ranking extensions are the maximal conflict-free sets $\{a\}, \{b\}, \{c\}$, i.e., $\mathcal{E}_{jz}$ clearly violates admissibility.

Attack on 2-loop: $\{a, b, c\}$ with $a \rhd b \rhd c \rhd b$.

We have $R(\psi_{\emptyset}) = 4, R(\psi_a) = 2, R(\psi_b) = R(\psi_c) = 3, R(\psi_{a,c}) = 1$, but ∞ for the others. Here $\mathcal{E}_{jz}(\mathcal{A}) = \{\{a, c\}\}$ picks up the canonical stable extension.

Attack from 2-loop: $\{a, b, c\}$ with $b \rhd a \rhd b \rhd c$.

We get $R(\psi_{\emptyset}) = 4, R(\psi_a) = 3, R(\psi_b) = 2, R(\psi_c) = 3$, $R(\psi_{a,b}) = R(\psi_{b,c}) = \infty$, and $R(\psi_{a,c}) = 2$. $\mathcal{E}_{jz}(\mathcal{A}) = \{\{b\}, \{a, c\}\}$ thus collects the stable extensions.

3,1-loop: $\{a, b, c\}$ with $a \rhd a \rhd b \rhd c \rhd a$.

$E = \emptyset$ is here the only admissible extension. The maximal coherent set is $\mathbb{A}^+ = \{b, c\}$, and we get $R(\psi_b) = 1, R(\psi_c) = 2$, as well as $R(\psi_{\emptyset}) = 3$. It follows that $\mathcal{E}_{jz}(\mathcal{A}) = \{\{b\}\}$, rejecting the stage extension $\{c\}$.

3,2-loop: $\{a, b, c\}$ with $b \rhd a \rhd b \rhd c \rhd a$.

We have $R(\psi_{\emptyset}) = 5, R(\psi_a) = 4, R(\psi_b) = 3$, and $R(\psi_c) = 3$, i.e. $\mathcal{E}_{jz}(\mathcal{A}) = \{\{b\}, \{c\}\}$. But the stable extension $\{b\}$ is the only admissible ranking extension.

The previous examples show that the ranking extension semantics $\mathcal{E}_{jz}$ can diverge considerably from all the other major proposals found in the literature. It may look as if the main difference is its more liberal attitude towards some non-admissible, but still justifiable extensions. However, the semantics has an even more exotic flavour. Consider the following scenarios, where we indicate the minimal extension weights $r_{\mathcal{A}}(E)$.

2-loop chain: $\{a, b, c\}, b \rhd a \rhd b \rhd c \rhd b$:
$r(\{a, c\}) = 1 < 2 = r(\{b\})$.

Splitted 3-chain: $\{a, b, c, d\}, a \rhd b \rhd c, a \rhd d \rhd c$:
$r(\{a, c\}) = r(\{b, d\}) = 4$.

Spoon: $\{a, b, c, d\}, a \rhd b \rhd c \rhd d \rhd c$:
$r(\{a, d\}) = r(\{a, c\}) = r(\{b, d\}) = 3$.

The first example documents the rejection of a stable extension, namely $\{b\}$. The second one illustrates the impact of quantitative considerations when dealing with a splitted variant of simple reinstatement. The third instance shows the coexistence of two stable extension with a non-admissible one. That is, even attack-free a can be questioned under certain circumstances. It follows that the above ranking semantic interpretation of argumentation frameworks deviates considerably from standard accounts and expectations. Let us now investigate how $\mathcal{E}_{jz}$ handles some common principles for extension semantics.

Isomorphy: $f : \mathcal{A} \cong \mathcal{A}'$ implies $f'' : \mathcal{E}(\mathcal{A}') \cong \mathcal{E}(\mathcal{A})$.

Conflict-freedom: If $a, b \in E \in \mathcal{E}(\mathcal{A})$, then $a \not\rhd b$.

CF-maximality: If $E \in \mathcal{E}(\mathcal{A})$, then E is a maximal conflict-free subset of $\mathbb{A}$.

Inclusion-maximality: If $E, E' \in \mathcal{E}(\mathcal{A})$ and $E \subseteq E'$, then $E = E'$.

Reinstatement: If $E \in \mathcal{E}(\mathcal{A})$, $a \in \mathbb{A}$, and for each $b \rhd a$ there is an $a' \in E$ with $a' \rhd b$, then $a \in E$.

Directionality: Let $\mathcal{A}_1 = (\mathbb{A}_1, \rhd_1), \mathcal{A}_2 = (\mathbb{A}_2, \rhd_2)$ be such that $\mathbb{A}_1 \cap \mathbb{A}_2 = \emptyset$, $\rhd_0 \subseteq A_1 \times A_2$, $\mathcal{A} = (\mathbb{A}_1 \cup \mathbb{A}_2, \rhd_1 \cup \rhd_0 \cup \rhd_2)$. Then we have $\mathcal{E}(\mathcal{A}_1) = \{E \cap \mathbb{A}_1 \mid E \in \mathcal{E}(\mathcal{A})\}$.

Theorem 7.1 (Basic properties)
$\mathcal{E}_{jz}$ verifies isomorphy, conflict-freedom, inclusion maximality, and CF-maximality. It falsifies reinstatement and directionality.

The first four features are easy consequences of the $\mathcal{E}_{jz}$-specification. The violation of reinstatement directly follows from how the semantics handles 3-loops. The spoon example documents the failure of directionality if we set $\mathbb{A}_1 = \{a, b\}$. But directionality also fails for other prominent approaches, like the semi-stable semantics. Note however that it can be indirectly enforced by using $\mathcal{E}_{jz}$ as the base function for an SCC-recursive semantics [BGG 05].

The following properties are inspired by the cumulativity principle for nonmonotonic reasoning. They state that if we drop an argument rejected by every extension, then this shouldn't add or erase skeptically supported arguments.

Rejection cumulativity: ($\mathcal{A}|B$: $\mathcal{A}$ restricted to B)
Rej-Cut: If $a \notin \cup\mathcal{E}(\mathcal{A})$, then $\cap\mathcal{E}(\mathcal{A}|\mathbb{A} - \{a\}) \subseteq \cap\mathcal{E}(\mathcal{A})$.
Rej-CM: If $a \notin \cup\mathcal{E}(\mathcal{A})$, then $\cap\mathcal{E}(\mathcal{A}) \subseteq \cap\mathcal{E}(\mathcal{A}|\mathbb{A} - \{a\})$.

Although our semantics relies on default inference notions verifying cumulativity at the level of $\mid\!\sim^{\mathcal{I}}_{\Delta}$, it nevertheless fails to validate the previous postulates.

Theorem 7.2 (No rejection cumulativity)
$\mathcal{E}_{jz}$ violates Rej-Cut and Rej-CM.

The counterexample for Rej-CUT is provided by $b \rhd c \rhd a \rhd b \rhd a$, because $\{b\} \not\subseteq \{b\} \cap \{c\}$. The one for Rej-CM is obtained by adding $c \rhd b$. Here $\{c\} \not\subseteq \{b\} \cap \{c\}$.

Another idea for combining plausibilistic default reasoning and argumentation theory has been presented in [KIS 11]. It combines defeasible logic programming with a prioritization criterion based on System Z. While it handles some benchmarks better than the individual systems do, its heterogeneous character makes it hard to assess. It doesn't share our goal to seek a plausibilistic semantics for abstract argumentation and seems to produce different results even in the generic context.

8 Conclusions

We have shown how the ranking construction paradigm for default reasoning can be exploited to interpret abstract argumentation frameworks and to specify corresponding extension semantics by using generic argument instantiations and distinguished canonical ranking models. We have considered structured and conditional logical instantiations, defined attack between inference pairs, and after a further abstraction step, introduced simple semantic instantiations, which interpret arguments by triples of premise, strict, and defeasible content. While our basic ranking extension semantics $\mathcal{E}_{jz}$ is intuitively appealing and has some interesting properties, it also exhibits a surprisingly unorthodox behaviour. This needs further exploration to see whether there are approaches which share the same semantic spirit but can avoid abnormalities conflicting with the standard argumentation philosophy. Actually, we have been able to develop an alternative semantics which seems to meet these demands, but it will have to be discussed elsewhere.

9 Bibliography

BGG 05 P. Baroni, M. Giacomin, G. Guida. SCC-recursiveness: a general schema for argumentation semantics. AIJ 168:163-210, 2005.

BSS 00 S. Benferhat, A. Saffiotti, P. Smets. Belief functions and default reasoning. Artificial Intelligence 122(1-2): 1-69, 2000.

GMP 93 M. Goldszmidt, P. Morris, J. Pearl. A maximum entropy approach to nonmonotonic reasoning. IEEE Transact. Patt. Anal. and Mach. Int, 15:220-232, 1993.

KI 01 G. Kern-Isberner. Conditionals in nonmonotonoic reasoning and belief revision, LNAI 2087. Springer, 2001.

KIS 11 G. Kern-Isberner G.R. Simari. A Default Logical Semantics for Defeasible Argumentation. Proc. of FLAIRS 2011, AAAI Press, 2011.

KLM 90 S. Kraus, D. Lehmann, M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. In *Artificial Intelligence*, 44:167-207, 1990.

Mak 94 D. Makinson. General patterns of nonmonotonic reasoning. Handbook of Logic in AI and LP, vol. 3 (eds. Gabbay et al.): 35-110. Oxford University Press, 1994.

Pea 90 J. Pearl. System Z: a natural ordering of defaults with tractable applications to nonmonotonic reasoning. TARK 3: 121-135. Morgan Kaufmann, 1990.

Spo 88 W. Spohn. Ordinal conditional functions: a dynamic theory of epistemic states. Causation in Decision, Belief Change, and Statistics (eds. W.L. Harper, B. Skyrms): 105-134. Kluwer, 1988.

Spo 12 W. Spohn. The Laws of Belief. Ranking Theory and Its Philosophical Applications. Oxford University Press, Oxford 2012.

Wey 94 E. Weydert. General belief measures. UAI'94, Morgan Kaufmann.

Wey 95 E. Weydert. Defaults and infinitesimals. Defeasible inference by non-archimdean entropy maximization. UAI 95: 540-547. Morgan Kaufmann, 1995.

Wey 96 E. Weydert. System J - rev. entailment. FAPR 96:637-649. Springer, 1996.

Wey 98 E. Weydert. System JZ - How to build a canonical ranking model of a default knowledge base. KR 98: 190-201. Morgan Kaufmann, 1998.

Wey 03 E. Weydert. System JLZ - Rational default reasoning by minimal ranking constructions. Journal of Applied Logic 1(3-4): 273-308. Elsevier, 2003.

Wey 11 E. Weydert. Semi-stable extensions for infinite frameworks. In Proc. BNAIC 2012: 336343.

Wey 13 E. Weydert. On the Plausibility of Abstract Arguments. ECSQARU 2013, LNAI 7958 (ed. L. van der Gaag): 522-533 Springer, 2013.

An Approach to Forgetting in Disjunctive Logic Programs that Preserves Strong Equivalence

James P. Delgrande
School of Computing Science
Simon Fraser University
Burnaby, B.C. V5A 1S6
Canada
jim@cs.sfu.ca

Kewen Wang
School of Information and Communication Technology
Griffith University,
Brisbane, QLD 4111
Australia
k.wang@griffith.edu.au

Abstract

In this paper we investigate forgetting in disjunctive logic programs, where forgetting an atom from a program amounts to a reduction in the signature of that program. The goal is to provide an approach that is syntax-independent, in that if two programs are strongly equivalent, then the results of forgetting an atom in each program should also be strongly equivalent. Our central definition of forgetting is impractical but satisfies this goal: Forgetting an atom is characterised by the set of SE consequences of the program that do not mention the atom to be forgotten. We then provide an equivalent, practical definition, wherein forgetting an atom p is given by those rules in the program that don't mention p, together with rules obtained by a single inference step from rules that do mention p. Forgetting is shown to have appropriate properties; as well, the finite characterisation results in a modest (at worst quadratic) blowup. Finally we have also obtained a prototype implementation of this approach to forgetting.

Introduction

Forgetting is an operation for eliminating variables from a knowledge base (Lin and Reiter 1994; Lang *et al.* 2003). It constitutes a reduction in an agent's language or, more accurately, signature, and has been studied under different names, such as variable elimination, uniform interpolation and relevance (Subramanian *et al.* 1997). Forgetting has various potential uses in a reasoning system. For example, in query answering, if one can determine what is relevant to a query, then forgetting the irrelevant part of a knowledge base may yield a more efficient operation. Forgetting may also provide a formal account and justification of predicate hiding, for example for privacy issues. As well, forgetting may be useful in summarising a knowledge base or reusing part of a knowledge base or in clarifying relations between predicates.

The best-known definition of forgetting is with respect to classical propositional logic, and is due to George Boole (Boole 1854). To forget an atom p from a formula ϕ in propositional logic, one disjoins the result of uniformly substituting $\top$ for p in ϕ with the result of substituting $\bot$; that is, forgetting is given by $\phi[p/\top] \vee \phi[p/\bot]$. (Lin and Reiter 1994) investigated the theory of forgetting for first order logic and its application in reasoning about action. Forgetting has been applied in resolving conflicts (Eiter and Wang 2008; Zhang and Foo 1997), and ontology comparison and reuse (Kontchakov *et al.* 2008; Konev *et al.* 2013).

The knowledge base of an agent may be represented in a non-classical logic, in particular a nonmonotonic approach such as answer set programming (ASP) (Gelfond and Lifschitz 1988; Baral 2003; Gebser *et al.* 2012). However, the Boole definition clearly does not extend readily to logic programs. In the past few years, several approaches have been proposed for forgetting in ASP (Eiter and Wang 2006; 2008; Wang *et al.* 2005; Zhang *et al.* 2005; Zhang and Foo 2006). The approach to forgetting in (Zhang *et al.* 2005; Zhang and Foo 2006) is syntactic, in the sense that their definition of forgetting is given in terms of program transformations, but is not based on answer set semantics or SE models[1] (for normal logic programs). A semantic theory of forgetting for normal logic programs under answer set semantics is introduced in (Wang *et al.* 2005), in which a sound and complete algorithm is developed based a series of program transformations. This theory is further developed and extended to disjunctive logic programs (Eiter and Wang 2006; 2008). However, this theory of forgetting is defined in terms of standard answer set semantics instead of SE models.

In order to use forgetting in its full generality, for dealing with relevance or predicate hiding, or in composing, decomposing, and reusing answer set programs, it is desirable for a definition to be given in terms of the *logical content* of a program, that is in terms of SE models. For example, the reuse of knowledge bases requires that when a sub-program Q in a large program P is substituted with another program Q', the resulting program should be equivalent to P. This is not the case for answer set semantics due to its nonmonotonicity. As a result, two definitions of forgetting have been introduced in HT-logic (Wang *et al.* 2012; 2013). These approaches indirectly establish theories of forgetting under SE models as HT-logic provides a natural extension of SE models. The approach to interpolation for equilibrium logic introduced in (Gabbay *et al.* 2011) is more general than forgetting. However, the issue of directly establishing a theory of forgetting for disjunctive logic programs under SE models is still not fully resolved yet. In addition, it is even more challenging to develop efficient algorithm for computing a result of forgetting under SE models.

[1] See the next section for definitions.

A key intuition behind forgetting is that the logical consequences of a set of formulas that don't mention forgotten symbols should still be believed after forgetting. This leads to a very simple (abstract) knowledge-level definition, provided that a consequence operator is provided in the underlying logic. In particular, the semantics of a logic usually associates a set of models $Mod(\mathcal{K})$ with each knowledge base $\mathcal{K}$. This makes it straightforward to formulate a definition of forgetting based on the above intuition. However, such a definition of forgetting suffers from the problem of inexpressibility, i.e., the result of forgetting may not be expressible in the logic. In this paper, we establish such a theory of forgetting for disjunctive logic programs under SE models. Besides several important properties, we show that the result of forgetting for a given disjunctive program is still a disjunctive program. This result confirms the existence and expressibility of forgetting for DLP under SE models and in fact provides an algorithm for computing forgetting under SE models. We investigate some optimisation techniques for the algorithm and report a prototype implementation of the algorithm.

Answer Set Programming

Here we briefly review pertinent concepts in answer set programming; for details see (Gelfond and Lifschitz 1988; Baral 2003; Gebser *et al.* 2012).

Let $\mathcal{A}$ be an alphabet, consisting of a set of *atoms*. A (*disjunctive*) *logic program* over $\mathcal{A}$ is a finite set of rules of the form

$$a_1; \ldots; a_m \leftarrow b_1, \ldots, b_n, {\sim}c_1, \cdots, {\sim}c_p. \qquad (1)$$

where $a_i, b_j, c_k \in \mathcal{A}$, and $m, n, p \geq 0$ and $m + n + p > 0$. Binary operators ';' and ',' express disjunction and conjunction respectively. For atom a, ${\sim}a$ is (default) negation. We will use $\mathcal{L}_{\mathcal{A}}$ to denote the language (viz. set of rules) generated by $\mathcal{A}$.

Without loss of generality, we assume that there are no repeated literals in a rule. The *head* and *body* of a rule r, $H(r)$ and $B(r)$, are defined by:

$$\begin{array}{rcll} H(r) & = & \{a_1, \ldots, a_m\} & \text{and} \\ B(r) & = & \{b_1, \ldots, b_n, {\sim}c_1, \ldots, {\sim}c_p\}. & \end{array}$$

Given a set X of literals, we define

$$\begin{array}{rcl} X^+ & = & \{a \in \mathcal{A} \mid a \in X\}, \\ X^- & = & \{a \in \mathcal{A} \mid {\sim}a \in X\}, \text{ and} \\ {\sim}X & = & \{{\sim}a \mid a \in X \cap \mathcal{A}\}. \end{array}$$

For simplicity, we sometimes use a set-based notation, expressing a rule as in (1) as

$$H(r) \leftarrow B(r)^+, {\sim}B(r)^- .$$

The *reduct* of a program P with respect to a set of atoms Y, denoted P^Y, is the set of rules:

$$\{H(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap Y = \emptyset\}.$$

Note that the reduct consists of negation-free rules only. An *answer set* Y of a program P is a subset-minimal model of P^Y. A program induces 0, 1, or more *answer sets*. The set of all answer sets of a program P is denoted by $AS(P)$. For example, the program $P = \{a \leftarrow .\quad c; d \leftarrow a, {\sim}b\}$ has answer sets $AS(P) = \{\{a, c\}, \{a, d\}\}$. Notably, a program is nonmonotonic with respect to its answer sets. For example, the program $\{q \leftarrow {\sim}p\}$ has answer set $\{q\}$ while $\{q \leftarrow {\sim}p.\ \ p \leftarrow\}$ has answer set $\{p\}$.

SE Models

As defined by (Turner 2003), an *SE interpretation* on a signature $\mathcal{A}$ is a pair (X, Y) of interpretations such that $X \subseteq Y \subseteq \mathcal{A}$. An SE interpretation is an *SE model* of a program P if $Y \models P$ and $X \models P^Y$, where $\models$ is the relation of logical entailment in classical logic. The set of all SE models of a program P is denoted by $SE(P)$. Then, Y is an answer set of P iff $(Y, Y) \in SE(P)$ and no $(X, Y) \in SE(P)$ with $X \subset Y$ exists. Also, we have $(Y, Y) \in SE(P)$ iff $Y \in Mod(P)$.

A program P is *satisfiable* just if $SE(P) \neq \emptyset$.[2] Thus, for example, we consider $P = \{p \leftarrow {\sim}p\}$ to be satisfiable, since $SE(P) \neq \emptyset$ even though $AS(P) = \emptyset$. Two programs P and Q are *strongly equivalent*, symbolically $P \equiv_s Q$, iff $SE(P) = SE(Q)$. Alternatively, $P \equiv_s Q$ holds iff $AS(P \cup R) = AS(Q \cup R)$, for every program R (Lifschitz *et al.* 2001). We also write $P \models_s Q$ iff $SE(P) \subseteq SE(Q)$.

SE Consequence

While the notion of SE models puts ASP on a monotonic footing with respect to model theory, (Wong 2008) has subsequently provided an inferential system for rules that preserves strong equivalence, where his notion of *SE consequence* is shown to be sound and complete with respect to the semantic notion of *SE models*. His inference system is given as follows, where lower case letters are atoms, upper case are sets of atoms, and for a set of atoms $C = \{c_1, \ldots, c_n\}$, ${\sim}C$ stands for $\{{\sim}c_1, \ldots, {\sim}c_n\}$.

Inference Rules for SE Consequence:

Taut $x \leftarrow x$

Contra $\leftarrow x, {\sim}x$

Nonmin From $A \leftarrow B, {\sim}C$ infer
$A; X \leftarrow B, Y, {\sim}C, {\sim}Z$

WGPPE From $A_1 \leftarrow B_1, x, {\sim}C_1$ and
$A_2; x \leftarrow B_2, {\sim}C_2$ infer
$A_1; A_2 \leftarrow B_1, B_2, {\sim}C_1, {\sim}C_2$

S-HYP From $A_1 \leftarrow B_1, {\sim}x_1, {\sim}C_1,$
$\ldots,$
$A_n \leftarrow B_n, {\sim}x_n, {\sim}C_n,$
$A \leftarrow x_1, \ldots, x_n, {\sim}C$ infer
$A_1; \ldots; A_n \leftarrow$
$B_1, \ldots, B_n, {\sim}C_1, \ldots, {\sim}C_n, {\sim}A, {\sim}C$

[2] Note that many authors in the literature define satisfiability in terms of answer sets, in that for them a program is satisfiable if it has an answer set, i.e., $AS(P) \neq \emptyset$.

Several of these rules are analogous to or similar to well-known rules in the literature. For example, **Nonmin** is weakening; **WGPPE** is analogous to cut; and **S-HYP** is a version of hyper-resolution. Let $\vdash_s$ denote the consequence relation generated by these rules, for convenience allowing sets of rules on the right hand side of $\vdash_s$. Then $P \leftrightarrow_s P'$ abbreviates $P \vdash_s P'$ and $P' \vdash_s P$. As well, define

$$\mathcal{C}n_{\mathcal{A}}(P) = \{r \in \mathcal{L}_{\mathcal{A}} \mid P \vdash_s r\}.$$

Then the above set of inference rules is sound and complete with respect to the entailment $\models_s$.

Theorem 1 ((Wong 2008)) *$P \models_s r$ iff $P \vdash_s r$.*

The Approach

Formal Preliminaries

Since forgetting in our approach amounts to decreasing the alphabet, or signature, of a logic program, we need additional notation for relating signatures. Let $\mathcal{A}$ and $\mathcal{A}'$ be two signatures where $\mathcal{A}' \subset \mathcal{A}$. Then $\mathcal{A}'$ is a *reduction*[3] of $\mathcal{A}$, and $\mathcal{A}$ is an *expansion* of $\mathcal{A}'$. Furthermore, if w is an SE interpretation on $\mathcal{A}$ and w' is an SE interpretation on $\mathcal{A}'$ where w and w' agree on the interpretation of symbols in $\mathcal{A}'$ then w' is the *$\mathcal{A}$-reduction* of w, and w is an *$\mathcal{A}'$-expansion* of w'. For fixed $\mathcal{A}' \subset \mathcal{A}$, reductions are clearly unique whereas expansions are not.

For a logic program P, $\sigma(P)$ denotes the signature of P, that is, the set of atoms mentioned in P. SE models are defined with respect to an understood alphabet; for SE model w we also use $\sigma(w)$ to refer to this alphabet. Thus for example if $\mathcal{A} = \{a, b, c\}$ then, with respect to $\mathcal{A}$, the SE model $w = (\{a\}, \{a, b\})$ is more perspicuously written as $(\{a, \neg b, \neg c\}, \{a, b, \neg c\})$, and so in this case $\sigma(w) = \{a, b, c\}$.

If $\mathcal{A}' \subset \mathcal{A}$ and for SE models w, w' we have $\sigma(w) = \mathcal{A}$ and $\sigma(w') = \mathcal{A}'$ then we use $w_{|\mathcal{A}'}$ to denote the reduction of w with respect to $\mathcal{A}'$ and we use $w'_{\uparrow \mathcal{A}}$ to denote the set of expansions of w' with respect to $\mathcal{A}$. This notation extends to sets of models in the obvious way. As well, we use the notion of a reduction for logic programs; that is, for $\mathcal{A}' \subseteq \mathcal{A}$,

$$P_{|\mathcal{A}'} = \{r \in P \mid \sigma(r) \subseteq \mathcal{A}'\}.$$

An Abstract Characterisation of Forgetting

As described, our goal is to define forgetting with respect to the logical content of a logic program. For example, if we were to forget b from the program $\{a \leftarrow b.,\ b \leftarrow c.\}$, we would expect the rule $a \leftarrow c$ to be in the result, since it is implicit in the original program. Consequently, our primary definition is the following.

Definition 1 *Let P be a disjunctive logic program over signature $\mathcal{A}$. The result of* forgetting *$\mathcal{A}'$ in P, denoted $Forget(P, \mathcal{A}')$, is given by:*

$$Forget(P, \mathcal{A}') = \mathcal{C}n_{\mathcal{A}}(P) \cap \mathcal{L}_{\mathcal{A} \setminus \mathcal{A}'}.$$

[3]The standard term in model theory is *reduct* (Chang and Keisler 2012; Doets 1996; Hodges 1997). However *reduct* has its own meaning in ASP, and so we adopt this variation.

That is, the result of forgetting a set of atoms $\mathcal{A}'$ in program P is simply the set of SE consequences that of P over the original alphabet, but excluding atoms from $\mathcal{A}'$.

This definition is very simple. This characterization is abstract, at the *knowledge level*. As a consequence, many formal results are very easy to show. On the other hand, the definition is not immediately practically useful since forgetting results in an infinite set of rules. Consequently a key question is to determine a finite characterisation (that is to say, a uniform interpolant) of $Forget$. We explore these issues next.

The following results are elementary, but show that the definition of forgetting has the "right" properties.

Proposition 1 *Let P and P' be disjunctive logic program and let $\mathcal{A}$ (possibly primed or subscripted) be alphabets.*

1. $P \vdash_s Forget(P, \mathcal{A})$
2. *If* $P \leftrightarrow_s P'$ *then* $Forget(P, \mathcal{A}) \leftrightarrow_s Forget(P', \mathcal{A})$
3. $Forget(P, \mathcal{A}) = \mathcal{C}n_{\mathcal{A}'}(Forget(P, \mathcal{A}))$ *where* $\mathcal{A}' = \sigma(P) \setminus \mathcal{A}$.
4. $Forget(P, \mathcal{A}) = Forget(Forget(P, \mathcal{A} \setminus \{a\}), \{a\}))$
5. $Forget(P, \mathcal{A}_1 \cup \mathcal{A}_2) = Forget(Forget(P, \mathcal{A}_1), \mathcal{A}_2))$
6. P *is a* conservative extension *of* $Forget(P, \mathcal{A})$.

Thus, forgetting results in no consequences not in the original theory. As well, the result of forgetting is independent of syntax and yields a deductively-closed theory (Parts 2 and 3). Part 4 gives an iterative means of determining forgetting on an element-by-element basis. The next part, which generalises the previous, shows that forgetting is decomposable with respect to a signature, which in turn implies that forgetting is a commutative operation with respect to its second argument. Last, P is a conservative extension of the result of forgetting, which is to say, trivially $\sigma(P) \setminus \mathcal{A}' \subseteq \sigma(P)$, and the consequences of P and $Forget(P, \mathcal{A})$ coincide over the language $\mathcal{L}_{\sigma(P) \setminus \mathcal{A}'}$.

With regards to SE models, we obtain the following results giving an alternative characterisation of forgetting. Here only we use the notation $SE_{\mathcal{A}}(P)$ to indicate the SE models of program P over alphabet $\mathcal{A}$.

Proposition 2 *Let $\mathcal{A}' \subseteq \mathcal{A}$, and let $\sigma(P) \subseteq \mathcal{A}$.*

1. $SE_{\mathcal{A} \setminus \mathcal{A}'}(Forget(P, \mathcal{A}')) = SE_{\mathcal{A}}(P)_{|(\mathcal{A} \setminus \mathcal{A}')}$
2. $SE_{\mathcal{A}}(Forget(P, \mathcal{A}')) = (SE_{\mathcal{A}}(P)_{|(\mathcal{A} \setminus \mathcal{A}')})_{\uparrow \mathcal{A}}$

The first part provides a semantic characterisation of forgetting: the SE models of $Forget(P, \mathcal{A}')$ are exactly the SE models of P restricted to the signature $\mathcal{A} \setminus \mathcal{A}'$. Very informally, what this means is that the SE models of $Forget(P, \mathcal{A}')$ can be determined by simply dropping the symbols in $\mathcal{A}'$ from the SE models of P. The second part, which is a simple corollary of the first, expresses forgetting with respect to the original signature.

Of course, one may wish to re-express the effect of forgetting in the original language of P; in fact, many approaches to forgetting assume that the underlying language

is unchanged. To this end, we can consider a variant of Definition 1 as follows, where $\mathcal{A}' \subseteq \mathcal{A}$.

$$Forget_{\mathcal{A}}(P, \mathcal{A}') \equiv \mathcal{C}n_{\mathcal{A}}(Forget(P, \mathcal{A}')) \qquad (2)$$

That is, $Forget(P, \mathcal{A}')$ is re-expressed in the original language with signature $\mathcal{A}$. The result is a theory over the original language, but where the resulting theory carries no contingent information about the domain of application regarding elements of $\mathcal{A}'$.

The following definition is useful in stating results concerning forgetting.

Definition 2 *Signature $\mathcal{A}$ is* irrelevant *to P, $IR(P, \mathcal{A})$, iff there is P' such that $P \leftrightarrow_s P'$ and $\sigma(P') \cap \mathcal{A} = \emptyset$.*

Zhang and Zhou (2009) give four postulates characterising their approach to forgetting in the modal logic S5. An analogous result follows here with respect to forgetting re-expressed in the original signature:

Proposition 3 *Let $\mathcal{A}' \subseteq \mathcal{A}$ and let $\sigma(P)$, $\sigma(P') \subseteq \mathcal{A}$. Then $P' = Forget_{\mathcal{A}}(P, \mathcal{A}')$ iff*

1. $P \vdash_s P'$
2. *If $IR(r, \mathcal{A}')$ and $P \vdash_s r$ then $P' \vdash_s r$*
3. *If $IR(r, \mathcal{A}')$ and $P \nvdash_s r$ then $P' \nvdash_s r$*
4. $IR(P', \mathcal{A}')$

For the last three parts we have that, if a rule r is independent of a signature $\mathcal{A}'$, then forgetting $\mathcal{A}'$ has no effect on whether that formula is a consequence of the original knowledge base or not (Parts 2 and 3). The last part is a "success" postulate: the result of forgetting $\mathcal{A}'$ yields a theory expressible without $\mathcal{A}'$.

A Finite Characterisation of Forgetting

Aside: Forgetting in Propositional Logic We first take a quick detour to forgetting in propositional logic to illustrate the general approach to finitely characterising forgetting. Let ϕ be a formula in propositional logic and let p be an atom; the standard definition for forgetting p from ϕ in propositional logic is defined to be $\phi[p/\top] \vee \phi[p/\bot]$. It is not difficult to show that this is equivalent to Definition 1, but suitably re-expressed in terms of propositional logic. This definition however is not particularly convenient. It is applicable only to finite sets of formulas. As well, it results in a formula whose main connective is a disjunction.

An alternative is given as follows. Assume that a formula (or formulas) for forgetting is expressed in clause form, where a (disjunctive) clause is expressed as a set of literals. For forgetting an atom p, consider the set of all clauses obtained by resolving on p:

Definition 3 *Let S be a set of propositional clauses and $p \in \mathcal{P}$. Define*

$$\begin{aligned} Res(S, p) = \ \{\phi \mid \exists \phi_1, \phi_2 \in S \text{ such that} \\ p \in \phi_1 \text{ and } \neg p \in \phi_2, \ \text{ and} \\ \phi = (\phi_1 \setminus \{p\}) \cup (\phi_2 \setminus \{\neg p\})\} \end{aligned}$$

We obtain the following, where $Forget_{PC}$ refers to forgetting in propositional logic:

Theorem 2 *Let S be a set of propositional clauses over signature $\mathcal{P}$ and $p \in \mathcal{P}$.*

$$Forget_{PC}(P, p) \ \leftrightarrow \ S_{|(\mathcal{P} \setminus \{p\})} \cup Res(S, p).$$

This provides an arguably more convenient means of computing forgetting, in that it is easily implementable, and one remains with a set of clauses.

Back to Forgetting in Logic Programming: We can use the same overall strategy for computing forgetting in a disjunctive logic program. In particular, for forgetting an atom a, we can use the inference rules from (Wong 2008) to compute "resolvents" of rules that don't mention a. It proves to be the case that the corresponding definition is a bit more intricate, since it involves various combinations of **WGPPE** and **S-HYP**, but overall the strategy is the same as for propositional logic.

In the definition below, $ResLP$ corresponds to Res for forgetting in propositional logic. In propositional logic, Res was used to compute all resolvents on an atom a. Here the same thing is done: we consider instances of **WGPPE** and **S-HYP** in place of propositional resolution; these instances are given by the two parts of the union, respectively, below.

Definition 4 *Let P be a disjunctive logic program and $a \in \mathcal{A}$.*

Define:

$$\begin{aligned} &ResLP(P, a) = \\ &\quad \{r \mid \exists r_1, r_2 \in P \text{ such that} \\ &\qquad r_1 = A_1 \leftarrow B_1, a, {\sim}C_1, \\ &\qquad r_2 = A_2; a \leftarrow B_2, {\sim}C_2, \\ &\qquad r = A_1; A_2 \leftarrow B_1, B_2, {\sim}C_1, {\sim}C_2 \ \} \\ &\quad \cup \\ &\quad \{r \mid \exists r_1, \ldots, r_n, r' \in P \text{ such that } a = a_1 \\ &\qquad r_i = A_i \leftarrow B_i, {\sim}a_i, {\sim}C_i, \quad 1 \leq i \leq n \\ &\qquad r' = A \leftarrow a_1, \ldots a_n, {\sim}C \quad \text{and} \\ &\qquad r = A_1; \ldots; A_n \leftarrow \\ &\qquad\qquad B_1, \ldots, B_n, {\sim}C_1, \ldots, {\sim}C_n, {\sim}A, {\sim}C \ \} \end{aligned}$$

We obtain the following:

Theorem 3 *Let P be a disjunctive logic program over $\mathcal{A}$ and $a \in \mathcal{A}$. Assume that any rule $r \in P$ is satisfiable, non-tautologous, and contains no redundant occurrences of any atom.*

Then:

$Forget(P, a) \leftrightarrow_s P_{|(\mathcal{A} \setminus \{a\})} \ \cup \ ResLP(P, a)$.

Proof Outline: From Definition 1, $Forget(P, a)$ is defined to be the set of those SE consequences of program P that do not mention a. Thus for disjunctive rule r, $r \in Forget(P, a)$ means that $P \vdash_s r$ and $a \notin \sigma(r)$. Thus the left-to-right direction is immediate: Any $r \in P_{|(\mathcal{A} \setminus \{a\})}$ or $r \in ResLP(P, a)$ is a SE consequence of P that does not mention a.

For the other direction, assume that we have a proof of r from P, represented as a sequence of rules. If no rule in the proof mentions a, then we are done. Otherwise, since r

does not mention a, there is a last rule in the proof, call it r_n that does not mention a, but is obtained from rules that do mention a. The case where r_n is obtained via **Taut**, **Contra**, or **Nonmin** is easily handled. If r_n is obtained via **WGPPE** or **S-HYP** then there are rules r_k and r_l that mention a (and perhaps other rules in the case of **S-HYP**). If r_k,$r_l \in P$ then $r_n \in ResLP(P, a)$. If one of r_k, r_l is not in P (say, r_k) then there are several cases, but in each case it can be shown that the proof can be transformed to another proof where the index of r_k in the proof sequence is decreased and the index of no rule mentioning a is increased. This process must terminate (since a proof is a finite sequence), where the premisses of the proof are either rules of P that do not mention a, elements of $ResLP(P, a)$, or tautologies.

Consider the following case, where $r_n = A_1; A_2; A_3 \leftarrow B_1, B_2, B_3$, and we use the notation that each A_i is a set of implicitly-disjoined atoms while each B_i is a set of implicitly-conjoined literals. Assume that r_n is obtained by an application of **WGPPE** from $r_k = a; A_1; A_2 \leftarrow B_1, B_2$ and $r_l = A_3 \leftarrow a, B_3$. Assume further that r_k is obtained from $r_i = a; b; A_1 \leftarrow B_1$ and $r_j = A_2 \leftarrow b, B_2$ by an application of **WGPPE**. This situation is illustrated in Figure 1a.

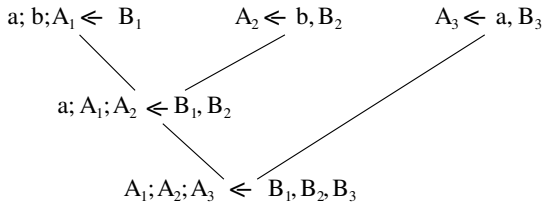


Figure 1a

Then essentially the steps involving the two applications of **WGPPE** can be "swapped", as illustrated in Figure 1b, where r_k is replaced by $r'_k = a; A_1; A_2 \leftarrow B_1, B_2$.

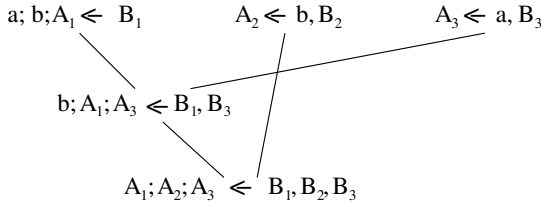


Figure 1b

Thus the step involving a is informally "moved up" in the proof. There are 12 other cases, involving various combinations of the inference rules, but all proceed the same as in the above. □

The theorem is expressed in terms of forgetting a single atom. Via Proposition 1.4 this readily extends to forgetting a set of atoms. Moreover, since we inherit the results of Propositions 1 and 3, we get that the results of forgetting are independent of syntax, even though the expression on the right hand side of Theorem 3 is a set of rules obtained by transforming and selecting rules in P. It can also be observed that forgetting an atom results in at worst a quadratic blowup in the size of the program. While this may seem comparatively modest, it implies that forgetting a set of atoms may result in an exponential blowup.

Example 1 *Let $P = \{p \leftarrow \sim q.\ \ r \leftarrow p\}$. Forgetting p yields $\{r \leftarrow \sim q\}$ (where $r \leftarrow \sim q$ is obtained by an application of* **WGPPE***), while forgetting q and r yield programs $\{r \leftarrow p\}$ and $\{p \leftarrow \sim q\}$ respectively.*

Computation of Forgetting

By Theorem 3, we have the following algorithm for computing the result of forgetting. A rule r is a tautology if it is of the form $r = A; b \leftarrow b, B, \sim C$; a rule r is a contradictory if it is of the form $r = A; c \leftarrow B, \sim c, \sim C$; a rule r is minimal if there is no rule r' in P such that $B(r') \subseteq B(r)$, $H(r') \subseteq H(r)$ and one of these two subset relations is proper; otherwise, r is non-minimal.

Algorithm 1 (Computing a result of forgetting)
Input: Disjunctive program P and literal a in P.
Output: $Forget(P, a)$.
Procedure:

Step 1. Remove tautology rules, contradiction rules and non-minimal rules from P. The resulting disjunctive program is still denoted P.

Step 2. Collect all rules in P that do not contain the atom a, denoted P'.

Step 3. For each pair of rules $r_1 = A_1 \leftarrow B_1, a, \sim C_1$ and $r_2 = A_2; a \leftarrow B_2, \sim C_2$, add the rule $r = A_1; A_2 \leftarrow B_1, B_2, \sim C_1, \sim C_2$ to P'

Step 4. For each rule $r' = A \leftarrow a_1, \ldots a_n, \sim C$ where for some i, $a_i = a$, and for each set of n rules $\{r_i = A_i \leftarrow B_i, \sim a_i, \sim C_i \mid 1 \leq i \leq n\}$, add the rule $r = A_1; \ldots; A_n \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_n, \sim A, \sim C$ to P'.

Step 5. Return P' as $Forget(P, a)$.

Some remarks for the algorithm are in order. Obviously, Step 1 is to preprocesss the input program by eliminating tautology rules, contradiction rules and non-minimal rules from P. Initially, all rules that do not contain a, which are trivial SE-consequences of P, are included in the result of forgetting. In many practical applications, such a part of input program is usually not very large and thus forgetting can be efficiently done although the input program can be very large. Step 3 and Step 4 implement two resolution rules **WGPPE** and **S-HYP**, respectively.

Conflict Resolving by Forgetting: Revisited

(Eiter and Wang 2006; 2008) explore how their semantic forgetting for logic programs can be used to resolve conflicts in multi-agent systems. However, their notion of forgetting is based on answer sets and thus does not preserve the syntactic structure of original logic programs, as pointed out in (Cheng *et al.* 2006). In this subsection, we demonstrate how this shortcoming of Eiter and Wang's forgetting can be overcome in our SE-forgetting for disjunctive programs.

The basic idea of conflict resolving (Eiter and Wang 2006; 2008) consists of two observations:

1. each answer set corresponds to an agreement among some agents;

2. conflicts are resolved by forgetting some literals/concepts for some agents/ontologies.

Definition 5 *Let $\mathcal{S} = (P_1, P_2, \ldots, P_n)$, where each logic program P_i represents the preferences/constraints of Agent i. A* compromise *of $\mathcal{S}$ is a sequence $C = (F_1, F_2, \ldots, F_n)$ where each F_i is a set of atoms to be forgotten from P_i. An* agreement *of $\mathcal{S}$ on C is an answer set of* $\mathsf{forget}(\mathcal{S}, C) = \mathsf{forget}(P_1, F_1) \cup \mathsf{forget}(P_2, F_2) \cup \cdots \cup \mathsf{forget}(P_n, F_n)$.

For specific applications, we may need to impose certain conditions on each F_i. However, the two algorithms (Algorithms 1 and 2) in (Cheng *et al.* 2006) may not produce intuitive results if directly used in a practical application. Consider a simple scenario with two agents.

Example 2 *(Cheng* et al. *2006) Suppose that two agents A1 and A2 try to reach an agreement on submitting a paper to a conference, as a* regular paper *or as a* system description*. If a paper is prepared as a system description, then the system may be implemented either in Java or Prolog. The preferences and constraints are as follows.*

1. *The same paper cannot be submitted as both a regular paper and system description.*
2. *A1 would like to submit the paper as a regular one and, in case the paper is submitted as a system description and there is no conflict, he would prefer to use Java.*
3. *A2 would like to submit the paper as a system description but not prefer regular paper.*

Obviously, the preferences of these two agents are jointly inconsistent and thus it is impossible to satisfy both at the same time. The scenario can be encoded as a collection of three disjunctive programs (P_0 stands for general constraints): $\mathcal{S} = (P_0, P_1, P_2)$ where R, S, J, P mean "regular paper," "system description," "Java" and "Prolog," respectively: $P_0 = \{\leftarrow R, S\}, P_1 = \{R \leftarrow . \quad J \leftarrow S, \sim P\}, P_2 = \{\leftarrow R. \quad S \leftarrow\}$.

Intuitively, if A_1 can make a compromise by forgetting R, then there will be an agreement $\{S, J\}$, that is, a system description is prepared and Java is used for implementing the system. However, if we directly use forgetting in conflict resolution, by forgetting R, we can only obtain an agreement $\{S\}$ which does not contain J. In fact, this is caused by the removal of $J \leftarrow S, \sim P$ in the process of forgetting. This rule is abundant in P_1 but becomes relevant when we consider the interaction of A1 with other agents (here A2).

As pointed out in (Cheng *et al.* 2006), it is necessary to develop a theory of forgetting for disjunctive programs such that *locally abundant* (or *locally irrelevant*) rules in the process of forgetting can be preserved. Our SE forgetting provides an ideal solution to the above problem. This can be seen from the definition of SE-forgetting and Algorithm 1 (if needed, we don't have to eliminate non-minimal rules in Step 1). In fact, $Forget(P_1, R) = \{J \leftarrow S, \sim P\}$, which preserves the locally redundant rule $J \leftarrow S, \sim P$.

Conclusion

In this paper we have addressed forgetting under SE models in disjunctive logic programs, wherein forgetting amounts to a reduction in the signature of a program. Essentially, the result of forgetting an atom (or set of atoms) from a program is the set of SE consequences of the program that do not mention that atom or set of atoms. This definition then is at the *knowledge level*, that is, it is abstract and is independent of how a program is represented. Hence this theory of forgetting is useful for tasks such as knowledge base comparison and reuse. A result of the proposed forgetting under SE models is also a result of forgetting under answer sets but not vice versa. Moreover, we have developed an efficient algorithm for computing forgetting in disjunctive logic programs, which is complete and sound with respect to the original knowledge-level definition.

A prototype implementation, of forgetting has been implemented in Java and is available publicly at `http://www.ict.griffith.edu.au/~kewen/SE-Forget/`. While our experiments on the efficiency of the system are still underway, preliminary results show that the algorithm is very efficient. Currently we are still working on improving efficiency of the implementation and are experimenting on applying it to large practical logic programs and randomly generated programs. We plan to apply this notion of forgetting to knowledge base comparison and reuse. For future work we also plan to investigate a similar approach to forgetting for other classes of logic programs.

References

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

George Boole. *An Investigation of the Laws of Thought*. Walton, London, 1854. (Reprinted by Dover Books, New York, 1954).

Chen C. Chang and H. Jerome Keisler. *Model Theory*. Dover Publications, third edition, 2012.

Fu-Leung Cheng, Thomas Eiter, Nathan Robinson, Abdul Sattar, and Kewen Wang. LPForget: A system of forgetting in answer set programming. In *Proceedings of the 19th Joint Australian Conference on Artificial Intelligence*, pages 1101–1105, 2006.

Kees Doets. *Basic Model Theory*. CSLI Publications, 1996.

Thomas Eiter and Kewen Wang. Forgetting and conflict resolving in disjunctive logic programming. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 238–243. AAAI Press, 2006.

Thomas Eiter and Kewen Wang. Forgetting in answer set programming. *Artificial Intelligence*, 172(14):1644–1672, 2008.

Dov M. Gabbay, David Pearce, and Agustín Valverde. Interpolable formulas in equilibrium logic and answer set programming. *J. Artif. Intell. Res. (JAIR)*, 42:917–943, 2011.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. The MIT Press, 1988.

Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, Cambridge, UK, 1997.

Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter. Model-theoretic inseparability and modularity of description logic ontologies. *Artificial Intelligence*, 203:66–103, 2013.

Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Can you tell the difference between DL-Lite ontologies? In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-08)*, pages 285–295, 2008.

J. Lang, P. Liberatore, and P. Marquis. Propositional independence : Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.

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

F. Lin and R. Reiter. Forget it! In *AAAI Fall Symposium on Relevance*, New Orleans, November 1994.

D. Subramanian, R. Greiner, and J. Pearl. Special issue on relevance. *Artificial Intelligence*, 97(1-2), 1997.

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

Kewen Wang, Abdul Sattar, and Kaile Su. A theory of forgetting in logic programming. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 682–688. AAAI Press, 2005.

Yisong Wang, Yan Zhang, Yi Zhou, and Mingyi Zhang. Forgetting in logic programs under strong equivalence. In *Proceedings of the Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

Yisong Wang, Kewen Wang, and Mingyi Zhang. Forgetting for answer set programming revisited. In *Proceedings, The 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1162–1168, 2013.

Ka-Shu Wong. Sound and complete inference rules for SE-consequence. *Journal of Artificial Intelligence Research*, 31(1):205–216, January 2008.

Y. Zhang and N. Foo. Answer sets for prioritized logic programs. In *Proceedings of the International Symposium on Logic Programming (ILPS-97)*, pages 69–84. MIT Press, 1997.

Yan Zhang and Norman Foo. Solving logic program conflict through strong and weak forgetting. *Artificial Intelligence*, 170:739–778, 2006.

Yan Zhang and Yi Zhou. Knowledge forgetting: Properties and applications. *Artificial Intelligence*, 173(16-17):1525–1537, November 2009.

Yan Zhang, Norman Y. Foo, and Kewen Wang. Solving logic program conflict through strong and weak forgettings. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 627–634, 2005.

Three Semantics for Modular Systems

Shahab Tasharrofi and Eugenia Ternovska
Simon Fraser University, email: {ter, sta44}@cs.sfu.ca

Abstract

In this paper, we further develop the framework of Modular Systems that lays model-theoretic foundations for combining different declarative languages, agents and solvers. We introduce a multi-language logic of modular systems. We define two novel semantics, a structural operational semantics, and an inference-based semantics. We prove the new semantics are equivalent to the original model-theoretic semantics and describe future research directions.

Introduction

Modular Systems (MS) (Tasharrofi and Ternovska 2011) is a language-independent formalism representing and solving complex problems specified declaratively. There are several motivations for introducing the MS formalism:

- the need to be able to split a large problem into subproblems, and to use the most suitable formalism for each part,
- the need to model distributed combinations of programs, knowledge bases, languages, agents, etc.,
- the need to model collaborative solving of complex tasks, such as in satisfiability-based solvers.

The MS formalism gave a unifying view, through a semantic approach, to formal and declarative modelling of modular systems. In that initial work, individual modules were considered from both model-theoretic and operational view. Under the model-theoretic view, a module is a set (or class) of structures, and under the operational view it is an operator, mapping a subset of the vocabulary to another subset. An abstract algebra on modules was given. It is similar to Codd's relational algebra and allows one to combine modules on abstract model-theoretic level, independently from what languages are used for describing them. An important operation in the algebra is the loop (or feedback) operation, since iteration underlies many solving methods. We showed that the power of the loop operator is such that the combined modular system can capture all of the complexity class NP even when each module is deterministic and polytime. Moreover, in general, adding loops gives a jump in the polynomial time hierarchy, one step from the highest complexity of the components. It is also shown that each module can be viewed as an operator, and when each module is (anti-) monotone, the number of the potential solutions can be reduced by using ideas from the logic programming community.

Inspired by practical combined solvers, the authors of (Tasharrofi, Wu, and Ternovska 2011; 2012) introduced an algorithm to solve model expansion tasks for modular systems. The evolution processes of different modules are jointly considered. The algorithm incrementally constructs structures for the expanded vocabulary by communicating with oracles associated with each module, who provide additional information in the form of reasons and advice to navigate the search. It was shown that the algorithm closely corresponds to what is done in practice in different areas such as Satisfiability Modulo Theories (SMT), Integer Linear Programming (ILP), Answer Set Programming (ASP).

Background: Model Expansion In (Mitchell and Ternovska 2005), the authors formalize combinatorial search problems as the task of *model expansion (MX)*, the logical task of expanding a given (mathematical) structure with new relations. Formally, the user axiomatizes the problem in some logic $\mathcal{L}$. This axiomatization relates an instance of the problem (a *finite structure*, i.e., a universe together with some relations and functions), and its solutions (certain *expansions* of that structure with new relations or functions). Logic $\mathcal{L}$ corresponds to a specification/modelling language. It could be an extension of first-order logic such as FO(ID), or an ASP language, or a modelling language from the CP community such as ESSENCE (Frisch et al. 2008). The MX framework was later extended to infinite structures to formalise built-in arithmetic in specification languages (Ternovska and Mitchell 2009; Tasharrofi and Ternovska 2010a).

Recall that a vocabulary is a set of non-logical (predicate and function) symbols. An interpretation for a vocabulary is provided by a *structure*, which consists of a set, called the domain or universe and denoted by $dom(.)$, together with a collection of relations and (total) functions over the universe. A structure can be viewed as an *assignment* to the elements of the vocabulary. An expansion of a structure $\mathcal{A}$ is a structure $\mathcal{B}$ with the same universe, and which has all the relations and functions of $\mathcal{A}$, plus some additional relations or functions.

Formally, the task of model expansion for an arbitrary logic $\mathcal{L}$ is: Given an $\mathcal{L}$-formula ϕ with vocabulary $\sigma \cup \varepsilon$ and a structure $\mathcal{A}$ for σ find an expansion of $\mathcal{A}$, to $\sigma \cup \varepsilon$, that satisfies ϕ. Thus, we expand the structure $\mathcal{A}$ with relations and functions to interpret ε, obtaining a model $\mathcal{B}$ of ϕ.

We call σ, the vocabulary of $\mathcal{A}$, the *instance* vocabulary, and $\varepsilon := vocab(\phi) \setminus \sigma$ the *expansion* vocabulary[1]. If $\sigma = \emptyset$, we talk about *model generation*, a particular type of model expansion that is often studied.

Given a specification, we can talk about a set of $\sigma \cup \varepsilon$-structures which satisfy the specification. Alternatively, we can simply talk about a given set of $\sigma \cup \varepsilon$-structures as an MX-task, without mentioning a particular specification the structures satisfy. These sets of structures will be called *modules* later in the paper. This abstract view makes our study of modularity language-independent.

Example 1 *The following logic program ϕ constitutes an MX specification for Graph 3-colouring:*

$$\begin{array}{c} 1\{R(x), B(x), G(x)\}1 \leftarrow V(x). \\ \bot \leftarrow R(x), R(y), E(x,y). \\ \bot \leftarrow B(x), B(y), E(x,y). \\ \bot \leftarrow G(x), G(y), E(x,y). \end{array}$$

An instance is a structure for vocabulary $\sigma = \{E\}$, i.e., a graph $\mathcal{A} = \mathcal{G} = (V; E)$. The task is to find an interpretation for the symbols of the expansion vocabulary $\varepsilon = \{R, B, G\}$ such that the expansion of $\mathcal{A}$ with these is a model of ϕ:

$$\underbrace{(\overbrace{V; E}^{\mathcal{A}},\ R^{\mathcal{B}}, B^{\mathcal{B}}, G^{\mathcal{B}})}_{\mathcal{B}} \models \phi.$$

The interpretations of ε, for structures $\mathcal{B}$ that satisfy ϕ, are exactly the proper 3-colourings of $\mathcal{G}$.

The model expansion task is very common in declarative programming, – given an input, we want to generate a solution to a problem specified declaratively. This is usually done through grounding, i.e., combining instance structure $\mathcal{A}$ to a problem description ϕ thus obtaining a reduction to a low-level solver language such as SAT, ASP, SMT, etc. Model Expansion framework was introduced for systematic study of declarative languages. In particular, it connects KR with descriptive complexity (Immerman 1982). It focuses on problems, not on problem instances, it *separates instances from problem descriptions*. Using the MX framework, one can produce expressiveness and capturing results for specification languages to guarantee:

- universality of a language for a class of problems,
- feasibility of a language by bounding resources needed to solve problems in that language.

In terms of complexity, MX lies in-between model checking (MC) (a full structure is given) and satisfiability (SAT) (we are looking for a structure). Model generation ($\sigma = \emptyset$) has the same complexity as MX. The authors of (Kolokolova et al. 2010) studied the complexity of the three tasks, MC, MX and SAT, for several logics. Despite the importance of MX task in several research areas, the task has not yet been studied sufficiently, unlike the two related tasks of MC and SAT.

[1]By ":=" we mean "is by definition" or "denotes". By $vocab(\phi)$ we understand the vocabulary of ϕ.

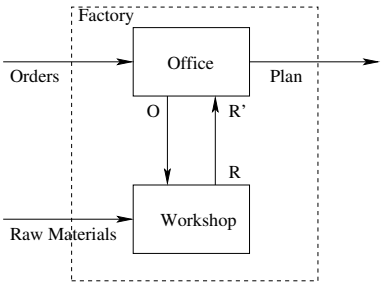


Figure 1: Modular representation of a factory

General Research Goal: Adding Modularity Given the importance of combining different languages and solvers to achieve ease of axiomatization and the best performance, our goal is to *extend the MX framework to combine modules specified in different languages.* The following example illustrates what we are aiming for.

Example 2 (Factory as Model Expansion) *In Figure 1, a part of a simple factory is represented as a modular system. Both the office and the workshop modules can be viewed as model expansion tasks. The instance vocabulary of the workshop is $\sigma = \{RawMaterials\}$ and expansion vocabulary $\varepsilon = \{R\}$. The bigger box with dashed borders is an MX task with instance vocabulary $\sigma' = \{Orders, RawMaterials\}$ and expansion vocabulary $\varepsilon' = \{Plan\}$ (the "internal" expansion symbols O and R are hidden from the outside). This task is a compound MX task whose result depends on the internal work of the office and the workshop, both of which can also have an internal structure and be represented as modular systems themselves.*

Contributions of this paper In this paper, we further develop the framework of Modular Systems. In this framework, primitive modules represent individual knowledge bases, agents, companies, etc. They can be axiomatized in a logic, be legacy systems, or be represented by a human who makes decisions. Unlike the previous work, we precisely define the notion of a well-formed modular system, and clearly separate the syntax of the algebraic language and the semantics of the algebra of modular systems. The syntax of the algebra uses a few operations, each of them (except feedback) is a counterpart of an operation in Codd's relational algebra, but over sets of structures rather than tables, and with directionality taken into account. The semantics of both primitive and compound modules is simply a set (class) of structures (an MX task). By relying on the semantics of the algebra, we then introduce its natural counterpart in logic. The logic for modular systems allows for multiple logics axiomatizing individual modules in the same formula. We expect that multi-language formalisms such as ID-logic (Denecker and Ternovska 2008) will be shown to be particular instances of this logic, and other combinations of languages will be similarly developed.

After giving the model-theoretic semantics of the algebra of modular systems, we define what it means, for a primitive module, to act as a non-deterministic operator on states of

the world represented by structures over a large vocabulary. For each expansion, there is a transition to a new structure where the interpretation of the expansion changes, and everything else moves to a new state by inertia. This definition is new and is more general than the one we introduced in the previous work. We then define the semantics of the algebraic operators by Plotkin-style structural operational semantics (Plotkin 1981). This definition also new. We then prove the equivalence of the two semantics, operational and model-theoretic. To illustrate the power of the projection operation, we show how a deterministic polytime program can be "converted" to a non-deterministic one that solves an NP-complete problem. In general, adding projection produces a jump in the computational complexity of the framework, similarly to feedback and union.

The authors of (Lierler and Truszczynski 2014) recently introduced an abstract modular inference systems formalism, and shown how propagations in solvers can be analyzed using abstract inference rules they introduced. We believe it is an important work. In this paper, we show how inference system can be lifted and integrated with our Modular Systems framework. The advantage of this integrations is that, with the help of the inference semantics, we can now go into much greater level of details of propagation processes in our abstract algorithm for solving modular systems. The inference semantics is the third semantics of modular systems mentioned in the title.

The importance of abstract study of modularity We now would like to discuss the potential implications of abstract study of modularity for KR and declarative programming.

A family of multi-language KR formalisms The Modular Systems framework gives rise to a whole new family of KR formalisms by giving the semantics to the combination of modules. This is can be viewed, for example, as a significant extension of answer set programming (ASP). In the past, combining ASP programs that were created separately from each other was only possible, under some conditions, in sequence. Now, we can combine them in a loop, use projections to hide parts of the vocabularies, etc. The previous results remain applicable. We expect, for example, that splittable programs under stable model semantics and stratifiable programs satisfy our conditions for sequential compositions of modules. Previously, in ASP, all modules had to be interpreted under one semantics (e.g. stable model semantics). Now, any model-theoretic semantics of individual modules is allowed. For example, some of the modules can be axiomatized, say, in first-order logic. That is, in particular, our proposal amounts to a "modular multi-language ASP".

Foundations in model theory We believe that classic model theory is the right abstraction tool and a good common ground for combining formalisms developed in different communities. It is sufficiently general and provides a rich machinery developed by generations of researchers. The machinery includes, for example, deep connections between expressiveness and computational complexity. In addition, the notion of a structure is important in KR as it abstractly represents our understanding of the world.

We believe that, despite common goals, the interaction between the CP community and various solver communities on one hand and the KR community is insufficient, and that foundations in model theory can make the interaction much more easy and fruitful.

Analyzing other KR systems Just as in the case of single-module system where we can use the purely semantical framework of model expansion, we can use the framework of Modular Systems to analyze multi-language KR formalisms and to study the expressive power of modular systems.

The modular framework generalizes naturally to the case where we need to study languages (logics) with "built-in" operations. In that case, embedded model expansion has to be considered, where the embedding is into an infinite structure interpreting, e.g., built-in arithmetical operations (Ternovska and Mitchell 2009; Tasharrofi and Ternovska 2010a).

Operational View Due to structural operational semantics, a new type of behaviour equivalence (bisimulation) can be defined on complex modules (e.g. represented by ASP programs). The operational view enables us to obtain results about our modular systems such as approximability of a sub-class of modular systems. While this operational view is novel and we have not developed it very much, we believe that this view allows one to apply the extensive research on proving properties of transition systems and the techniques developed in the situation calculus to prove useful facts about transition systems. We can do e.g. verification of correct behaviour, static or dynamic, particularly in the presence of arithmetic. The mathematical abstraction we proposed allows one to approach solving the problem of synthesis of modular systems abstractly, similarly to (Giacomo, Patrizi, and Sardiña 2013) Just as a Golog program can be synthesized from a library of available programs, a modular system can be synthesized from a library of available solutions to MX tasks.

Related Work Our work on modularity was initially inspired by (Järvisalo et al. 2009) who developed a constraint-based modularity formalism, where modules were represented by constraints and combined through operations of sequential composition and projection. A detailed comparison with that work is given in (Tasharrofi and Ternovska 2011).

The connections with the related formalism of Multi-Context Systems (MCSs), see (Brewka and Eiter 2007) and consequent papers, has been formally studied in (Tasharrofi 2013) and (Tasharrofi and Ternovska 2014). We only mention here that while the contexts are very general, and may have any semantics, not necessarily model-theoretic, the communication between knowledge bases happens through rules of a specific kind, that are essentially rules of logic programs with negation as failure. We, on the other hand, have chosen to represent communication simply through equality of vocabulary symbols, and to develop a model-theoretic algebra of modular systems.

Splitting results in logic programming (ASP) give conditions for separating a program into modules (Turner 1996;

1996). The results rely on a specific semantics, but can be used for separating programs into modules to represent in our formalism. The same applies to modularity of inductive definitions (Denecker and Ternovska 2008; Vennekens, Gilis, and Denecker 2006; Denecker and Ternovska 2004).

The Generate-Define-Test parts of Answer Set Programs, as discussed in (Denecker et al. 2012), are naturally representable as a sequential composition of the corresponding modules.

A recent work is (Lierler and Truszczynski 2014), where the authors introduce an abstract approach to modular inference systems and solvers was already mentioned, and is used in this paper.

The Algebra of Modular Systems

Each modular system abstractly represents an MX task, i.e., a set (or class) of structures over some instance (input) and expansion (output) vocabulary. Intuitively, a modular system is described as a set of primitive modules (individual MX tasks) combined using the operations of:

1. Projection($\pi_\nu(M)$) which restricts the vocabulary of a module. Intuitively, the projection operator on M defines a modular system that acts as M internally but where some vocabulary symbols are hidden from the outside.
2. Composition($M_1 \rhd M_2$) which connects outputs of M_1 to inputs of M_2. As its name suggests, the composition operator is intended to take two modular systems and defines a multi-step operation by serially composing M_1 and M_2.
3. Union($M_1 \cup M_2$) which, intuitively, models the case when we have two alternatives to do a task (that we can choose from).
4. Feedback($M[R = S]$) which connects output S of M to its inputs R. As the name suggests, the feedback operator models systems with feedbacks or loops. Intuitively, feedbacks represent fixpoints (not necessarily minimal) of modules viewed as operators, since they state that some outputs must be equal to some inputs.
5. Complementation($\overline{M}$) which does "the opposite" of what M does.

These operations are similar to the operations of Codd's relational algebra, but they work on sets of structures instead of relational tables. Thus, our algebra can be viewed as a higher-order counterpart of Codd's algebra, with loops. One can introduce other operations, e.g. as combinations of the ones above. The algebra of modular systems is formally defined recursively starting from primitive modules.

Definition 1 (Primitive Module) *A* primitive module M *is a model expansion task (or, equivalently, a class of structures) with distinct instance (input) vocabulary σ and expansion (output) vocabulary ε.*

A primitive module M can be given, for example, by a decision procedure D_M that decides membership in M. It can also be given by a first- or second-order formula ϕ. In this case, M is all the models of ϕ, $M = Mod(\phi)$. It could also be given by an ASP program. In this case, M would be the stable models of the program, $M = StableMod(\phi)$.

Remark 1 *A module M can be given through axiomatizing it by a formula ϕ in some logic $\mathcal{L}$ such that $vocab(\phi) = \sigma \cup \varepsilon_a \cup \varepsilon$. That is, ϕ may contain auxiliary expansion symbols that are different from the output symbols ε of M. (It may not even be possible to axiomatize M in that particular logic $\mathcal{L}$ without using any auxiliary symbols). In this case, we take $M = Mod(\phi)|_{(\sigma\cup\varepsilon)}$, the models of ϕ restricted to $\sigma \cup \varepsilon$.*

Example 3 *For example, formula ϕ of Example 1 describes the model expansion task for the problem of Graph 3-colouring. Thus, ϕ can be the representation of a module M_{col} with instance vocabulary $\{E\}$ and expansion vocabulary $\{R, G, B\}$.*

Before recursively defining our algebraic language, we have to define composable and independent modules (Järvisalo et al. 2009):

Definition 2 (Composable, Independent) *Modules M_1 and M_2 are* composable *if $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$ (no output interference). Module M_2 is* independent *from M_1 if $\sigma_{M_2} \cap \varepsilon_{M_1} = \emptyset$ (no cyclic module dependencies).*

Independence is needed for the definition of union, both properties, comparability and independence are needed for sequential composition, non-empty σ is needed for feedback.

Definition 3 (Well-Formed Modular Systems ($\mathrm{MS}(\sigma, \varepsilon)$)) *The* set of all well-formed modular systems $\mathrm{MS}(\sigma, \varepsilon)$ *for a given input, σ, and output, ε, vocabularies is defined as follows.*

Base Case, Primitive Modules: *If M is a primitive module with instance (input) vocabulary σ and expansion (output) vocabulary ε, then $M \in \mathrm{MS}(\sigma, \varepsilon)$.*

Projection *If $M \in \mathrm{MS}(\sigma, \varepsilon)$ and $\tau \subseteq \sigma \cup \varepsilon$, then $\pi_\tau(M) \in \mathrm{MS}(\sigma \cap \tau, \varepsilon \cap \tau)$.*

Sequential Composition: *If $M \in \mathrm{MS}(\sigma, \varepsilon)$, $M' \in \mathrm{MS}(\sigma', \varepsilon')$, M is composable (no output interference) with M', and M is independent from M' (no cyclic dependencies) then $(M \rhd M') \in \mathrm{MS}(\sigma \cup (\sigma' \setminus \varepsilon), \varepsilon \cup \varepsilon')$.*

Union: *If $M \in \mathrm{MS}(\sigma, \varepsilon)$, $M' \in \mathrm{MS}(\sigma', \varepsilon')$, M is independent from M', and M' is also independent from M then $(M \cup M') \in \mathrm{MS}(\sigma \cup \sigma', \varepsilon \cup \varepsilon')$.*

Feedback: *If $M \in \mathrm{MS}(\sigma, \varepsilon)$, $R \in \sigma$, $S \in \varepsilon$, and R and S are symbols of the same type and arity, then $M[R = S] \in \mathrm{MS}(\sigma \setminus \{R\}, \varepsilon \cup \{R\})$.*

Complementation: *If $M \in \mathrm{MS}(\sigma, \varepsilon)$, then $\overline{M} \in \mathrm{MS}(\sigma, \varepsilon)$.*

Nothing else is in the set $\mathrm{MS}(\sigma, \varepsilon)$.

Note that the feedback (loop) operator is not defined for the case $\sigma = \emptyset$. However, composition with a module that selects structures where interpretations of two expansion predicates are equal is always possible. The feedback operator was introduced because loops are important in information propagation, e.g. in all software systems and in solvers (e.g. ILP, ASP-CP, DPLL(T)-based) (Tasharrofi, Wu, and Ternovska 2011; 2012). Feedback operation converts an instance predicate to an expansion predicate, and equates it to another expansion predicate. Feedbacks are, in a sense, fixpoints, not necessarily minimal[2]. They add expressive power

[2]Modular systems under supported semantics (Tasharrofi 2013) allow one to focus on minimal models.

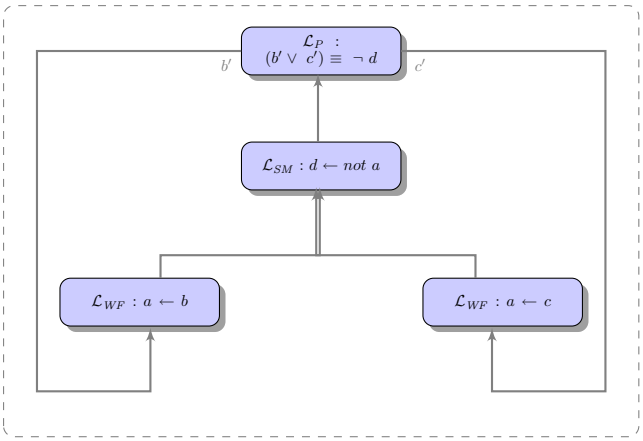


Figure 2: A simple modular system where modules are axiomatized in different languages.

to the algebra of modular systems through introducing additional non-determinism, which is not achieved by equating two expansion predicates. We discuss this issue again after the multi-language logic of modular systems is introduced.
The input-output vocabulary of module M is denoted $vocab(M)$. Modules can have "hidden" vocabulary symbols, see Remark 1.

The description of a modular system (as in Definition 3) gives an algebraic formula representing a system. *Subsystems* of a modular system M are sub-formulas of the formula that represents M. Clearly, each subsystem of a modular system is a modular system itself.

Example 4 (Simple Modular System) *Consider the following axiomatizations of modules*[3]*, each in the corresponding logic $\mathcal{L}_i$.*

$$P_{M_1} := \{\mathcal{L}_{WF} :\ a \leftarrow b\},$$
$$P_{M_2} := \{\mathcal{L}_{WF} :\ a \leftarrow c\},$$
$$P_{M_3} := \{\mathcal{L}_{SM} :\ d \leftarrow not\ a\},$$
$$P_{M_4} := \{\mathcal{L}_P :\ b' \vee c' \equiv \neg\, d\}.$$

$\mathcal{L}_{WF}$ is the logic of logic programs under the well-founded semantics, $\mathcal{L}_{SM}$ is the logic of logic programs under the stable model semantics, $\mathcal{L}_P$ is propositional logic.

The modular system in Figure 2 is represented by the following algebraic specification.

$$M := \pi_{\{a,b,c,d\}}((((M_1 \cup M_2) \rhd M_3) \rhd M_4)[c = c'][b = b']).$$

Module $M' := (((M_1 \cup M_2) \rhd M_3) \rhd M_4)$ has $\sigma_{M'} = \{b, c\}$, $\varepsilon_{M'} = \{a, b', c', d\}$. After adding feedbacks, we have $M'' := M'[c = c'][b = b']$, which turns instance symbols b and c into expansion symbols, so we have $\sigma_{M''} = \emptyset$ and $\varepsilon_{M''} = \{a, b, c, b', c', d\}$, and in addition, the interpretations of c and c', and b and b' must coincide. Finally, projection hides c' and b'.

Module M corresponds to the whole modular system denoted by the box with dotted borders. Its input-output vocabularies are as follows: $\sigma_M = \emptyset$, $\varepsilon_M = \{a, b, c, d\}$, b' and c' are "hidden" from the outside. They are auxiliary expansion symbols, see Remark 1.

Modules $(M_1 \cup M_2)$ and M_3 in this example are composable (no output interference) and independent (no cyclic dependencies), M_1 and M_2 are independent.

The paper (Tasharrofi and Ternovska 2011) contains a more applied example, of a business process planner, where each module represents a business partner.

Multi-Language Logic of Modular Systems It is possible to introduce a multi-language logic of modular systems, where formulas of different languages are combined using conjunctions[4] (standing for $\rhd$), disjunctions ($\cup$), existential second-order quantification (π_ν), etc. For example, model expansion for the following formula

$$\phi_M := \exists b' \exists c' ((((\{\mathcal{L}_{WF} :\ a \leftarrow b\} \vee \{\mathcal{L}_{WF} :\ a \leftarrow c\}) \\ \wedge \{\mathcal{L}_{SM} :\ d \leftarrow not\ a\} \wedge \{\mathcal{L}_P :\ d \leftarrow not\ a\}) \\ \wedge [b = b' \wedge c = c'].$$

with $\sigma_M = \emptyset$ and $\varepsilon = \{a, b, c, d\}$ and "hidden" (auxilliary, see Remark 1) vocabulary $\varepsilon_a = \{b', c'\}$ corresponds to the modular system in Figure 2 from Example 4.

Feedback is a meta-logic operation that does not have a counterpart among logic connectives. Feedback does not exist for model generation ($\sigma = \emptyset$) and increases the number of symbols in the expansion vocabulary. In our example, former instance symbols (b and c in this case) become expansion symbols, and become equal to the outputs b' and c' thus forming loops.

Note also that projections (thus quantifiers) over variables ranging over domain objects can be achieved if such variables are considered to be a part of the vocabularies of modules. In this logic, the full version of ID-logic, for example, would correspond to the case without feedbacks and all modules limited to either those axiomatized in first-order logic or definitions under well-founded semantics. A formal study of such a multi-language logic in connection with existing KR formalisms (such as, e.g. ID-logic, combinations such as ASP and Description logic. etc.) is left as a future research direction.

Note that if all modules are axiomatized in second-order logic, our task is just model expansion for classic second-order logic that is naturally expressible by adding existential second-order quantifiers at the front. If there are multiple languages, we can talk about the *complexity of model expansion for the combined formula (or modular system) as a function of the expressiveness of the individual languages*, which is a study of practical importance.

Model-Theoretic Semantics

So far, we introduced the syntax of the algebraic language using the notion of a well-formed modular system. Those

[3] In realistic examples, module axiomatizations are much more complex and contain multiple rules or axioms.

[4] It will be clear from the semantics that the operation $\rhd$ is commutative.

are primitive modules (that are sets of structures) or are constructed inductively by the algebraic operations of composition, union, projection, loop. Model-theoretic semantics associates, with each modular system, a set of structures. Each such structure is called a *model* of that modular system. Let us assume that the domains of all modules are included in a (potentially infinite) universal domain U.

Definition 4 (Models of a Modular System) *Let $M \in \mathrm{MS}(\sigma, \varepsilon)$ be a modular system and $\mathcal{B}$ be a $(\sigma \cup \varepsilon)$-structure. We construct the set $M^{mt} = Mod(M)$ of models of module M under model-theoretic semantics recursively, by structural induction on the structure of a module.*

Base Case, Primitive Module: *$\mathcal{B}$ is a model of M if $\mathcal{B} \in M$.*

Projection: *$\mathcal{B}$ is a model of $M := \pi_{(\sigma\cup\varepsilon)}(M')$ (with $M' \in \mathrm{MS}(\sigma', \varepsilon')$) if a $(\sigma' \cup \varepsilon')$-structure $\mathcal{B}'$ exists such that $\mathcal{B}'$ is a model of M' and $\mathcal{B}'$ expands $\mathcal{B}$.*

Composition: *$\mathcal{B}$ is a model of $M := M_1 \rhd M_2$ (with $M_1 \in \mathrm{MS}(\sigma_1, \varepsilon_1)$ and $M_2 \in \mathrm{MS}(\sigma_2, \varepsilon_2)$) if $\mathcal{B}|_{(\sigma_1\cup\varepsilon_1)}$ is a model of M_1 and $\mathcal{B}|_{(\sigma_2\cup\varepsilon_2)}$ is a model of M_2.*

Union: *$\mathcal{B}$ is a model of $M := M_1 \cup M_2$ (with $M_1 \in \mathrm{MS}(\sigma_1, \varepsilon_1)$ and $M_2 \in \mathrm{MS}(\sigma_2, \varepsilon_2)$) if either $\mathcal{B}|_{(\sigma_1\cup\varepsilon_1)}$ is a model of M_1, or $\mathcal{B}|_{(\sigma_2\cup\varepsilon_2)}$ is a model of M_2.*

Feedback: *$\mathcal{B}$ is a model of $M := M'[R = S]$ (with $M' \in \mathrm{MS}(\sigma', \varepsilon')$) if $R^{\mathcal{B}} = S^{\mathcal{B}}$ and $\mathcal{B}$ is model of M'.*

Complementation: *$\mathcal{B}$ is a model of $M := \overline{M'}$ (with $M, M' \in \mathrm{MS}(\sigma, \varepsilon)$) if and $\mathcal{B}$ is not a model of M'. That is, $\overline{M'}$ denotes the complement of M in the set of all possible $\sigma \cup \varepsilon$-structures over the universal domain U.*

Nothing else is a model of M.

Note that, by this semantics, sequential composition is a commutative operation (we could have used $\bowtie$ notation), however the direction of information propagation is uniquely given by the separations of the input and output vocabularies. Notice that it's not possible to compose two modules in two different ways. If it was possible, then in the compound module we would had that the intersection of the input and the output vocabularies would not be empty, and this is not allowed. So, we prefer to use $\rhd$ instead of $\bowtie$ for both historic and mnemonic reasons, and encourage the reader to write algebraic formulas in a way that corresponds to their visualizations of the corresponding modular systems.

An example illustrating the semantics of the feedback operator, as well as non-determinism introduced by this operator is given in the appendix.

The task of model expansion for modular system M takes a σ-structure $\mathcal{A}$ and finds (or reports that none exists) a $(\sigma \cup \varepsilon)$-structure $\mathcal{B}$ that expands $\mathcal{A}$ and is a model of M. Such a structure $\mathcal{B}$ is a *solution of M for input $\mathcal{A}$*.

Remark 2 *The semantics does not put any finiteness restriction on the domains of structures. Thus, the framework works for modules with infinite structures.*

Structural Operational Semantics

In this section, we introduce a novel Structural Operational Semantics of modular systems.

We now focus on potentially infinite all-inclusive vocabulary τ that subsumes the vocabularies of all modules considered. Thus, we always have $vocab(M) \subseteq \tau$.

Definition 5 (State of a Modular Systems) *A τ-state of a modular system $M \in \mathrm{MS}(\sigma, \varepsilon)$ is a τ-structure such that $(\sigma \cup \varepsilon) \subseteq \tau$.*

The semantics we give is structural because, for example, the meaning of the sequential composition, $M_1 \rhd M_2$, is defined through the meaning of M_1 and the meaning of M_2.

Definition 6 (Modules as Operators) *We say that a well-formed modular system M (non-deterministically) maps τ-state $\mathcal{B}_1$ to τ-state $\mathcal{B}_2$, notation $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$, if we can apply the rules of the structural operational semantics (below) starting from this expression and arriving to $true$. In that case, we say that transition $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$ is* derivable. **Primitive modules** *M:*

$$\frac{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{true} \text{ if } \mathcal{B}_2|_{(\sigma\cup\varepsilon)} \in M \text{ and } \mathcal{B}_2|_{(\tau\setminus\varepsilon)} = \mathcal{B}_1|_{(\tau\setminus\varepsilon)}.$$

We proceed by induction on the structure of modular system M. **Projection** *$\pi_\nu(M)$:*

$$\frac{(\pi_\nu(M), \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M, \mathcal{B}_1') \longrightarrow \mathcal{B}_2'} \text{ if } \mathcal{B}_1'|_\nu = \mathcal{B}_1|_\nu \text{ and } \mathcal{B}_2'|_\nu = \mathcal{B}_2|_\nu.$$

Composition *$M_1 \rhd M_2$:*

$$\frac{(M_1 \rhd M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_1, \mathcal{B}_1) \longrightarrow \mathcal{B}' \text{ and } (M_2, \mathcal{B}') \longrightarrow \mathcal{B}_2}.$$

Union *$M_1 \cup M_2$:*

$$\frac{(M_1 \cup M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_1, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}, \qquad \frac{(M_1 \cup M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}.$$

Feedback *$M[R = S]$:*

$$\frac{(M[R = S], \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}, \text{ if } R^{\mathcal{B}_1} = S^{\mathcal{B}_2}.$$

Complementation *$\overline{M}$:*

$$\frac{(\overline{M}, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{true} \text{ if } (M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2 \text{ is not derivable}.$$

Nothing else is derivable.

Let us clarify the projection operation $\pi_\nu(M)$. Let $vocab(M) = \sigma' \cup \varepsilon'$, let $\nu = \sigma \cup \varepsilon$, $\sigma \subseteq \sigma'$, $\varepsilon \subseteq \varepsilon'$. Module $\pi_\nu(M)$, viewed as an operator, is applied to τ-structure $\mathcal{B}_1$. It (a) expands σ-part of $\mathcal{B}_1$ to σ' by an arbitrary interpretation over the same domain, and then (b) applies M to the modified input, (c) projects the result of application of M onto ε, ignoring everything else, (d) the interpretations of $\tau \setminus \epsilon$ are moved from $\mathcal{B}_1$ by inertia.

Definition 7 (Operational Semantics) *Let M be a well-formed modular system in $\mathrm{MS}(\sigma, \varepsilon)$. The semantics of M is given by the following set.*

$$M^{op} := \{\mathcal{B} \mid (\mathcal{B}_1, M) \longrightarrow \mathcal{B}_2 \text{ and } \mathcal{B}|_\sigma = \mathcal{B}_1|_\sigma,\ \mathcal{B}|_\varepsilon = \mathcal{B}_2|_\varepsilon\}.$$

Figure 3 illustrates this definition.

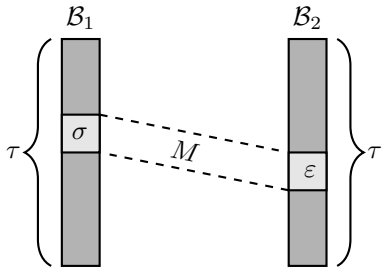


Figure 3: An illustration of Definition 7. Module $M \in \mathrm{MS}(\sigma, \varepsilon)$ maps a τ-structure $\mathcal{B}_1$ (with $(\sigma \cup \varepsilon) \subseteq \tau$) to a τ-structure $\mathcal{B}_2$ by changing the interpretation ε according to M (so that the σ part and the new ε part, together, form a model of M). Interpretation of all other symbols, including those in σ, stays the same. This is similar to how frame axioms keep fluents that are not affected by actions unchanged in the situation calculus.

Corollary 1 *Every result of application of M is its fixpoint. That is, for any τ-states $\mathcal{B}_1$, $\mathcal{B}_2$, if $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$, then $(M, \mathcal{B}_2) \longrightarrow \mathcal{B}_2$.*

Proof: By Definition 7, because of inertia, the interpretation of σ is transferred from $\mathcal{B}_1$ to $\mathcal{B}_2$. Since the interpretation of ε is already changed by M, nothing is to be changed, and $(M, \mathcal{B}_2) \longrightarrow \mathcal{B}_2$. ∎

Theorem 1 (Operational = Model-theoretic Semantics) *Let M be a well-formed modular system in $\mathrm{MS}(\sigma, \varepsilon)$. Then, its model-theoretic and operational semantics coincide,*

$$M^{mt} = M^{op}.$$

The most important consequence of this theorem is that all the results obtained when modules are viewed as operators, still hold when modules are viewed as sets of structures (and vice versa). Thus, we may use either of these semantics. From now on, by M we mean either one of these sets M^{mt} or M^{op}.

Proof: We prove the statement inductively.

Base case, primitive module By definition, model-theoretically, $\mathcal{B}$ is a model of M if $\mathcal{B} \in M$. On the other hand, operationaly,

$$M^{op} := \{\mathcal{B} \mid (\mathcal{B}_1, M) \longrightarrow \mathcal{B}_2 \text{ and } \mathcal{B}|_\sigma = \mathcal{B}_1|_\sigma,\ \mathcal{B}|_\varepsilon = \mathcal{B}_2|_\varepsilon\},$$

where

$$\frac{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{true} \text{ if } \mathcal{B}_2|_{(\sigma \cup \varepsilon)} \in M \text{ and } \mathcal{B}_2|_{(\tau \setminus \varepsilon)} = \mathcal{B}_1|_{(\tau \setminus \varepsilon)}.$$

Thus, $\mathcal{B} \in M$, and the two semantics coincide for primitive modules.

Our inductive hypothesis is that the statement of the theorem holds for M_1, M_2 and M'. We proceed inductively.

Projection $M := \pi_\nu(M')$. By the hypothesis, $(M')^{mt} = (M')^{op}$, where $(M')^{op}$ is constructed "from pieces", $(M')^{op} := \{\mathcal{B}' \mid (\mathcal{B}'_1, M') \longrightarrow \mathcal{B}'_2 \text{ and } \mathcal{B}|_\sigma = \mathcal{B}'_1|_\sigma,\ \mathcal{B}|_\varepsilon = \mathcal{B}'_2|_\varepsilon\}$. We apply the rule

$$\frac{(\pi_\nu(M'), \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M', \mathcal{B}'_1) \longrightarrow \mathcal{B}'_2} \text{ if } \mathcal{B}'_1|_\nu = \mathcal{B}_1|_\nu \text{ and } \mathcal{B}'_2|_\nu = \mathcal{B}_2|_\nu$$

and obtain that $(\pi_\nu(M'), \mathcal{B}_1) \longrightarrow \mathcal{B}_2$ where $\mathcal{B}_1$ and $\mathcal{B}_2$ are just like $\mathcal{B}'_1$ and $\mathcal{B}'_2$ on the vocabulary ν. Now, $M := \pi_\nu(M')$ is constructed "from σ and ε pieces" of $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively (where $\nu = \sigma \cup \varepsilon$):

$$M^{op} := \{\mathcal{B} \mid (\mathcal{B}_1, M) \longrightarrow \mathcal{B}_2 \text{ and } \mathcal{B}|_\sigma = \mathcal{B}_1|_\sigma,\ \mathcal{B}|_\varepsilon = \mathcal{B}_2|_\varepsilon\},$$

On the other hand, model-theoretically, $\mathcal{B}$ is a model of $M := \pi_{(\sigma \cup \varepsilon)}(M')$ (with $M' \in \mathrm{MS}(\sigma', \varepsilon')$) if a $(\sigma' \cup \varepsilon')$-structure $\mathcal{B}'$ exists such that $\mathcal{B}'$ is a model of M' and $\mathcal{B}'$ expands $\mathcal{B}$, which makes the two semantics equal for projection, $(M)^{mt} = (M)^{op}$.

We omit the proofs for the other inductive cases. ∎

Applications of Operational View We now discuss how the operational semantics can be used. For example, we can consider modular systems at various levels of granularity. We might be interested in the following question: if M gives a transition from a structure $\mathcal{B}$ to structures $\mathcal{B}'$, then what are the transitions given by the subsystems of M? While answering this question in its full generality is algorithmically impossible, we may study the question of whether a particular transition by a subsystem exists. To answer it, one has to start from the system and build down to the subsystem using the rules of the structural operation semantics. Reasoning about subsystems of a modular system can be useful in business process modelling. Suppose a particular transition should hold for the entire process. This might be the global task of an organization. In order to make that transition, the subsystems have to perform their own transitions. Those transitions are derivable using the rules of structural operational semantics.

Complexity In the following proposition, we assume a standard encoding of structures as binary strings) as is common in Descriptive complexity (Immerman 1982). Note that if M is deterministic, it is polytime in the size of the encoding of the input structure. This is because the domain remains the same, the arities of the relations in ε are fixed, so we need (n^k) steps to construct new interpretations of ε, and move the remaining relations.

Proposition 1 *Let M be a module that performs a (deterministic) polytime computation. Projection $\pi_\nu(M)$ increases the complexity of M from P to NP. More generally, for an operator M on the k-th level of the Polynomial Time hierarchy (PH), projection can increase the complexity of M from Δ^P_k to Σ^P_{k+1}.*

Proof: We will show the property for the jump from P to NP, for illustration. The proof generalizes to all levels of PH. Let M takes an instance of an NP-complete problem, such as a graph in 3-Colourability, encoded in σ_G, and what it means to be 3-Colourable, as a formula encoded in the interpretation of σ_ϕ, and returns an instance of SAT encoded in ε, a CNF formula that is satisfiable if and only if the graph is 3-Colourable, and a yes/no answer bit represented by ε_{answer}. Thus, M performs a deterministic (thus, polytime) reduction. Consider $\pi_\nu(M)$, where $\nu = \sigma_G \cup \varepsilon_{answer}$. This module takes a graph and returns a yes or no answer depending on whether the graph is 3-colourable. Thus, $\pi_\nu(M)$ solves an NP-complete problem. ∎

Union and feedback change the complexity as well.

Inference Semantics of Modular Systems

In modular systems, each agent or a knowledge base can have its own way of reasoning, that can be formulated through inferences or propagations. To define inferential semantics for modular systems, we closely follow (Lierler and Truszczynski 2014). Since input/output is not considered by the authors, their case corresponds to the instance vocabulary being empty, $\sigma = \emptyset$, i.e., model generation, and can be viewed as an analysis of the after-grounding faze. Since we want to separate problem descriptions and their instances (and reuse problem descriptions), as well as to define additional algebraic operations (the authors consider conjunctions only), we need to allow $\sigma \neq \emptyset$, and present inferences on partial structures. This is not hard however.

We start by assuming that there is a constant for every element of the domains. We view structures as sets of ground atoms. We now closely follow and generalize the definitions of (Lierler and Truszczynski 2014) from sets of propositional atoms to first-order structures, to establish a connection to the Modular Systems framework presented above. The propositional case then corresponds to structures over the domain $\{\langle\,\rangle\}$ containing the empty tuple that interprets propositional symbols that are true.

Let a fixed countably infinite set of ground atoms τ be given. We use $Lit(\tau)$ to denote the set of all literals over τ. For $S \subseteq Lit(\tau)$:

$S^+ := \tau \cap S$

$S^- := \{a \in \tau \mid \neg a \in S\}$

$l \in Lit(\tau)$ is unassigned in S if $l \notin M$ and $\bar{l} \notin S$

S is consistent if $S^+ \cap S^- \neq \emptyset$

Let $C(\tau)$ be all consistent subsets of $Lit(\tau)$.

Definition 8 (Abstract Inference Representation of M) *An* abstract inference representation M^i of module M *over a vocabulary τ is a finite set of pairs of the form (S, l), where $S \in C(\tau)$, $l \in Lit(\tau)$, and $l \notin Lit(\tau)$. Such pairs are called* inferences of the module M.

In the exposition below, we view structures as sets of propositional atoms, $\mathcal{B} \subseteq \tau$.

S is consistent with $\mathcal{B} \subseteq \tau$ if $S^+ \subseteq \mathcal{B}$ and $S^- \cap \mathcal{B} = \emptyset$. Literal l is consistent with $\mathcal{B} \subseteq \tau$ if $\{l\}$ is consistent with $\mathcal{B}$.

Definition 9 (Primitive Module, Inferential Semantics) *A primitive module $M \in MS(\sigma, \varepsilon)$ is a set of $(\sigma \cup \varepsilon)$-structures $\mathcal{B}$ such that for every inference $(S, l) \in M^i$ such as S is consistent with $\mathcal{B}$, l is consistent with $\mathcal{B}$, too.*

Thus, primitive modules, even when they are represented through abstract inferences, are sets of structures as before, and the definitions of the algebraic operations do not need to be changed.

The inference framework can be viewed as yet another (very useful) way of representing modules. Since the inference framework is abstract, we cannot prove a correspondence between a given individual module presented as a set of structures or as an operator on one hand and as an inferential representation on the other in general, without specifying what inference mechanism is used. However, we can do it for particular cases such as $Ent(T)$ (Lierler and Truszczynski 2014), which is left for a future paper.

With the inference semantics as described, we can now model problems (sets of instances) rather than single instances as a combination of other problems. This semantics allows one to study the details of propagation of information in the process of constructing solutions to modular systems, through incremental construction of partial structures as in (Tasharrofi, Wu, and Ternovska 2011; 2012), but in more detail. This direction is left for future research.

Conclusion and Future Directions

We described a modular system framework, where primitive and compound modules are sets (classes) of structures, and combinations of modules are achieved by applying algebraic operations that are a higher-order counterpart of Codd's relational algebra operations. An additional operation is the feedback operator that connects output symbols with the input ones and is used to model information propagation such as loops of software systems and solvers.

We defined two novel semantics of modular systems, operational and inferential, that are equivalent to the original model-theoretic semantics (Tasharrofi and Ternovska 2011). We presented a multi-language logic, a syntactic counterpart of the algebra of modular systems. Minimal models of modular systems are introduced in a separate paper on supported modular systems, see also (Tasharrofi 2013).

The framework of modular systems gives us, through its semantic-based approach, a unifying perspective on multi-language formalisms and solvers. More importantly, it gives rise to a whole new family of multi-language KR formalisms, where new formalisms can be obtained by instantiating specific logics defining individual modules.

The framework can be used for analysis of existing KR languages. In particular, expressiveness and complexity results for combined formalisms can be obtained in a way similar to the previous work (Mitchell and Ternovska 2008; Tasharrofi and Ternovska 2010b; 2010a) where single-module embedded model expansion was used.

References

Brewka, G., and Eiter, T. 2007. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07) - Volume 1*, 385–390. AAAI Press.

Denecker, M., and Ternovska, E. 2004. Inductive situation calculus. In *Proc., KR-04*.

Denecker, M., and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM transactions on computational logic (TOCL)* 9(2):1–51.

Denecker, M.; Lierler, Y.; Truszczynski, M.; and Vennekens, J. 2012. A tarskian informal semantics for answer set programming. In Dovier, A., and Costa, V. S., eds., *ICLP (Technical Communications)*, volume 17 of *LIPIcs*, 277–289. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Frisch, A. M.; Harvey, W.; Jefferson, C.; Martínez-Hernández, B.; and Miguel, I. 2008. Essence: A constraint language for specifying combinatorial problems. *Constraints* 13:268–306.

Giacomo, G. D.; Patrizi, F.; and Sardiña, S. 2013. Automatic behavior composition synthesis. *Artif. Intell.* 196:106–142.

Immerman, N. 1982. Relational queries computable in polynomial time. In *STOC '82: Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, 147–152.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A module-based framework for multi-language constraint modeling. In *Proceedings of the 10th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Computer Science (LNCS)*, 155–168. Springer-Verlag.

Kolokolova, A.; Liu, Y.; Mitchell, D.; and Ternovska, E. 2010. On the complexity of model expansion. In *Proc., 17th Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, 447–458. Springer. LNCS 6397.

Lierler, Y., and Truszczynski, M. 2014. Abstract modular inference systems and solvers. In *Proceedings of the 16th International Symposium on Practical Aspects of Declarative Languages (PADL'14)*.

Mitchell, D. G., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *Proc. AAAI*, 430–435.

Mitchell, D. G., and Ternovska, E. 2008. Expressiveness and abstraction in ESSENCE. *Constraints* 13(2):343–384.

Plotkin, G. 1981. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University. Also published in: Journal of Logic and Algebraic Programming, 60-61:17-140, 2004.

Tasharrofi, S., and Ternovska, E. 2010a. PBINT, a logic for modelling search problems involving arithmetic. In *Proceedings of the 17th Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'17)*. Springer. LNCS 6397.

Tasharrofi, S., and Ternovska, E. 2010b. Built-in arithmetic in knowledge representation languages. In *NonMon at 30 (Thirty Years of Nonmonotonic Reasoning)*.

Tasharrofi, S., and Ternovska, E. 2011. A semantic account for modularity in multi-language modelling of search problems. In *Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS)*, 259–274.

Tasharrofi, S., and Ternovska, E. 2014. Generalized multi-context systems. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR2014)*.

Tasharrofi, S.; Wu, X. N.; and Ternovska, E. 2011. Solving modular model expansion tasks. In *Proceedings of the 25th International Workshop on Logic Programming (WLP'11)*, volume abs/1109.0583. Computing Research Repository (CoRR).

Tasharrofi, S.; Wu, X. N.; and Ternovska, E. 2012. Solving modular model expansion: Case studies. In *Postproceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management and 25th Workshop on Logic Programming*, 175–187. Lecture Notes in Artificial Intelligence (LNAI).

Tasharrofi, S. 2013. *Solving Model Expansion Tasks: System Design and Modularity*. Ph.D. Dissertation, Simon Fraser University, Burnaby, BC, Canada.

Ternovska, E., and Mitchell, D. G. 2009. Declarative programming of search problems with built-in arithmetic. In *Proc. of IJCAI*, 942–947.

Turner, H. 1996. Splitting a default theory. In *Proceedings of the 13th National Conference on Artificial intelligence (AAAI'96) - Volume 1*, 645–651. AAAI Press.

Vennekens, J.; Gilis, D.; and Denecker, M. 2006. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic* 7(4):765–797.

Appendix

Example 5 *We illustrate models of a simple modular system with feedback operator. Consider the following axiomatization P_{M_0} of a primitive module M_0, where $\sigma_{M_0} = \{i\}$ and $\varepsilon_{M_0} = \{a, b\}$.*

$$P_{M_0} := \left\{ \mathcal{L}_{SM} : \begin{array}{l} a \leftarrow i, not\ b, \\ b \leftarrow i, not\ a. \end{array} \right\}$$

We will demonstrate how the set of models of this program changes when we use the feedback operator. When the input i is true (given by the corresponding instance structure), then

$$StableMod(P_{M_0},\ i = true) = \{\{a\}, \{b\}\}.$$

When i is false, there is one model, where everything is false,

$$StableMod(P_{M_0},\ i = false) = \{\emptyset\}.$$

Module M_0 is the set of structures for the entire $\sigma_{M_0} \cup \varepsilon_{M_0}$ vocabulary. Since we are dealing with a propositional case, each structure is represented by a set of atoms that are true in that structure.

$$M_0 = \{\{i, a\}, \{i, b\}, \emptyset\}.$$

Now consider a different module, M_1, with $\sigma_{M_1} = \{i, a, b\}$ and $\varepsilon_{M_1} = \{a', b'\}$, axiomatized by

$$P_{M_1} := \left\{ \mathcal{L}_{SM} : \begin{array}{l} a' \leftarrow i, not\ b, \\ b' \leftarrow i, not\ a. \end{array} \right\}$$

This modular system is deterministic, – for each input (each of the eight possible interpretations of i, a and b), there is at most one model.

i	a	b	*Models of* M_1
$\bot$	$\bot$	$\bot$	$\{\emptyset\}$
$\bot$	$\top$	$\bot$	$\{\emptyset\}$
$\bot$	$\bot$	$\top$	$\{\emptyset\}$
$\bot$	$\top$	$\top$	$\{\emptyset\}$
$\top$	$\bot$	$\bot$	$\{\{i, a', b'\}\}$
$\top$	$\top$	$\bot$	$\{\{i, a, a'\}\}$
$\top$	$\bot$	$\top$	$\{\{i, b, b'\}\}$
$\top$	$\top$	$\top$	$\{\{i, a, b\}\}$

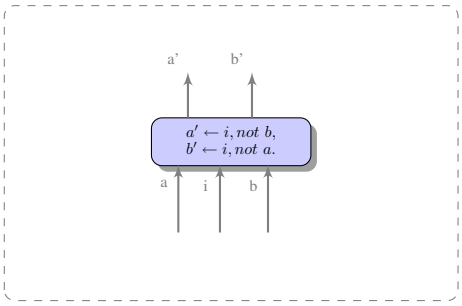


Figure 4: Module M_1.

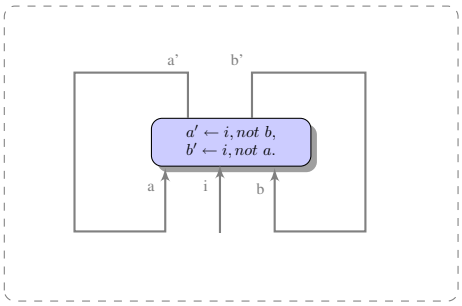


Figure 5: Module M_2.

Thus, we have:

$$M_1 = \{\emptyset, \{i, a', b'\}, \{i, a, a'\}, \{i, b, b'\}, \{i, a, b\}\}.$$

If we add feedback, we obtain the following system $M_2 = M_1[a = a'][b = b']$. *Its input is* i, *all other symbols are in the expansion vocabulary. The models are:*

i	*Models of* M_2
$\bot$	$\{\emptyset\}$
$\bot$	$\{\emptyset\}$
$\top$	$\{\{i, a, a'\}\}$
$\top$	$\{\{i, b, b'\}\}$

$$M_2 = M_1[a = a'][b = b'] = \{\emptyset, \{i, a, a'\}, \{i, b, b'\}\}.$$

As we see here, after adding feedback, for the same input i, *we obtain two different models. Thus, by means of feedback, a deterministic system* M_1 *was turned into a non-deterministic system* M_2.

This modular system is deterministic, – for each input (each of the eight possible interpretations of i, a *and* b*), there is at most one model. Notice also that*

$$\pi_{\{i,a,b\}}(M_1[a = a'][b = b']) = M_0.$$

Generalizing Modular Logic Programs *

João Moura and Carlos Viegas Damásio
CENTRIA - Centre for Artificial Intelligence
Universidade Nova de Lisboa, Portugal

Abstract

Even though modularity has been studied extensively in conventional logic programming, there are few approaches on how to incorporate modularity into Answer Set Programming, a prominent rule-based declarative programming paradigm. A major approach is Oikarinnen and Janhunen's Gaifman-Shapiro-style architecture of program modules, which provides the composition of program modules. Their module theorem properly strengthens Lifschitz and Turner's splitting set theorem for normal logic programs. However, this approach is limited by module conditions that are imposed in order to ensure the compatibility of their module system with the stable model semantics, namely forcing output signatures of composing modules to be disjoint and disallowing positive cyclic dependencies between different modules. These conditions turn out to be too restrictive in practice and in this paper we discuss alternative ways of lift both restrictions independently, effectively solving the first, widening the applicability of this framework and the scope of the module theorem.

1 Introduction

Over the last few years, answer set programming (ASP) (Eiter et al. 2001; Baral 2003; Lifschitz 2002; Marek and Truszczynski 1999; Niemelä 1998) emerged as one of the most important methods for declarative knowledge representation and reasoning. Despite its declarative nature, developing ASP programs resembles conventional programming: one often writes a series of gradually improving programs for solving a particular problem, e.g., optimizing execution time and space. Until recently, ASP programs were considered as integral entities, which becomes problematic as programs become more complex, and their instances grow. Even though modularity is extensively studied in logic programming, there are only a few approaches on how to incorporate it into ASP (Gaifman and Shapiro 1989; Oikarinen and Janhunen 2008; Dao-Tran et al. 2009; Babb and Lee 2012) or other module-based constraint modeling frameworks (Järvisalo et al. 2009; Tasharrofi and Ternovska 2011). The research on modular systems of logic program has followed two main-streams (Bugliesi, Lamma, and Mello 1994). One is programming in-the-large where compositional operators are defined in order to combine different modules, e.g., (Mancarella and Pedreschi 1988; Gaifman and Shapiro 1989; O'Keefe 1985). These operators allow combining programs algebraically, which does not require an extension of the theory of logic programs. The other direction is programming-in-the-small, e.g., (Giordano and Martelli 1994; Miller 1986), aiming at enhancing logic programming with scoping and abstraction mechanisms available in other programming paradigms. This approach requires the introduction of new logical connectives in an extended logical language. The two mainstreams are thus quite divergent.

The approach of (Oikarinen and Janhunen 2008) defines modules as structures specified by a program (knowledge rules) and by an interface defined by input and output atoms which for a single module are, naturally, disjoint. The authors also provide a module theorem capturing the compositionality of their module composition operator. However, two conditions are imposed: there cannot be positive cyclic dependencies between modules and there cannot be common output atoms in the modules being combined. Both introduce serious limitations, particularly in applications requiring integration of knowledge from different sources. The techniques used in (Dao-Tran et al. 2009) for handling positive cycles among modules are shown not to be adaptable for the setting of (Oikarinen and Janhunen 2008).

In this paper we discuss two alternative solutions to the common outputs problem, generalizing the module theorem by allowing common output atoms in the interfaces of the modules being composed. A use case for this requirement can be found in the following example.

Example 1 *Alice wants to buy a car, wanting it to be safe and not expensive; she preselected 3 cars, namely c_1, c_2 and c_3. Her friend Bob says that car c_2 is expensive, while Charlie says that car c_3 is expensive. Meanwhile, she consulted two car magazines reviewing all three cars. The first considered c_1 safe and the second considered c_1 to be safe while saying that c_3 may be safe. Alice is very picky regarding safety, and so she seeks some kind of agreement between the reviews.*

*The work of João Moura was supported by grant SFRH/BD/69006/2010 from Fundação para a Ciência e Tecnologia (FCT) from the Portuguese Ministério do Ensino e da Ciência. Research also supported by FCT funded project ERRO: Efficient reasoning with rules and ontologies (ref. PTDC/EIA-CCO/121823/2010).

The described situation can be captured with five modules, one for Alice, other three for her friends, and another for each magazine. Alice should conclude that c_1 is safe since both magazines agree on this. Therefore, one would expect Alice to opt for car c_1 since it is not expensive, and it is reviewed as being safe. However, the current state-of-the-art does not provide any way of combining these modules since they share common output atoms. ■

In summary, the fundamental results of (Oikarinen and Janhunen 2008) require a syntactic operation to combine modules – basically corresponding to the union of programs –, and a compositional semantic operation joining the models of the modules. The module theorem states that the models of the combined modules can be obtained by applying the semantics of the natural join operation to the original models of the modules – which is compositional.

The authors show however that allowing common outputs destroys this property. There are two alternatives to pursue:

(1) Keep the syntactic operation: use the union of programs to syntactically combine modules, plus some bookkeeping of the interface, and thus the semantic operation on models has to be changed;

(2) Keep the semantic operation: the semantic operation is the natural join of models, and thus a new syntactic operation is required to guarantee compositionality.

Both will be explored in this paper as they correspond to different and sensible ways of combining two sources of information, already identified in Example 1: the first alternative is necessary for Alice to determine if a car is expensive; the second alternative captures the way Alice determines whether a car is safe or not. Keeping the syntactic operation is shown to be impossible since models do not convey enough information to obtain compositionality. We present a solution to this problem based on a transformation that introduces the required extra information. The second solution is possible, and builds on the previous module transformation.

This paper proceeds in Section 2 with an overview of the modular logic programming paradigm, identifying some of its shortcomings. In Section 3 we discuss alternative methods for lifting the restriction that disallows positive cyclic dependencies, and in Section 4 introduce two new forms of composing modules allowing common outputs, one keeping the original syntactic $union$ operator and the other keeping the original semantic model $join$ operator. We finish with conclusions and a general discussion.

2 Modularity in Answer Set Programming

Modular aspects of Answer Set Programming have been clarified in recent years, with authors describing how and when two program parts (modules) can be composed (Oikarinen and Janhunen 2008; Dao-Tran et al. 2009; Järvisalo et al. 2009) under the stable model semantics. In this paper, we will make use of Oikarinen and Janhunen's logic program modules defined in analogy to (Gaifman and Shapiro 1989) which we review after presenting the syntax of answer set programs.

2.1 Answer set programming paradigm

Logic programs in the answer set programming paradigm are formed by finite sets of rules r having the following syntax:

$$L_1 \leftarrow L_2, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n. \quad (1)$$

$(n \geq m \geq 0)$ where each L_i is a logical atom without the occurrence of function symbols – arguments are either variables or constants of the logical alphabet.

Considering a rule of the form **(1)**, let $Head_P(r) = L_1$ be the literal in the head, $Body_P^+(r) = \{L_2, \ldots, L_m\}$ be the set with all positive literals in the body, $Body_P^-(r) = \{L_{m+1}, \ldots, L_n\}$ be the set containing all negative literals in the body, and $Body_P(r) = \{L_2, \ldots, L_n\}$ be the set containing all literals in the body. If a program is positive we will omit the superscript in $Body_P^+(r)$. Also, if the context is clear we will omit the subscript mentioning the program and write simply $Head(r)$ and $Body(r)$ as well as the argument mentioning the rule.

The semantics of stable models is defined via the reduct operation (Gelfond and Lifschitz 1988). Given an interpretation M (a set of ground atoms), the reduct P^M of a program P with respect to M is the program

$$\{Head(r) \leftarrow Body^+(r) \mid r \in P, Body^-(r) \cap M = \emptyset\}.$$

The interpretation M is a stable model of P iff $M = LM(P^M)$, where $LM(P^M)$ is the least model of program P^M.

The syntax of logic programs has been extended with other constructs, namely weighted and choice rules (Niemelä 1998). In particular, choice rules have the following form, for $(n \geq 1)$:

$$\{A_1, \ldots, A_n\} \leftarrow B_1, \ldots B_k, not\ C_1, \ldots, not\ C_m. \quad (2)$$

As observed by (Oikarinen and Janhunen 2008), the heads of choice rules possessing multiple atoms can be freely split without affecting their semantics. When splitting such rules into n different rules $\{a_i\} \leftarrow B_1, \ldots B_k, not\ C_1, \ldots, not\ C_m$ where $1 \leq i \leq n$, the only concern is the creation of n copies of the rule body $B_1, \ldots B_k, not\ C_1, \ldots, not\ C_m$. However, new atoms can be introduced to circumvent this. There is a translation of these choice rules to normal logic programs (Ferraris and Lifschitz 2005), which we assume is performed throughout this paper but that is omitted for readability. We deal only with ground programs and use variables as syntactic placeholders.

2.2 Modular Logic Programming

Modules, in the sense of (Oikarinen and Janhunen 2008), are essentially sets of rules with an input and output interface:

Definition 1 (Program Module) *A logic program module $\mathcal{P}$ is a tuple $\langle R, I, O, H\rangle$ where:*

1. *R is a finite set of rules;*
2. *I, O, and H are pairwise disjoint sets of input, output, and hidden atoms;*
3. *$At(R) \subseteq At(\mathcal{P})$ defined by $At(\mathcal{P}) = I \cup O \cup H$; and*

4. $Head(R) \cap I = \emptyset$.

The set of atoms in $At_v(\mathcal{P}) = I \cup O$ are considered to be *visible* and hence accessible to other modules composed with $\mathcal{P}$ either to produce input for $\mathcal{P}$ or to make use of the output of $\mathcal{P}$. We use $At_i(\mathcal{P}) = I$ and $At_o(\mathcal{P}) = O$ to represent the input and output signatures of $\mathcal{P}$, respectively. The hidden atoms in $At_h(\mathcal{P}) = At(\mathcal{P}) \backslash At_v(\mathcal{P}) = H$ are used to formalize some auxiliary concepts of $\mathcal{P}$ which may not be sensible for other modules but may save space substantially. The condition $head(R) \notin I$ ensures that a module may not interfere with its own input by defining input atoms of I in terms of its rules. Thus, input atoms are only allowed to appear as conditions in rule bodies.

Example 2 *The use case in Example 1 is encoded into the five modules shown here:*

$$
\begin{array}{ll}
\mathcal{P}_A = & < \{buy(X) \leftarrow car(X), safe(X), not\ exp(X). \\
& \quad car(c_1). \quad car(c_2). \quad car(c_3).\}, \\
& \{safe(c_1), safe(c_2), safe(c_3), \\
& \quad exp(c_1), exp(c_2), exp(c_3)\}, \\
& \{buy(c_1), buy(c_2), buy(c_3)\}, \\
& \{car(c_1), car(c_2), car(c_3)\} > \\
\mathcal{P}_B = & < \{exp(c_2).\}, \{\}, \{exp(c_2), exp(c_3)\}, \{\} > \\
\mathcal{P}_C = & < \{exp(c_3).\}, \{\}, \\
& \{exp(c_1), exp(c_2), exp(c_3)\}, \{\} > \\
\mathcal{P}_{mg_1} = & < \{safe(c_1).\}, \{\}, \\
& \{safe(c_1), safe(c_2), safe(c_3)\}, \{\} > \\
\mathcal{P}_{mg_2} = & < \{safe(X) \leftarrow car(X), airbag(X). \\
& \quad car(c_1).\ car(c_2).\ car(c_3).\ airbag(c_1). \\
& \{airbag(c_3)\}.\ \}, \\
& \{\}, \{safe(c_1), safe(c_2), safe(c_3)\}, \\
& \{airbag(c_1), airbag(c_2), airbag(c_3), \\
& \quad car(c_1), car(c_2), car(c_3)\} > \qquad \blacksquare
\end{array}
$$

In Example 2, module $\mathcal{P}_A$ encodes the rule used by Alice to decide if a car should be bought. The safe and expensive atoms are its inputs, and the buy atoms its outputs; it uses hidden atoms $car/1$ to represent the domain of variables. Modules $\mathcal{P}_B$, $\mathcal{P}_C$ and $\mathcal{P}_{mg_1}$ capture the factual information in Example 1. They have no input and no hidden atoms, but Bob has only analyzed the price of cars c_2 and c_3. The ASP program module for the second magazine is more interesting[1], and expresses the rule used to determine if a car is safe, namely that a car is safe if it has an airbag; it is known that car c_1 has an airbag, c_2 does not, and the choice rule states that car c_3 may or may not have an airbag.

Next, the stable model semantics is generalized to cover modules by introducing a generalization of the Gelfond-Lifschitz's fixpoint definition. In addition to weekly default literals (i.e., not), also literals involving input atoms are used in the stability condition. In (Oikarinen and Janhunen 2008), the stable models of a module are defined as follows:

Definition 2 (Stable Models of Modules) *An interpretation $M \subseteq At(\mathcal{P})$ is a stable model of an ASP program module $\mathcal{P} = \langle R, I, O, H \rangle$, if and only if $M = LM\left(R^M \cup \{a. | a \in M \cap I\}\right)$. The stable models of $\mathcal{P}$ are denoted by $AS(\mathcal{P})$.*

Intuitively, the stable models of a module are obtained from the stable models of the rules part, for each possible combination of the input atoms.

Example 3 *Program modules $\mathcal{P}_B$, $\mathcal{P}_C$, and $\mathcal{P}_{mg_1}$ have each a single answer set $AS(\mathcal{P}_B)$ = $\{\{exp(c_2)\}\}$, $AS(\mathcal{P}_C)$ = $\{\{exp(c_3)\}\}$, and $AS(\mathcal{P}_{mg_1})$ = $\{\{safe(c_1)\}\}$. Module $\mathcal{P}_{mg_2}$ has two stable models, namely: $\{safe(c_1), car(c_1), car(c_2), car(c_3), airbag(c_1)\}$, and $\{safe(c_1), safe(c_3), car(c_1), car(c_2), car(c_3), airbag(c_1), airbag(c_3)\}$.*

Alice's ASP program module has $2^6 = 64$ models corresponding each to an input combination of safe and expensive atoms. Some of these models are:

$$
\begin{array}{ll}
\{ & buy(c_1), car(c_1), car(c_2), car(c_3), safe(c_1) \quad \} \\
\{ & buy(c_1), buy(c_3), car(c_1), car(c_2), car(c_3), \\
& safe(c_1), safe(c_3) \quad \} \\
\{ & buy(c_1), car(c_1), car(c_2), car(c_3), exp(c_3), \\
& safe(c_1), safe(c_3) \quad \} \blacksquare
\end{array}
$$

2.3 Composing programs from models

The composition of models is obtained from the union of program rules and by constructing the composed output set as the union of modules' output sets, thus removing from the input all the specified output atoms. (Oikarinen and Janhunen 2008) define their first composition operator as follows: Given two modules $\mathcal{P}_1 = \langle R_1, I_1, O_1, H_1 \rangle$ and $\mathcal{P}_2 = \langle R_2, I_2, O_2, H_2 \rangle$, their composition $\mathcal{P}_1 \oplus \mathcal{P}_2$ is defined when their output signatures are disjoint, that is, $O_1 \cap O_2 = \emptyset$, and they respect each others hidden atoms, i.e., $H_1 \cap At(\mathcal{P}_2) = \emptyset$ and $H_2 \cap At(\mathcal{P}_1) = \emptyset$. Then their composition is

$$\mathcal{P}_1 \oplus \mathcal{P}_2 = \langle R_1 \cup R_2, (I_1 \backslash O_2) \cup (I_2 \backslash O_1), O_1 \cup O_2, H_1 \cup H_2 \rangle$$

However, the conditions given for $\oplus$ are not enough to guarantee compositionality in the case of answer sets and as such they define a restricted form:

Definition 3 (Module Union Operator $\sqcup$) *Given modules $\mathcal{P}_1, \mathcal{P}_2$, their union is $\mathcal{P}_1 \sqcup \mathcal{P}_2 = \mathcal{P}_1 \oplus \mathcal{P}_2$ whenever* **(i)** *$\mathcal{P}_1 \oplus \mathcal{P}_2$ is defined and* **(ii)** *$\mathcal{P}_1$ and $\mathcal{P}_2$ are mutually independent*[2].

Natural join ($\bowtie$) on visible atoms is used in (Oikarinen and Janhunen 2008) to combine stable models of modules as follows:

Definition 4 (Join) *Given modules $\mathcal{P}_1$ and $\mathcal{P}_2$ and sets of interpretations $A_1 \subseteq 2^{At(\mathcal{P}_1)}$ and $A_2 \subseteq 2^{At(\mathcal{P}_2)}$, the natural join of A_1 and A_2 is:*

$$
\begin{array}{ll}
A_1 \bowtie A_2 = \{ & M_1 \cup M_2 \mid M_1 \in A_1, M_2 \in A_2 \text{ and} \\
& M_1 \cap At_v(\mathcal{P}_2) = M_2 \cap At_v(\mathcal{P}_1)\}
\end{array}
$$

This leads to their main result, stating that:

[1] car belongs to both hidden signatures of $\mathcal{P}_A$ and $\mathcal{P}_{mg_2}$ which is not allowed when composing these modules, but for clarity we omit a renaming of the $car/1$ predicate.

[2] There are no positive cyclic dependencies among rules in different modules, defined as loops through input and output signatures.

Theorem 1 (Module Theorem) *If $\mathcal{P}_1, \mathcal{P}_2$ are modules such that $\mathcal{P}_1 \sqcup \mathcal{P}_2$ is defined, then*

$$AS(\mathcal{P}_1 \sqcup \mathcal{P}_2) = AS(\mathcal{P}_1) \bowtie AS(\mathcal{P}_2)$$

Still according to (Oikarinen and Janhunen 2008), their module theorem also straightforwardly generalizes for a collection of modules because the module union operator $\sqcup$ is commutative, associative, and has the identity element $< \emptyset, \emptyset, \emptyset, \emptyset >$.

Example 4 *Consider the composition $\mathcal{Q} = (\mathcal{P}_A \sqcup \mathcal{P}_{mg_1}) \sqcup \mathcal{P}_B$. First, we have*

$$\mathcal{P}_A \sqcup \mathcal{P}_{mg_1} = \left\langle \begin{array}{l} \{buy(X) \leftarrow car(X), safe(X), \\ \qquad not\ exp(X). \\ \ car(c_1).\ car(c_2).\ car(c_3).\ safe(c_1).\}, \\ \{exp(c_1), exp(c_2), exp(c_3)\}, \\ \{buy(c_1), buy(c_2), buy(c_3), \\ \ safe(c_1),\ safe(c_2),\ safe(c_3)\}, \\ \{car(c_1), car(c_2), car(c_3)\} \end{array} \right\rangle$$

It is immediate to see that the module theorem holds in this case. The visible atoms of $\mathcal{P}_A$ are $safe/1$, $exp/1$ and $buy/1$, and the visible atoms for $\mathcal{P}_{mg_1}$ are $\{safe(c_1), safe(c_2)\}$. The only model for $\mathcal{P}_{mg_1} = \{safe(c_1)\}$ when naturally joined with the models of $\mathcal{P}_A$, results in eight possible models where $safe(c_1)$, $not\ safe(c_2)$, and $not\ safe(c_3)$ hold, and $exp/1$ vary. The final ASP program module $\mathcal{Q}$ is

$$\left\langle \begin{array}{l} \{buy(X) \leftarrow car(X), safe(X), not\ exp(X). \\ \ car(c_1).\ car(c_2).\ car(c_3).\ exp(c_2).\ safe(c_1).\}, \\ \{exp(c_1)\}, \\ \{buy(c_1), buy(c_2), buy(c_3), exp(c_2), \\ \ safe(c_1), safe(c_2), safe(c_3)\}, \\ \{car(c_1), car(c_2), car(c_3)\} \end{array} \right\rangle$$

The stable models of $\mathcal{Q}$ are thus:

$$\{safe(c_1), exp(c_1), exp(c_2), car(c_1), car(c_2), car(c_3)\}$$
$$\{buy(c_1), safe(c_1), exp(c_2), car(c_1), car(c_2), car(c_3)\}$$ ■

2.4 Visible and Modular Equivalence

The notion of visible equivalence has been introduced in order to neglect hidden atoms when logic programs are compared on the basis of their models. The compositionality property from the module theorem enabled the authors to port this idea to the level of program modules–giving rise to modular equivalence of logic programs.

Definition 5 *Given two logic program modules $\mathcal{P}$ and $\mathcal{Q}$, they are:*
Visibly equivalent: *$\mathcal{P} \equiv_v \mathcal{Q}$ iff $At_v(\mathcal{P}) = At_v(\mathcal{Q})$ and there is a bijection $f : AS(\mathcal{P}) \rightarrow AS(\mathcal{Q})$ such that for all $M \in AS(\mathcal{P})$, $M \cap At_v(\mathcal{P}) = f(M) \cap At_v(\mathcal{Q})$.*
Modularly equivalent: *$\mathcal{P} \equiv_m \mathcal{Q}$ iff $At_i(\mathcal{P}) = At_i(\mathcal{Q})$ and $\mathcal{P} \equiv_v Q$.*

So, two modules are visibly equivalent if there is a bijection among their stable models, and they coincide in their visible parts. If additionally, the two program modules have the same input and output atoms, then they are modularly equivalent.

2.5 Shortcomings

The conditions imposed in these definitions bring about some shortcomings such as the fact that the output signatures of two modules must be disjoint which disallows many practical applications e.g., we are not able to combine the results of program module $\mathcal{Q}$ with any of $\mathcal{P}_C$ or $\mathcal{P}_{mg_2}$, and thus it is impossible to obtain the combination of the five modules. Also because of this, the module union operator $\sqcup$ is not reflexive. By trivially waiving this condition, we immediately get problems with conflicting modules. The compatibility criterion for the operator $\bowtie$ also rules out the compositionality of mutually dependent modules, but allows positive loops inside modules or negative loops in general.

Example 5 (Common Outputs) *Given $\mathcal{P}_B$ and $\mathcal{P}_C$, which respectively have:*
$AS(\mathcal{P}_B)$=$\{\{exp(c_2)\}\}$ and $AS(\mathcal{P}_C)$=$\{\{exp(c_3)\}\}$, the single stable model of their union $AS(\mathcal{P}_B \sqcup \mathcal{P}_C)$ is:

$$\{exp(c_2), exp(c_3)\}$$

However, the join of their stable models is $AS(\mathcal{P}_B) \bowtie AS(\mathcal{P}_C) = \emptyset$, invalidating the module theorem. ■

We illustrate next the issue with positive loops between modules.

Example 6 (Cyclic Dependencies) *Take the following two program modules:*

$$\mathcal{P}_1 = \langle \{airbag \leftarrow safe.\}, \{safe\}, \{airbag\}, \emptyset \rangle$$
$$\mathcal{P}_2 = \langle \{safe \leftarrow airbag.\}, \{airbag\}, \{safe\}, \emptyset \rangle$$

Their stable models are:

$$AS(\mathcal{P}_1) = AS(\mathcal{P}_2) = \{\{\}, \{airbag, safe\}\}$$

while the single stable model of the union $AS(\mathcal{P}_1 \sqcup \mathcal{P}_2)$ is the empty model $\{\}$. Therefore $AS(\mathcal{P}_1 \sqcup \mathcal{P}_2) \neq AS(\mathcal{P}_1) \bowtie AS(\mathcal{P}_2)$ = $\{\{\}, \{airbag, safe\}\}$, thus also invalidating the module theorem. ■

3 Positive Cyclic Dependencies Between Modules

To attain a generalized form of compositionality we need to be able to deal with the two restrictions identified previously, namely cyclic dependencies between modules. In the literature, (Dao-Tran et al. 2009) presents a solution based on a model minimality property. It forces one to check for minimality on every comparable models of all program modules being composed. It is not applicable to our setting though, which can be seen in Example 7 where logical constant $\bot$ represents value $false$.

Example 7 (Problem with minimization) *Given modules $\mathcal{P}_1 = \langle \{a \leftarrow b.\ \bot \leftarrow not\ b.\}, \{b\}, \{a\}, \{\} \rangle$ with one answer set $\{a, b\}$, and $\mathcal{P}_2 = \langle \{b \leftarrow a.\}, \{a\}, \{b\}, \{\} \rangle$ with stable models $\{\}$ and $\{a, b\}$, their composition has no inputs and no intended stable models while their minimal join contains $\{a, b\}$.* ■

Another possible solution requires the introduction of extra information in the models to be able to detect mutual positive dependencies. This need has been identified before (Slota and Leite 2012) and is left for future work.

4 Generalizing Modularity in ASP by Allowing Common Outputs

After having identified the shortcomings in the literature, we proceed now to seeing how compositionality can be maintained while allowing modules to have common output atoms. In this section we present two versions of compositions: (1) A relaxed composition operator ($\uplus$), aiming at maximizing information in the stable models of modules. Unfortunately, we show that this operation is not compositional. (2) A conservative composition operator ($\otimes$), aiming at maximizing compatibility of atoms in the stable models of modules. This version implies redefining the composition operator by resorting to a program transformation but uses the original join operator.

4.1 Extra module operations

First, one requires fundamental operations for renaming atoms in the output signatures of modules with fresh ones:

Definition 6 (Output renaming) *Let $\mathcal{P}$ be the program module $\mathcal{P} = \langle R, I, O, H\rangle$, $o \in O$ and $o' \notin At(\mathcal{P})$. The renamed output program module $\rho_{o' \leftarrow o}(\mathcal{P})$ is the program module $\langle R' \cup \{\bot \leftarrow o', not\ o.\}, I \cup \{o\}, \{o'\} \cup (O \setminus \{o\}), H\rangle$. The program part R' is constructed by substituting the head of each rule $o \leftarrow Body$ in R by $o' \leftarrow Body$. The heads of other rules remain unchanged, as well as the bodies of all rules.*

Mark that, by making o an input atom, the renaming operation can introduce extra stable models. However, the original stable models can be recovered by selecting the models where o' has exactly the same truth-value of o. The constraint throws away models where o' holds but not o. We will abuse notation and denote $\rho_{o'_1 \leftarrow o_1}\left(\ldots\left(\rho_{o'_n \leftarrow o_n}(\mathcal{P})\right)\ldots\right)$ by $\rho_{\{o'_1,\ldots,o'_n\} \leftarrow \{o_1,\ldots,o_n\}}(\mathcal{P})$.

Example 8 (Renaming) *Recall the module representing Alice's conditions in Example 2. Its renamed output program module $\rho_{o' \leftarrow o}(\mathcal{P}_A)$ is the program module:*

$$
\begin{aligned}
\rho_{o' \leftarrow o}(\mathcal{P}_A) =< \quad & \{buy'(X) \leftarrow car(X), safe(X), \\
& \qquad not\ exp(X). \\
& \ car(c_1).\quad car(c_2).\quad car(c_3). \\
& \ \bot \leftarrow buy(X)', not\ buy(X).\}, \\
& \{buy(X), safe(c_1), safe(c_2), safe(c_3), \\
& \ exp(c_1), exp(c_2), exp(c_3)\}, \\
& \{buy'(c_1), buy'(c_2), buy'(c_3)\}, \\
& \{car(c_1), car(c_2), car(c_3)\} > \qquad \blacksquare
\end{aligned}
$$

Still before we dwell any deeper in this subject, we define operations useful to project or hide sets of atoms from a module.

Definition 7 (Hiding and Projecting Atoms) *Let $\mathcal{P} = \langle R, I, O, H\rangle$ be a module and S an arbitrary set of atoms. If we want to* Hide *(denoted as $\setminus$) S from program module $\mathcal{P}$, we use $\mathcal{P}\setminus S = \langle R \cup \{\{i\}. \mid i \in I \cap S\}, I \setminus S, O \setminus S, H \cup ((I \cup O) \cap S)\rangle$. Dually, we can* Project *(denoted as $|$) over S in the following way: $\mathcal{P}\,|_S = \langle R \cup \{\{i\}. \mid i \in I \setminus S\}, I \cap S, O \cap S, H \cup ((I \cup O) \setminus S)\rangle$.*

Both operators *Hide* and *Project* do not change the stable models of the original program, i.e. $AS(\mathcal{P}) = AS(\mathcal{P}\setminus S) = AS(\mathcal{P}_{|S})$ but do change the set of visible atoms $At_v(\mathcal{P}\setminus S) = At_v(\mathcal{P}) \setminus S$ and $At_v(\mathcal{P} \mid S) = At_v(\mathcal{P}) \cap S$

4.2 Relaxed Output Composition

For the reasons presented before, we start by defining a generalized version of the composition operator, by removing the condition enforcing disjointness of the output signatures of the two modules being combined.

Definition 8 (Relaxed Composition) *Given two modules $\mathcal{P}_1 = \langle R_1, I_1, O_1, H_1\rangle$ and $\mathcal{P}_2 = \langle R_2, I_2, O_2, H_2\rangle$, their composition $\mathcal{P}_1 \uplus \mathcal{P}_2$ is defined when they respect each others hidden atoms, i.e., $H_1 \cap At(\mathcal{P}_2) = \emptyset$ and $H_2 \cap At(\mathcal{P}_1) = \emptyset$. Then their composition is $\mathcal{P}_1 \uplus \mathcal{P}_2 = \langle R_1 \cup R_2, (I_1 \cup I_2) \setminus (O_1 \cup O_2), O_1 \cup O_2, H_1 \cup H_2\rangle$.*

Obviously, the following important properties hold for $\uplus$:

Lemma 1 *The relaxed composition operator is reflexive, associative, commutative and has the identity element $< \emptyset, \emptyset, \emptyset, \emptyset >$.*

Having defined the way to deal with common outputs in the composition of modules, we would like to redefine the operator $\bowtie$ for combining the stable models of these modules. However, this is shown here to be impossible.

Lemma 2 *The operation $\uplus$ is not compositional, i.e. for any join operation $\bowtie'$, it is not always the case that $AS(\mathcal{P}_1 \uplus \mathcal{P}_2) = AS(\mathcal{P}_1) \bowtie' AS(\mathcal{P}_2)$.*

As we have motivated in the introduction, it is important to applications to be able to use $\uplus$ to combine program modules, and retain some form of compositionality. The following definition presents a construction that adds the required information in order to be able to combine program modules using the original natural join.

Definition 9 (Transformed Relaxed Composition) *Consider the program modules $\mathcal{P}_1 = \langle R_1, I_1, O_1, H_1\rangle$ and $\mathcal{P}_2 = \langle R_2, I_2, O_2, H_2\rangle$. Let $O = O_1 \cap O_2$, and define the sets of newly introduced atoms O'=$\{o' \mid o \in O\}$ and O''=$\{o'' \mid o \in O\}$. Construct program module:*

$$
\begin{aligned}
&\mathcal{P}_{union} = < R_{union}, O' \cup O'', O, \emptyset > \text{ where:} \\
&R_{union} = \{o \leftarrow o'. \mid o' \in O'\} \cup \{o \leftarrow o''. \mid o'' \in O''\}.
\end{aligned}
$$

The transformed relaxed composition is defined as the program module

$$
\begin{aligned}
(\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2) = \quad & [\rho_{O' \leftarrow O}(\mathcal{P}_1) \sqcup \rho_{O'' \leftarrow O}(\mathcal{P}_2) \sqcup \mathcal{P}_{union}] \setminus \\
& [O' \cup O'']
\end{aligned}
$$

Intuitively, we rename the common output atoms in the original modules, and introduce an extra program module that unites the contributions of each module by a pair of rules for each common atom $o \leftarrow o'$ and $o \leftarrow o''$. We then hide all the auxiliary atoms to obtain the original visible signature. If $O = \emptyset$ then $\mathcal{P}_{union}$ is empty, and all the other modules are not altered, falling back to the original definition.

Theorem 2 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be arbitrary program modules without positive dependencies among them. Then, modules joined with operators $\uplus$ and $\uplus^{RT}$ are modularly equivalent:*

$$\mathcal{P}_1 \uplus \mathcal{P}_2 \equiv_m \mathcal{P}_1 \uplus^{RT} \mathcal{P}_2.$$

The important remark is that according to the original module theorem we have: $AS(\rho_{O' \leftarrow O}(\mathcal{P}_1) \sqcup \rho_{O'' \leftarrow O}(\mathcal{P}_2) \sqcup \mathcal{P}_{union}) = AS(\rho_{O' \leftarrow O}(\mathcal{P}_1)) \bowtie AS(\rho_{O'' \leftarrow O}(\mathcal{P}_2)) \bowtie AS(\mathcal{P}_{union})$. Therefore, from a semantical point of view, users can always substitute module $\mathcal{P}_1 \uplus \mathcal{P}_2$ by $\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2$, which has an extra cost since the models of the renamed program modules may increase. This is, however, essential to regain compositionality.

Example 9 *Considering program modules $\mathcal{Q}_1 =< \{a. \ \bot \leftarrow a, b.\}, \emptyset, \{a, b\}, \emptyset >$ and $\mathcal{Q}_2 = \langle \{b.\}, \emptyset, \{b\}, \emptyset \rangle$, we have:*

$$\begin{array}{lcll}
\rho_{a',b' \leftarrow a,b}(\mathcal{P}_1) = & < & \{ & a'. \ \bot \leftarrow a', not\ a. \\
 & & & \bot \leftarrow b', not\ b.\}, \\
 & & \{ & a, b\}, \{a', b'\}, \emptyset \quad > \\
\rho_{a'',b'' \leftarrow a,b}(\mathcal{P}_2) = & < & \{ & b''. \ \bot \leftarrow a'', not\ a. \\
 & & & \bot \leftarrow b'', not\ b.\}, \\
 & & \{ & a, b\}, \{a'', b''\}, \emptyset \quad > \\
\mathcal{P}_{union} = & < & \{ & a \leftarrow a'. \ a \leftarrow a''. \\
 & & & b \leftarrow b'. \ b \leftarrow b''.\}, \\
 & & \{ & a', a'', b', b''\}, \{a, b\}, \emptyset > \\
\rho_{a',b' \leftarrow a,b}(\mathcal{Q}_1) = & < & \{ & a'. \ \bot \leftarrow a, b. \\
 & & & \bot \leftarrow a', not\ a. \\
 & & & \bot \leftarrow b', not\ b.\}, \\
 & & \{ & a, b\}, \{a', b'\}, \emptyset \quad > \\
\rho_{a'',b'' \leftarrow a,b}(\mathcal{Q}_2) = & & & \rho_{a'',b'' \leftarrow a,b}(\mathcal{P}_2) \\
\mathcal{Q}_3 = \mathcal{P}_{union} & & &
\end{array}$$

The stable models of the first two modules are $\{\{a, a'\}, \{a, b, a'\}\}$ and $\{\{b, b''\}, \{a, b, b''\}\}$, respectively. Their join is $\{\{a, b, a', b''\}\}$ and the returned model belongs to $\mathcal{P}_{union}$ (and thus it is compatible), and corresponds to the only intended model $\{a, b\}$ of $\mathcal{P}_1 \uplus \mathcal{P}_2$. Note that the stable models of $\mathcal{P}_{union}$ are 16, corresponding to the models of propositional formula $(a \equiv a' \vee a'') \wedge (b \equiv b' \vee b'')$. Regarding, the transformed module $\rho_{a',b' \leftarrow a,b}(\mathcal{Q}_1)$ it discards the model $\{a, b, a'\}$, having stable models $\{\{a, a'\}\}$. But now the join is empty, as intended. ■

4.3 Conservative Output Composition

In order to preserve the original outer join operator, which is widely used in databases, for the form of composition we introduce next one must redefine the original composition operator ($\oplus$). We do that resorting to a program transformation s.t. the composition operator remains compositional with respect to the join operator ($\bowtie$). The transformation we present next consists of taking Definition 9 and adding an extra module to guarantee that only compatible models (models that coincide on the visible part) are retained.

Definition 10 (Conservative Composition) *Let $\mathcal{P}_1 = \langle R_1, I_1, O_1, H_1 \rangle$ and $\mathcal{P}_2 = \langle R_2, I_2, O_2, H_2 \rangle$ be modules such that their outputs are disjoint $O = O_1 \cap O_2 \neq \emptyset$. Let $O' = \{o' \mid o \in O\}$ and $O'' = \{o'' \mid o \in O\}$ be sets of newly introduced atoms.*

Construct program modules:

$$\begin{array}{l}
\mathcal{P}_{union} =< R_{union}, O' \cup O'', O, \emptyset > \text{ where:} \\
R_{union} = \{o \leftarrow o'. \mid o' \in O'\} \cup \{o \leftarrow o''. \mid o'' \in O''\}. \\
\mathcal{P}_{filter} =< \{\bot \leftarrow o', not\ o''. \ \bot \leftarrow not\ o', o''. \mid o \in O\}, \\
\qquad O' \cup O'', \emptyset, \emptyset >
\end{array}$$

The conservative composition is defined as the program module: $\mathcal{P}_1 \otimes \mathcal{P}_2 = [(\rho_{O' \leftarrow O}(\mathcal{P}_1) \sqcup \rho_{O'' \leftarrow O}(\mathcal{P}_2) \sqcup \mathcal{P}_{union} \sqcup \mathcal{P}_{filter}] \setminus (O' \cup O'')$.

Note here that each clause not containing atoms that belong to $O_1 \cap O_2$ in $\mathcal{P}_1 \cup \mathcal{P}_2$ is included in $\mathcal{P}_1 \otimes \mathcal{P}_2$. So, if there are no common output atoms the original union based composition is obtained. Therefore, it is easy to see that this transformational semantics ($\otimes$) is a conservative extension to the existing one ($\oplus$).

Theorem 3 (Conservative Module Theorem) *If $\mathcal{P}_1, \mathcal{P}_2$ are modules such that $\mathcal{P}_1 \otimes \mathcal{P}_2$ is defined, then a model $M \in AS(\mathcal{P}_1 \otimes \mathcal{P}_2)$ iff $M \cap (At(\mathcal{P}_1) \cup At(\mathcal{P}_2)) \in AS(\mathcal{P}_1) \bowtie AS(\mathcal{P}_2)$.*

The above theorem is very similar to the original Module Theorem of Oikarinnen and Janhunen apart from the extra renamed atoms required in $\mathcal{P}_1 \otimes \mathcal{P}_2$ to obtain compositionality.

Example 10 *Returning to the introductory example, we can conclude that $\mathcal{P}_{mg_1} \otimes \mathcal{P}_{mg_2}$ has only one answer set:*

$$\{safe(c_1), airbag(c_1), car(c_1), car(c_2), car(c_3)\}$$

since this is the only compatible model between $\mathcal{P}_{mg_1}$ and $\mathcal{P}_{mg_2}$. The stable models of $\rho(\mathcal{P}_{mg_1})$ and $\rho(\mathcal{P}_{mg_2})$, are collected in the table below where compatible models appear in the same row and $car(c_1), car(c_2), car(c_3)$ has been omitted from $AS(\rho(\mathcal{P}_{mg_2}))$. Atom s (respectively a) stands for $safe$ (respectively $airbag$).

Answer sets of $\rho(\mathcal{P}_{mg_1})$	*Answer sets of $\rho(\mathcal{P}_{mg_2})$*
$\{s(c_1), s'(c_1)\}$	$\{s(c_1), s''(c_1), a(c_1)\}$
$\{s(c_1), s(c_2), s'(c_1)\}$	$\{s(c_1), s(c_2), s''(c_1), a(c_1)\}$
$\{s(c_1), s(c_3), s'(c_1)\}$	$\{s(c_1), s(c_3), s''(c_1), a(c_1)\}$ $\{s(c_1), s(c_3), s''(c_1),$ $s''(c_3), a(c_1), a(c_3)\}$
$\{s(c_1), s(c_2), s(c_3),$ $s'(c_1)\}$	$\{s(c_1), s(c_2), s(c_3),$ $s''(c_1), a(c_1)\}$ $\{s(c_1), s(c_2), s(c_3), s''(c_1),$ $s''(c_3), a(c_1), a(c_3), c(c_1)\}$

The only compatible model retained after composing with $\mathcal{P}_{union}$ and $\mathcal{P}_{filter}$ is the combination of the stable models in the first row:

$$\{s(c_1), s'(c_1), s''(c_1), a(c_1), c(c_1), c(c_2), c(c_3)\}.$$

Naturaly, this corresponds to the intended result if we ignore the s' and s'' atoms. ■

We underline that models of composition $\mathcal{P}_1 \otimes \mathcal{P}_2$ will either contain all atoms o, o', and o'' or none of them, and will only join compatible models from $\mathcal{P}_1$ having $\{o, o'\}$ with models in $\mathcal{P}_2$ having $\{o, o''\}$, or models without atoms in $\{o, o', o''\}$.

Shortcomings Revisited The resulting models of composing modules using the transformation and renaming methods described so far in this Section 4 can be minimised a posteriori following the minimization method described in Section 3.

4.4 Complexity

Regarding complexity, checking the existence of $M \in P_1 \oplus P_2$ and $M \in P_1 \uplus^{RT} P_2$ is an NP-complete problem. It is immediate to define a decision algorithm belonging to Σ_2^p that checks existence of a stable model of the module composition operators. This is strictly less than the results in the approach of (Dao-Tran et al. 2009) where the existence decision problem for propositional theories is NEXP^{NP}-complete – however their approach allows disjunctive rules.

5 Conclusions and Future Work

We redefined the necessary operators in order to relax the conditions for combining modules with common atoms in their output signatures. Two alternative solutions are presented, both allowing us to retain compositionality while dealing with a more general setting than before. (Dao-Tran et al. 2009) provide an embedding of the original composition operator of Oikarinen and Janhunen into their approach. Since our constructions rely on a transformational approach using operator $\sqcup$ of Oikarinen and Janhunen, by composing both translations, an embedding into (Dao-Tran et al. 2009) is immediately obtained. It remains to be checked whether the same translation can be used in the presence of positive cycles. (Tasharrofi and Ternovska 2011) take (Janhunen et al. 2009) and extend it with an algebra which includes a new operation of feedback (loop) over modules. They have shown that the loop operation adds significant expressive power – modules can can express all (and only) problems in NP. The other issues remain unsolved though.

The module theorem has been extended to the general theory of stable models (Babb and Lee 2012), being applied to non-ground logic programs containing choice rules, the count aggregate, and nested expressions. It is based on the new findings about the relationship between the module theorem and the splitting theorem. It retains the composition condition of disjoint outputs and still forbids positive dependencies between modules. As for disjunctive versions, (Janhunen et al. 2009) introduced a formal framework for modular programming in the context of DLPs under stable-model semantics. This is based on the notion of DLP-functions, which resort to appropriate input/output interfacing. Similar module concepts have already been studied for the cases of normal logic programs and ASPs and even propositional theories, but the special characteristics of disjunctive rules are properly taken into account in the syntactic and semantic definitions of DLP functions presented therein. In (Gebser et al. 2011), MLP is used as a basis for Reactive Answer Set Programming, aiming at reasoning about real-time dynamic systems running online in changing environments.

As future work we can straightforwardly extend these results to probabilistic reasoning with stable models by applying the new module theorem to (Damásio and Moura 2011), as well as to DLP functions and general stable models. An implementation of the framework is also foreseen in order to assess the overhead when compared with the original benchmarks in (Oikarinen and Janhunen 2008). Based on our own preliminary work and results in the literature, we believe that a fully compositional semantics can be attained by resorting to partial interpretations e.g., SE-models (Turner 2003) for defining program models at the semantic level. It is known that one must include extra information about the support of each atom in the models in order to attain generalized compositionality and SE-models appear to be enough.

References

Babb, J., and Lee, J. 2012. Module theorem for the general theory of stable models. *TPLP* 12(4-5):719–735.

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

Bugliesi, M.; Lamma, E.; and Mello, P. 1994. Modularity in logic programming. *J. Log. Program.* 19/20:443–502.

Damásio, C. V., and Moura, J. 2011. Modularity of p-log programs. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, LPNMR'11, 13–25. Berlin, Heidelberg: Springer-Verlag.

Dao-Tran, M.; Eiter, T.; Fink, M.; and Krennwallner, T. 2009. Modular nonmonotonic logic programming revisited. In Hill, P. M., and Warren, D. S., eds., *ICLP 2009, Pasadena, USA, 2009*, volume 5649.

Eiter, T.; Faber, W.; Leone, N.; and Pfeifer, G. 2001. Computing preferred and weakly preferred answer sets by meta-interpretation in answer set programming. In *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming*, 45–52. AAAI Press.

Ferraris, P., and Lifschitz, V. 2005. Weight constraints as nested expressions. *TPLP* 5(1-2):45–74.

Gaifman, H., and Shapiro, E. 1989. Fully abstract compositional semantics for logic programs. In *symposium on Principles of programming languages*, POPL, 134–142. New York, NY, USA: ACM.

Gebser, M.; Grote, T.; Kaminski, R.; and Schaub, T. 2011. Reactive answer set programming. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, LPNMR'11, 54–66. Berlin, Heidelberg: Springer-Verlag.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Program.* MIT Press.

Giordano, L., and Martelli, A. 1994. Structuring logic programs: a modal approach. *The Journal of Logic Programming* 21(2):59 – 94.

Janhunen, T.; Oikarinen, E.; Tompits, H.; and Woltran, S. 2009. Modularity aspects of disjunctive stable models. *J. Artif. Int. Res.* 35(1):813–857.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A module-based framework for multi-language constraint modeling. In Erdem, E.; Lin, F.; and Schaub, T., eds., *Proceedings of the 10th International Conference*

on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009), volume 5753 of *Lecture Notes in Artificial Intelligence*, 155–169. Springer.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138(1-2):39–54.

Mancarella, P., and Pedreschi, D. 1988. An algebra of logic programs. In *ICLP/SLP*, 1006–1023.

Marek, V. W., and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*.

Miller, D. 1986. A theory of modules for logic programming. In *In Symp. Logic Programming*, 106–114.

Niemelä, I. 1998. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25:72–79.

Oikarinen, E., and Janhunen, T. 2008. Achieving compositionality of the stable model semantics for smodels programs1. *Theory Pract. Log. Program.* 8(5-6):717–761.

O'Keefe, R. A. 1985. Towards an algebra for constructing logic programs. In *SLP*, 152–160.

Slota, M., and Leite, J. 2012. Robust equivalence models for semantic updates of answer-set programs. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Proc. of KR 2012*. AAAI Press.

Tasharrofi, S., and Ternovska, E. 2011. A semantic account for modularity in multi-language modelling of search problems. In *Proceedings of the 8th international conference on Frontiers of combining systems*, FroCoS'11, 259–274. Berlin, Heidelberg: Springer-Verlag.

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

A Proofs

Proof 1 (Lemma 2) *A join operation is a function mapping a pair of sets of interpretations into a set of interpretations. Consider the following program modules:*

$$\begin{array}{ll} \mathcal{P}_1 =< \{a.\},\emptyset,\{a,b\},\emptyset > & \mathcal{Q}_1 =< \{a.\ \bot \leftarrow a,b.\}, \\ & \quad \emptyset,\{a,b\},\emptyset > \\ \mathcal{P}_2 = \langle\{b.\},\emptyset,\{b\},\emptyset\rangle & \mathcal{Q}_2 = \langle\{b.\},\emptyset,\{b\},\emptyset\rangle \\ \mathcal{P}_1 \uplus \mathcal{P}_2 =< \{a.\ b.\},\emptyset, & \mathcal{Q}_1 \uplus \mathcal{Q}_2 =< \{a.\ \bot \leftarrow a,b. \\ \quad \{a,b\},\emptyset > & \quad b.\},\emptyset,\{a,b\},\emptyset > \end{array}$$

One sees that $AS(\mathcal{P}_1) = AS(\mathcal{Q}_1) = \{\{a\}\}$*, and* $AS(\mathcal{P}_2) = AS(\mathcal{Q}_2) = \{\{b\}\}$ *but* $AS(\mathcal{P}_1 \uplus \mathcal{P}_2) = \{\{a,b\}\}$ *while* $AS(\mathcal{Q}_1 \uplus \mathcal{Q}_2) = \{\}$*. Therefore, it cannot exist* $\bowtie'$ *since this would require* $AS(\mathcal{P}_1 \uplus \mathcal{P}_2) = AS(\mathcal{P}_1) \bowtie' AS(\mathcal{P}_2) = \{\{a\}\} \bowtie' \{\{b\}\} = AS(\mathcal{Q}_1) \bowtie' AS(\mathcal{Q}_2) = AS(\mathcal{Q}_1 \uplus \mathcal{Q}_2)$*, a contradiction.* □

Proof 2 (Theorem 2) *By reduction of the conditions of the theorem to the conditions necessary for applying the original Module Theorem. If* $\mathcal{P}_1 \uplus \mathcal{P}_2$ *is defined then let their transformed relaxed composition be* $T = (\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2)$*. It is clear that the output atoms of* T *are* $O_1 \cup O_2$*, the input atoms are* $(I_1 \cup I_2) \setminus (O_1 \cup O_2)$*, and the hidden atoms are* $H_1 \cup H_2 \cup O' \cup O''$*. Note that before the application of the hiding operator the output atoms are* $O_1 \cup O_2 \cup O' \cup O''$*. The original composition operator* $\sqcup$ *can be applied since the outputs of* $\rho_{O' \leftarrow O}(\mathcal{P}_1)$*,* $\rho_{O'' \leftarrow O}(\mathcal{P}_2)$ *and* $\mathcal{P}_{union}$ *are respectively* $O' \cup (O_1 \setminus O)$*,* $O'' \cup (O_2 \setminus O)$ *and* $O = O_1 \cap O_2$*, which are pairwise disjoint. Because of this, we are in the conditions of the original Module Theorem and thus it is applicable to the result of the modified composition* $\uplus$ *iff the transformation did not introduce positive loops between the program parts of the three auxiliary models. If* $\mathcal{P}_1 \uplus \mathcal{P}_2$ *had no loops between the common output atoms than its transformation* $\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2$ *also does not because it results from a renaming into new atoms.*

Consider now the rules part of T*; if we ignore the extra introduced atoms in* O' *and* O'' *the program obtained has exactly the same stable models of the union of program parts of* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*. Basically, we are substituting the union of* $o \leftarrow Body_1^1., \ldots, o \leftarrow Body_m^1.$ *in* $\mathcal{P}_1$*, and* $o \leftarrow Body_1^2., \ldots, o \leftarrow Body_n^2.$ *in* $\mathcal{P}_2$ *by:*

$$\begin{array}{ll} o \leftarrow o'. & o \leftarrow o''. \\ o' \leftarrow Body_1^1. & o'' \leftarrow Body_1^2. \\ \ldots & \ldots \\ o' \leftarrow Body_m^1. & o'' \leftarrow Body_n^2. \\ \bot \leftarrow o', not\ o. & \bot \leftarrow o'', not\ o. \end{array}$$

This guarantees visible equivalence of $\mathcal{P}_1 \uplus \mathcal{P}_2$ *and* $\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2$*, since the models of each combined modules are in one-to-one correspondence, and they coincide in the visible atoms. The contribution of the common output atoms is recovered by the joins involving atoms in* O'*,* O'' *and* O*, that are all pairwise disjoint, and ensuring that stable models obey to* $o = o' \vee o''$ *via program module* $\mathcal{P}_{union}$*. The constraints introduced in the transformed models* $\rho_{O' \leftarrow O}(\mathcal{P}_1)$ *(resp.* $\rho_{O'' \leftarrow O}(\mathcal{P}_2)$*) simply prune models that have* o *false and* o' *(resp.* o''*) true, reducing the number of models necessary to consider. Since the input and output atoms of* $\mathcal{P}_1 \uplus \mathcal{P}_2$

and $\mathcal{P}_1 \uplus^{RT} \mathcal{P}_2$ are the same, then $\mathcal{P}_1 \uplus \mathcal{P}_2 \equiv_m \mathcal{P}_1 \uplus^{RT} \mathcal{P}_2$. □

Proof 3 (Theorem 3) *The theorem states that if we ignore the renamed literals in $\otimes$ the models are exactly the same, as expected. The transformed program module $\mathcal{P}_1 \otimes \mathcal{P}_2$ corresponds basically to the union of programs, as seen before. Consider a common output atom o. The constraints in the module part $\mathcal{P}_{filter}$ combined with the rules in $\mathcal{P}_{union}$ restrict the models to the cases for which $o \equiv o' \equiv o''$. The equivalence $o \equiv o'$ restricts the stable models of $\rho_{o' \leftarrow o}(\mathcal{P}_1)$ to the original stable models (except for the extra atom o') of $\mathcal{P}_1$, and similarly the equivalence $o \equiv o''$ filters the stable models of $\rho_{o'' \leftarrow o}(\mathcal{P}_2)$ obtaining the original stable models of $\mathcal{P}_2$. Now it is immediate to see that compositionality is retained by making the original common atoms o compatible.* □

The Multi-engine ASP Solver ME-ASP: Progress Report

Marco Maratea
DIBRIS,
Univ. degli Studi di Genova,
Viale F. Causa 15, 16145 Genova, Italy
marco@dist.unige.it

Luca Pulina
POLCOMING,
Univ. degli Studi di Sassari,
Viale Mancini 5, 07100 Sassari, Italy
lpulina@uniss.it

Francesco Ricca
Dip. di Matematica ed Informatica,
Univ. della Calabria,
Via P. Bucci, 87030 Rende, Italy
ricca@mat.unical.it

Abstract

ME-ASP is a multi-engine solver for ground ASP programs. It exploits algorithm selection techniques based on classification to select one among a set of out-of-the-box heterogeneous ASP solvers used as black-box engines. In this paper we report on (i) a new optimized implementation of ME-ASP; and (ii) an attempt of applying algorithm selection to non-ground programs. An experimental analysis reported in the paper shows that (i) the new implementation of ME-ASP is substantially faster than the previous version; and (ii) the multi-engine recipe can be applied to the evaluation of non-ground programs with some benefits.

Introduction

Answer Set Programming (Baral 2003; Eiter, Gottlob, and Mannila 1997; Gelfond and Lifschitz 1988; 1991; Marek and Truszczyński 1998; Niemelä 1998) (ASP) is a declarative language based on logic programming and non-monotonic reasoning. The applications of ASP belong to several areas, e.g., ASP was used for solving a variety of hard combinatorial problems (see e.g., (Calimeri et al. 2011) and (Potsdam since 2002)).

Nowadays, several efficient ASP systems are available (Gebser et al. 2007; Janhunen, Niemelä, and Sevalnev 2009; Leone et al. 2006; Lierler 2005; Mariën et al. 2008; Simons, Niemelä, and Soininen 2002). It is well-established that, for solving empirically hard problems, there is rarely a best algorithm/heuristic, while it is often the case that different algorithms perform well on different problems/instances. It can be easily verified (e.g., by analyzing the results of the ASP competition series) that this is the case also for ASP implementations. In order to take advantage of this fact, one should be able to select automatically the "best" solver on the basis of the characteristics (called *features*) of the instance in input, i.e., one has to consider to solve an *algorithm selection problem* (Rice 1976).

Inspired by the successful attempts (Gomes and Selman 2001; O'Mahony et al. 2008; Pulina and Tacchella 2009; Xu et al. 2008) done in the neighbor fields of SAT, QSAT and CSP, the application of algorithm selection techniques to ASP solving was ignited by the release of the portfolio solver CLASPFOLIO (Gebser et al. 2011). This solver imports into ASP the SATZILLA (Xu et al. 2008) approach. Indeed, CLASPFOLIO employs inductive techniques based on *regression* to choose the "best" configuration/heuristic of the solver CLASP. The complete picture of inductive approaches applied to ASP solving includes also techniques for learning heuristics orders (Balduccini 2011), solutions to combine portfolio and automatic algorithm configuration approaches (Silverthorn, Lierler, and Schneider 2012), automatic selection of a scheduling of ASP solvers (Hoos et al. 2012) (in this case CLASP configurations), and the multi-engine approach. The aim of a multi-engine solver (Pulina and Tacchella 2009) is to select the "best" solver among a set of efficient ones used as *black-box engines*. The multi-engine ASP solver ME-ASP was proposed in (Maratea, Pulina, and Ricca 2012b), and ports to ASP an approach applied before to QBF (Pulina and Tacchella 2009).

ME-ASP exploits inductive techniques based on *classification* to choose, on a per instance basis, an engine among a selection of black-box heterogeneous ASP solvers. The first implementation of ME-ASP, despite not being highly optimized, already reached good performance. Indeed, ME-ASP can combine the strengths of its component engines, and thus it performs well on a broad set of benchmarks including 14 domains and 1462 ground instances (detailed results are reported in (Maratea, Pulina, and Ricca 2014a)).

In this paper we report on (i) a new optimized implementation of ME-ASP; and on (ii) a first attempt of applying algorithm selection to the entire process of computing answer sets of non-ground programs.

As a matter of fact, the ASP solutions available at the state of the art employing machine-learning techniques are devised to solve ground (or propositional) programs, and – to the best of our knowledge – no solution has been proposed that is able to cope directly with non-ground programs. Note that ASP programmers almost always write non-ground programs, which have to be first instantiated by a grounder. It is well-known that such instantiation phase can influence significantly the performance of the entire solving process. At the time of this writing, there are two prominent alternative implementations that are able to instantiate ASP programs: DLV (Leone et al. 2006) and GRINGO (Gebser, Schaub, and Thiele 2007). Once the peculiarities of the instantiation process are properly taken into account, both implementations can be combined in a multi-engine grounder by applying also to this phase an algorithm selection recipe, building on (Maratea, Pulina, and Ricca 2013). The entire process

of evaluation of a non-ground ASP program can be, thus, obtained by applying algorithm selection to the instantiation phase, selecting either DLV or GRINGO; and, then, in a subsequent step, evaluating the propositional program obtained in the first step with a multi-engine solver.

An experimental analysis reported in the paper shows that (i) the new implementation of ME-ASP is substantially faster than the previous version; and (ii) the straight application of the multi-engine recipe to the instantiation phase is already beneficial. At the same time, it remains space for future work, and in particular for devising more specialized techniques to exploit the full potential of the approach.

A Multi-Engine ASP system

We next overview the components of the multi-engine approach, and we report on the way we have instantiated it to cope with instantiation and solving, thus obtaining a complete multi-engine system for computing answer sets of non-ground ASP programs.

General Approach. The design of a multi-engine solver based on classification is composed of three main ingredients: (i) a set of features that are significant for classifying the instances; (ii) a selection of solvers that are representative of the state of the art and complementary; and (iii) a choice of effective classification algorithms. Each instance in a fairly-designed *training set* of instances is analyzed by considering both the features and the performance of each solvers. An inductive model is computed by the classification algorithm during this phase. Then, each instance in a *test set* is processed by first extracting its features, and the solver is selected starting from these features using the learned model. Note that, this schema does not make any assumption (other than the basic one of supporting a common input) on the engines.

The ME-ASP solver. In (Maratea, Pulina, and Ricca 2012b; 2014a) we described the choices we have made to develop the ME-ASP solver. In particular, we have singled out a set of syntactic features that are both significant for classifying the instances and cheap-to-compute (so that the classifier can be fast and accurate). In detail, we considered: the number of rules and number of atoms, the ratio of horn, unary, binary and ternary rules, as well as some ASP peculiar features, such as the number of true and disjunctive facts, and the fraction of normal rules and constraints. The number of resulting features, together with some of their combinations, amounts to 52. In order to select the engines we ran preliminary experiments (Maratea, Pulina, and Ricca 2014a) to collect a pool of solvers that is representative of the state-of-the-art solver (SOTA), i.e., considering a problem instance, the oracle that always fares the best among the solvers that entered the system track of the 3rd ASP Competition (Calimeri et al. 2011), plus DLV. The pool of engines collected in ME-ASP is composed of 5 solvers, namely CLASP, CLASPD, CMODELS, DLV, and IDP, as submitted to the 3rd ASP Competition. We experimented with several classification algorithms (Maratea, Pulina, and Ricca 2014a), and proved empirically that ME-ASP can perform better than its engines with any choice. Nonetheless, we selected the k-nearest neighbor (kNN) classifier for our new implementation: it was already used in ME-ASP (Maratea, Pulina, and Ricca 2012b), with good performance, and it was easy to integrate its implementation in the new version of the system.

Multi-engine instantiator. Concerning the automatic selection of the grounder, we selected: number of disjunctive rules, presence of queries, the total amount of functions and predicates, number of strongly connected and Head-Cycle Free(Ben-Eliyahu and Dechter 1994) components, and stratification property, for a total amount of 11 features. These features are able to discriminate the class of the problem, and are also pragmatically cheap-to-compute. Indeed, given the high expressivity of the language, non-ground ASP programs (which are usually written by programmers) contain only a few rules. Concerning the grounders, given that there are only two alternative solutions, namely DLV and GRINGO, we considered both for our implementation.

Concerning the classification method, we used an implementation of the PART decision list generator (Frank and Witten 1998), a classifier that returns a human readable model based on if-then-else rules. We used PART because, given the relatively small total amount of features related to the non-ground instances, it allows us to compare the generated model with respect to the knowledge of a human expert.

Multi-Engine System ME-ASP^{gr}. Given a (non-ground) ASP program, the evaluation workflow of the multi-engine ASP solution called ME-ASP^{gr} is the following: (i) non-ground features extraction, (ii) grounder selection, (iii) grounding phase, (iv) ground features extraction, (v) solver selection, and (vi) solving phase on ground program.

Implementation and Experiments

In this section we report the results of two experiments conceived to assess the performance of the new versions of the ME-ASP system. The first experiment has the goal of measuring the performance improvements obtained by the new optimized implementation of the ME-ASP solver. The second experiment assesses ME-ASP^{gr} and reports on the performance improvements that can be obtained by selecting the grounder first and then calling the ME-ASP solver. ME-ASP and ME-ASP^{gr} are available for download at `www.mat.unical.it/ricca/me-asp`. Concerning the hardware employed and the execution settings, all the experiments run on a cluster of Intel Xeon E31245 PCs at 3.30 GHz equipped with 64 bit Ubuntu 12.04, granting 600 seconds of CPU time and 2GB of memory to each solver. The benchmarks used in this paper belong to the suite of benchmarks, encoded in the ASP-Core 1.0 language, of the 3rd ASP Competition. Note that in the 4th ASP Competition (Alviano et al. 2013) the new language ASP-Core 2.0 has been introduced. We still rely on the language of the 3rd ASP Competition given that the total amount of solvers and grounders supporting the new standard language is very limited with respect to the number of tools supporting ASP-Core 1.0.

Assessment of the new implementation of ME-ASP. The original implementation of ME-ASP was obtained by combining a general purpose feature extractor (that we have

initially developed for experimenting with a variety of additional features) developed in Java, with a collection of Perl scripts linking the other components of the system, which are based on the *rapidminer* library. This is a general purpose implementation supporting also several classification algorithms. Since the CPU time spent for the extraction of features and solver selection has to be made negligible, we developed an optimized version of ME-ASP. The goal was to optimize the interaction among system components and further improve their efficiency. To this end, we have re-engineered the feature extractor, enabling it to read ground instances expressed in the numeric format used by GRINGO. Furthermore, we have integrated it with an implementation of the kNN algorithm built on top of the ANN library (`www.cs.umd.edu/~mount/ANN`) in the same binary developed in C++. This way the new implementation minimizes the overhead introduced by solver selection.

We now present the results of an experiment in which we compare the old implementation of ME-ASP, labelled ME-ASP^{old}, with the new one, labelled ME-ASP^{new}. In this experiment, assessing solving performance, we used GRINGO as grounder for both implementations, and we considered problems belonging to the *NP* and *Beyond NP* classes of the competition (i.e., the grounder and domains considered by ME-ASP^{old} (Maratea, Pulina, and Ricca 2014a)). The inductive model used in ME-ASP^{new} was the same used in ME-ASP^{old} (details are reported in (Maratea, Pulina, and Ricca 2014a)). The plot in Figure 1 (top) depicts the performance of both ME-ASP^{old} and ME-ASP^{new} (dotted red and solid blue lines in the plot, respectively). Considering the total amount of *NP* and *Beyond NP* instances evaluated at the 3rd ASP Competition (140), ME-ASP^{new} solved 92 instances (77 *NP* and 15 *Beyond NP*) in about 4120 seconds, while ME-ASP^{old} solved 77 instances (62 *NP* and 15 *Beyond NP*) in about 6498 seconds. We report an improvement both in the total amount of solved instances (ME-ASP^{new} is able to solve 66% of the whole set of instances, while ME-ASP^{new} stops at 51%) and in the average CPU time of solved instances (about 45 seconds against 84).

The improvements of ME-ASP^{new} are due to its optimized implementation. Once feature extraction and solver selection are made very efficient, it is possible to extract features for more instances and the engines are called in advance w.r.t. what happens in ME-ASP^{old}. This results in more instances that are processed and solved by ME-ASP^{new} within the timeout.

Assessment of the complete system. We developed a preliminary implementation of a grounder selector, which combines a feature extractor for non-ground programs written in Java, and an implementation of the PART decision list generator, as mentioned in the previous section. The grounder selector is then combined with ME-ASP^{new}.

We now present the results of an experiment in which we compare ME-ASP^{gr} with ME-ASP^{new}, and the SOTA solver. ME-ASP^{new} coupled with DLV (resp. GRINGO) is denoted by ME-ASP^{new} (dlv) (resp. ME-ASP^{new} (gringo)). In this case we considered all the benchmark problems of the 3rd ASP Competition, including the ones belonging to the *P* class. Indeed, in this case we are interested also in

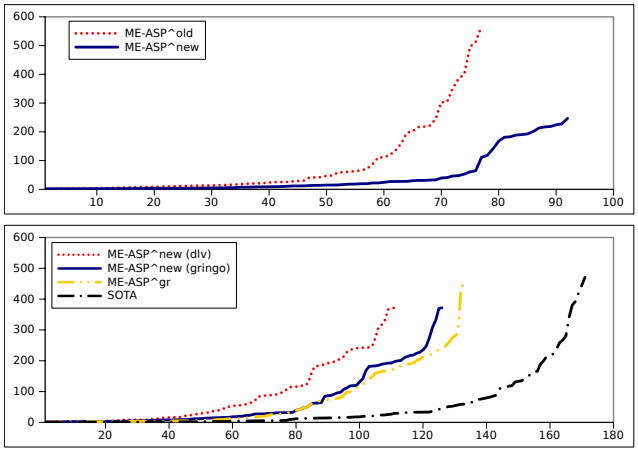


Figure 1: Performance of ME-ASP^{old} and ME-ASP^{new} on *NP* and *Beyond NP* instances evaluated at the 3rd ASP Competition (top); performance of ME-ASP^{gr}, its engines and SOTA on the complete set of instances evaluated at the 3rd ASP Competition (bottom). In the x-axis it is shown the total amount of solved instances, while y-axis reports the CPU time in seconds.

grounders' performance, which is crucial in the *P* class.

The plot in Figure 1 (bottom) shows the performance of the aforementioned solvers. In the plot, we depict the performance of ME-ASP^{new} (dlv) with a red dotted line, ME-ASP^{new} (gringo) with a solid blue line, ME-ASP^{gr} with a double dotted dashed yellow line, and, finally, with a dotted dashed black line we denote the performance of the SOTA solver. Looking at the plot, we can see that ME-ASP^{new} (gringo) solves more instances that ME-ASP^{new} (dlv) – 126 and 111, respectively – while both are outperformed by ME-ASP^{gr}, that is able to solve 134 instances. The average CPU time of solved instances for ME-ASP^{new} (dlv), ME-ASP^{new} (gringo) and ME-ASP^{gr} is 86.86, 67.93 and 107.82 seconds, respectively. Looking at the bottom plot in Figure 1, concerning the performance of the SOTA solver, we report that it is able to solve 173 instances out of a total of 200 instances (evaluated at the 3rd ASP Competition), highlighting room for further improving this preliminary version of ME-ASP^{gr}. Indeed, the current classification model predicts GRINGO for most of the *NP* instances, but having a more detailed look at the results, we notice that CLASP and IDP with GRINGO solve both 72 instances, while using DLV they solve 93 and 92 instances, respectively. A detailed analysis of the performance of the various ASP solvers with both grounders can be found in (Maratea, Pulina, and Ricca 2013).

It is also worth mentioning that the output formats of GRINGO and DLV differ, thus there are combinations grounder/solver that require additional conversion steps in our implementation. Since the new feature extractor is designed to be compliant with the numeric format produced by GRINGO, if DLV is selected as grounder then the non-ground program is instantiated twice. Moreover, if DLV is selected as grounder, and it is not selected also as solver, the produced propositional program is fed in gringo to be

converted in numeric format. These additional steps, due to technical issues, result in a suboptimal implementation of the execution pipeline that could be further optimized in case both grounders would agree on a common output format.

Conclusion. In this paper we presented improvements to the multi-engine ASP solver ME-ASP. Experiments show that (i) the new implementation of ME-ASP is more efficient, and (ii) the straight application of the multi-engine recipe to the instantiation phase is already beneficial. Directions for future research include exploiting the full potential of the approach by predicting the pair grounder+solver, and importing policy adaptation techniques employed in (Maratea, Pulina, and Ricca 2014b).

Acknowledgments. This research has been partly supported by Regione Calabria under project PIA KnowRex POR FESR 2007- 2013 BURC n. 49 s.s. n. 1 16/12/2010, the Italian Ministry of University and Research under PON project "Ba2Know S.I.-LAB" n. PON03PE_0001, the Autonomous Region of Sardinia (Italy) and the Port Authority of Cagliari (Italy) under L.R. 7/2007, Tender 16 2011 project "DESCTOP", CRP-49656.

References

Rice, J. R. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming*, 1070–1080. Cambridge, Mass.: MIT Press.

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

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive Datalog. *ACM TODS* 22(3):364–418.

Frank, E., and Witten, I. H. 1998. Generating accurate rule sets without global optimization. In *ICML'98*, 144.

Marek, V. W., and Truszczyński, M. 1998. Stable models and an alternative logic programming paradigm. *CoRR* cs.LO/9809032.

Niemelä, I. 1998. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. In *CANR 98 Workshop*, 72–79.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence* 126(1-2):43–62.

Potsdam, U. since 2002. asparagus homepage. `http://asparagus.cs.uni-potsdam.de/`.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138:181–234.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Tempe, Arizona: CUP.

Lierler, Y. 2005. Disjunctive Answer Set Programming via Satisfiability. In *LPNMR 05*, LNCS 3662, 447–451.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL* 7(3):499–562.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *IJCAI-07*, 386–392.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo : A New Grounder for Answer Set Programming. In *LPNMR 2007*, LNCS 4483, 266–271.

Mariën, M.; Wittocx, J.; Denecker, M.; and Bruynooghe, M. 2008. Sat(id): Satisfiability of propositional logic extended with inductive definitions. In *SAT 08*, LNCS, 211–224.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *ICAICS 08*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR* 32:565–606.

Janhunen, T.; Niemelä, I.; and Sevalnev, M. 2009. Computing stable models via reductions to difference logic. In *LPNMR 09*, LNCS, 142–154.

Pulina, L., and Tacchella, A. 2009. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints* 14(1):80–116.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T.; Schneider, M. T.; and Ziller, S. 2011. A portfolio solver for answer set programming: Preliminary report. In *LPNMR 11*, LNCS 6645, 352–357.

Balduccini, M. 2011. Learning and using domain-specific heuristics in ASP solvers. *AICOM* 24(2):147–164.

Ben-Eliyahu R.; Dechter R. 1994. Propositional Semantics for Disjunctive Logic Programs *Annals of Mathematics and Artificial Intelligence*. 12:53–87, Science Publishers.

Calimeri, F.; Ianni, G.; Ricca, F.; et al. 2011. The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In *Proc. of LPNMR11.*, 388–403 LNCS.

Hoos, H.; Kaminski, R.; Schaub, T.; and Schneider, M. T. 2012. ASPeed: Asp-based solver scheduling. In *Tech. Comm. of ICLP 2012*, volume 17 of *LIPIcs*, 176–187.

Maratea, M.; Pulina, L.; and Ricca, F. 2012b. The multi-engine asp solver me-asp. In *JELIA 2012.*, LNCS 7519, 484–487.

Silverthorn, B.; Lierler, Y.; and Schneider, M. 2012. Surviving solver sensitivity: An asp practitioner's guide. In *Tech. Comm. of ICLP 2012*, volume 17 of *LIPIcs*, 164–175.

Alviano, M.; Calimeri, F.; Charwat, G.; et al. 2013. The fourth answer set programming competition: Preliminary report. In *LPNMR*, LNCS 8148, 42–53.

Maratea, M.; Pulina, L.; and Ricca, F. 2013. Automated selection of grounding algorithm in answer set programming. In *AI* IA 2013*. International Publishing. 73–84.

Maratea, M.; Pulina, L.; and Ricca, F. 2014a. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*. DOI: `http://dx.doi.org/10.1017/S1471068413000094`

Maratea, M.; Pulina, L.; and Ricca, F. 2014b. Multi-engine asp solving with policy adaptation. *JLC*. In Press.

Preliminary Report on WASP 2.0*

Mario Alviano, Carmine Dodaro and Francesco Ricca
Department of Mathematics and Computer Science, University of Calabria, Italy
{alviano,dodaro,ricca}@mat.unical.it

Abstract

Answer Set Programming (ASP) is a declarative programming paradigm. The intrinsic complexity of the evaluation of ASP programs makes the development of more effective and faster systems a challenging research topic. This paper reports on the recent improvements of the ASP solver WASP. WASP is undergoing a refactoring process which will end up in the release of a new and more performant version of the software. In particular the paper focus on the improvements to the core evaluation algorithms working on normal programs. A preliminary experiment on benchmarks from the 3rd ASP competition belonging to the NP class is reported. The previous version of WASP was often not competitive with alternative solutions on this class. The new version of WASP shows a substantial increase in performance.

Introduction

Answer Set Programming (ASP) (Gelfond and Lifschitz 1991) is a declarative programming paradigm which has been proposed in the area of non-monotonic reasoning and logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use a solver to find them.

Despite the intrinsic complexity of the evaluation of ASP, after twenty years of research many efficient ASP systems have been developed. (e.g. (Alviano et al. 2011; Gebser et al. 2007; Lierler and Maratea 2004)). The availability of robust implementations made ASP a powerful tool for developing advanced applications in the areas of Artificial Intelligence, Information Integration, and Knowledge Management. These applications of ASP have confirmed the viability of the use of ASP. Nonetheless, the interest in developing more effective and faster systems is still a crucial and challenging research topic, as witnessed by the results of the ASP Competition series (see e.g. (Calimeri, Ianni, and Ricca 2014)).

*This research has been partly supported by the European Commission, European Social Fund of Regione Calabria, the Regione Calabria under project PIA KnowRex POR FESR 2007- 2013 BURC n. 49 s.s. n. 1 16/12/2010, and the Italian Ministry of University and Research under PON project "Ba2Know S.I.-LAB" n. PON03PE_0001.

This paper reports on the recent improvements of the ASP solver for propositional programs WASP (Alviano et al. 2013). The new version of WASP is inspired by several techniques that were originally introduced for SAT solving, like the Davis-Putnam-Logemann-Loveland (DPLL) backtracking search algorithm (Davis, Logemann, and Loveland 1962), *clause learning* (Zhang et al. 2001), *backjumping* (Gaschnig 1979), *restarts* (Gomes, Selman, and Kautz 1998), and *conflict-driven heuristics* (Moskewicz et al. 2001). The mentioned SAT-solving methods have been adapted and combined with state-of-the-art pruning techniques adopted by modern native ASP solvers (Alviano et al. 2011; Gebser et al. 2007). In particular, the role of Boolean Constraint Propagation in SAT-solvers is taken by a procedure combining the *unit propagation* inference rule with inference techniques based on ASP program properties. In particular, support inferences are implemented via Clark's completion, and the implementation of the well-founded operator is based on source pointers (Simons, Niemelä, and Soininen 2002).

In the following, we overview the techniques implemented by the 2.0 version of WASP, focusing on the improvements to the core evaluation algorithms working on normal programs. Then we compare the new implementation with the previous one.

We also report on a preliminary experiment in which we compare the old and new versions of WASP with the latest version of clasp, which is the solver that won the 3rd and 4th edition of the ASP competition. Benchmarks were taken from the 3rd ASP competition and belong to the NP class, i.e., the class of problems where the previous version of WASP was often not competitive with alternative solutions. The result show that WASP 2.0 is substantially faster than WASP 1.0 and is often competitive with clasp.

ASP Language

Let $\mathcal{A}$ be a countable set of propositional atoms. A *literal* is either an atom (a positive literal), or an atom preceded by the *negation as failure* symbol $\sim$ (a negative literal). The complement of a literal ℓ is denoted $\overline{\ell}$, i.e., $\overline{a} = \sim a$ and $\overline{\sim a} = a$ for an atom a. This notation extends to sets of literals, i.e., $\overline{L} := \{\overline{\ell} \mid \ell \in L\}$ for a set of literals L.

A *program* is a finite set of rules of the following form:

$$a_0 \text{ :- } a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \quad (1)$$

where $n \geq m \geq 0$ and each a_i $(i = 0, \ldots, n)$ is an atom. The atom a_0 is called head, and the conjunction $a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$ is referred to as body. Rule r is said to be regular if $H(r) \neq \bot$, where $\bot$ is a fixed atom in $\mathcal{A}$, and a constraint otherwise. For a rule r of the form (1), the following notation is also used: $H(r)$ denotes the head atom a_0; $B(r)$ denotes the set $\{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$ of body literals; $B^+(r)$ and $B^-(r)$ denote the set of atoms appearing in positive and negative body literals, respectively; $C(r) := H(r) \cup \overline{B(r)}$ is the clause representation of r.

An *interpretation* I is a set of literals, i.e., $I \subseteq \mathcal{A} \cup \overline{\mathcal{A}}$. Intuitively, literals in I are true, literals whose complements are in I are false, and all other literals are undefined. I is total if there are no undefined literals, and I is inconsistent if $\bot \in I$ or there is $a \in \mathcal{A}$ such that $\{a, \sim a\} \subseteq I$. An interpretation I satisfies a rule r if $C(r) \cap I \neq \emptyset$, while I violates r if $C(r) \subseteq \overline{I}$. A *model* of a program $\mathcal{P}$ is a consistent, total interpretation satisfying all rules of $\mathcal{P}$. The semantics of a program $\mathcal{P}$ is given by the set of its *answer sets* (or stable models) (Gelfond and Lifschitz 1991), where an interpretation I is an answer set for $\mathcal{P}$ if I is a subset-minimal model of the reduct $\mathcal{P}^I$ obtained by deleting from $\mathcal{P}$ each rule r such that $B^-(r) \cap I \neq \emptyset$, and then by removing all the negative literals from the remaining rules.

Answer Set Computation in WASP 2.0

In this section we review the algorithms implemented in WASP 2.0. The presentation is properly simplified to focus on the main principles.

Completion and Program Simplification

The first step of the evaluation in WASP 2.0 is a program transformation step. The input program first undergoes a Clark's completion transformation step, and then is simplified applying techniques in the style of satelite (Eén and Biere 2005). Given a rule $r \in \mathcal{P}$, let aux_r denote a fresh atom, i.e., an atom not appearing elsewhere. The completion of $\mathcal{P}$, denoted $Comp(\mathcal{P})$, consists of the following clauses:

- $\{\sim a, aux_{r_1}, \ldots, aux_{r_n}\}$ for each atom a occurring in $\mathcal{P}$, where $r_1, \ldots, r_n$ are the rules of $\mathcal{P}$ whose head is a;
- $\{H(r), \sim aux_r\}$ and $\{aux_r\} \cup \overline{B(r)}$ for each rule $r \in \mathcal{P}$;
- $\{\sim aux_r, \ell\}$ for each $r \in \mathcal{P}$ and $\ell \in B(r)$.

After the computation of Clark's completion, *simplification* techniques are applied (Eén and Biere 2005). These consist of polynomial algorithms for strengthening and for removing redundant clauses, and also include atoms elimination by means of clause rewriting.

Main Algorithm

An answer set of a given propositional program $Comp(\mathcal{P})$ is computed in WASP 2.0 by using Algorithm 1, which is similar to the DPLL procedure in SAT solvers. Initially, interpretation I is set to $\{\sim\bot\}$. Function Propagate (line 2) extends

Algorithm 1: Compute Answer Set

```
   Input  : An interpretation I for a program Comp(P)
   Output: An answer set for Comp(P) or Incoherent
1  begin
2      while Propagate(I) do
3          if I is total then
4              return I;
5          ℓ := ChooseUndefinedLiteral();
6          I' := ComputeAnswerSet(I ∪ {ℓ});
7          if I' ≠ Incoherent then
8              return I';
9          if there are violated (learned) clauses then
10             return Incoherent;
11     AnalyzeConflictAndLearnClauses(I);
12     return Incoherent;
```

Function Propagate(I)

```
1  while UnitPropagation(I) do
2      if not WellFoundedPropagation(I) then
3          return true;
4  return false;
```

I with those literals that can be deterministically inferred. This function returns false if an inconsistency (or conflict) is detected, true otherwise. When no inconsistency is detected, interpretation I is returned if total (lines 2–3). Otherwise, an undefined literal, say ℓ, is chosen according to some heuristic criterion (line 5). Then computation then proceeds with a recursive call to ComputeAnswerSet on $I \cup \{\ell\}$ (line 6). In case the recursive call returns an answer set, the computation ends returning it (lines 7–8). Otherwise, the algorithm unrolls choices until consistency of I is restored (backjumping; lines 9–10), and the computation resumes by propagating the consequences of the clause learned by the conflict analysis. Conflicts detected during propagation are analyzed by procedure *AnalyzeConflictAndLearnClauses* (line 11).

The main algorithm is usually complemented with some heuristic techniques that control the number of learned clauses (which may be exponential in number), and possibly restart the computation to explore different branches of the search tree. Moreover, a crucial role is played by the heuristic criteria used for selecting branching literals. WASP 2.0 adopts the same branching and deletion heuristics of the SAT solver MiniSAT (Eén and Sörensson 2003). The restart policy is based on the sequence of thresholds introduced in (Luby, Sinclair, and Zuckerman 1993).

Propagation and clause learning are described in more detail in the following.

Propagation. WASP 2.0 implements two deterministic inference rules for pruning the search space during answer set computation. These propagation rules are named *unit* and *well-founded*. Unit propagation is applied first (line 1

of function Propagate). It returns false if an inconsistency arises. Otherwise, well-founded propagation is applied (line 2). Function WellFoundedPropagation may learn an implicit clause in P, in which case true is returned and unit propagation is applied on the new clause. When no new clause can be learned by WellFoundedPropagation, function Propagate returns true to report that no inconsistency has been detected. More in details, unit propagation is as in SAT solvers: An undefined literal ℓ is inferred by unit propagation if there is a rule r that can be satisfied only by ℓ, i.e., r is such that $\ell \in C(r)$ and $C(r) \setminus \{\ell\} \subseteq \overline{I}$. Concerning well-founded propagation, we must first introduce the notion of unfounded set. A set X of atoms is *unfounded* if for each rule r such that $H(r) \cap X \neq \emptyset$, at least one of the following conditions is satisfied: (i) $\overline{B(r)} \cap I \neq \emptyset$; (ii) $B^+(r) \cap X \neq \emptyset$; (iii) $I \cap H(r) \setminus X \neq \emptyset$. Intuitively, atoms in X can have support only by themselves. When an unfounded set X is found, function WellFoundedPropagation learns a clause forcing falsity of an atom in X. Clauses for other atoms in X will be learned on subsequent calls to the function, unless an inconsistency arises during unit propagation. In case of inconsistencies, indeed, the unfounded set X is recomputed.

Conflict Analysis and Learning. Clause learning acquires information from conflicts in order to avoid exploring the same search branch several times. WASP 2.0 adopts a learning schema based on the concept of the first Unique Implication Point (UIP) (Moskewicz et al. 2001), which is computed by analyzing the so-called implication graph. Roughly, the implication graph contains a node for each literal in I, and arcs from ℓ_i to ℓ_0 ($i = 1, \ldots, n$; $n \geq 1$) if literal ℓ_0 is inferred by unit propagation on clause $\{\ell_0, \ldots, \ell_n\}$. Each literal $\ell \in I$ is associated with a *decision level*, corresponding to the depth nesting level of the recursive call to ComputeAnswerSet on which ℓ is added to I. A node n in the implication graph is a UIP for a decision level d if all paths from the choice of level d to the conflict literals pass through n. The first UIP is the UIP for the decision level of the conflict that is closest to the conflict. The learning schema is as follows: Let u be the first UIP. Let L be the set of literals different form u occurring in a path from u to the conflict literals. The learned clause comprises u and each literal ℓ such that the decision level of ℓ is lower than the one of u and there is an arc $(\overline{\ell}, \ell')$ in the implication graph for some $\ell' \in L$.

Comparing WASP 1.0 and WASP 2.0

In this section we compare WASP 2.0 to WASP 1.0. First of all we observe that WASP 1.0 does not implement any program transformation phase, whereas WASP 2.0 applies both Clark's completion and program simplification in the style of (Eén and Biere 2005). The addition of this preprocessing step brings advantages in both terms of simplifying the implementation of the propagation procedure and in terms performance. The Clark's completion introduces a number of clauses that represent support propagation, which is implemented natively in WASP 1.0 instead. The subsequent program simplification step optimizes the program by eliminating redundant atoms (also introduced by the completion) and shrinking definitions. This results in a program that is usually easier to evaluate. Concerning the well-founded operator both WASP 2.0 and WASP 1.0 compute unfounded sets according to the *source pointers* (Simons, Niemelä, and Soininen 2002) technique. WASP 1.0, which implements a native inference rule, immediately infers unfounded atoms as false, and updates a special implementation of the implication graph. In contrast, WASP 2.0 learns a clause representing the inference (also called loop formula) and propagates it with unit propagation. This choice combined with Clark's completion allows to simplify conflict analysis, learning and backjumping. Indeed, WASP 1.0 implements specialized variants of these procedures that require the usage of complex data structures that are difficult to optimize. Since in WASP 2.0 literals are always inferred by the UnitPropagation procedure, we could adopt an implementation of these strategies optimized as in modern SAT solvers. Finally both WASP 2.0 and WASP 1.0 implement conflict-driven branching heuristics. WASP 2.0 uses a branching heuristic inspired to the one of MiniSAT, while WASP 1.0 uses an extension of the BerkMin (Goldberg and Novikov 2002) heuristics extended by adding a look-ahead technique and an additional ASP-specific criterion.

Experiment

In this section we report the results of an experiment assessing the performance of WASP 2.0. In particular, we compare WASP 2.0 with WASP 1.0 and clasp. All the solvers used gringo 3.0.5 (Gebser et al. 2011) as grounder. clasp and WASP 1.0 has been executed with the same heuristic setting used in (Alviano et al. 2013). Concerning clasp we used the version 3.0.1. The experiment was run on a Mac Pro equipped with two 3 GHz Intel Xeon X5365 (quad core) processors, with 4 MB of L2 cache and 16 GB of RAM, running Debian Linux 7.3 (kernel ver. 3.2.0-4-amd64). Binaries were generated with the GNU C++ compiler 4.7.3-4 shipped by Debian. Time limit was set to 600 seconds. Performance was measured using the tools pyrunlim and pyrunner (*https://github.com/alviano/python*).

Tested instances are among those in the System Track of the 3rd ASP Competition (Calimeri, Ianni, and Ricca 2014), in particular all instances in the NP category. This category includes planning domains, temporal and spatial scheduling problems, combinatorial puzzles, graph problems, and a number of real-world domains in which ASP has been applied. (See (Calimeri, Ianni, and Ricca 2014) for an exhaustive description of the benchmarks.)

Table 1 summarizes the number of solved instances and the average running times in seconds for each solver. In particular, the first two columns report the total number of instances (#) and the number of instances that are solved by all solvers ($\#_{all}$), respectively; the remaining columns report the number of solved instances within the time-out (sol.), and the running times averaged both over solved instances (t) and over instances solved by all variants (t_{all}).

We observe that WASP 2.0 outperforms WASP 1.0. In fact, WASP 2.0 solved 17 instances more than WASP 1.0,

Table 1: Average running time and number of solved instances

			clasp			WASP 1.0			WASP 2.0		
Problem	**#**	**#$_{all}$**	**sol.**	**t**	**t$_{all}$**	**sol.**	**t**	**t$_{all}$**	**sol.**	**t**	**t$_{all}$**
DisjunctiveScheduling	10	5	5	16.8	16.8	5	29.0	29.0	5	188.4	188.4
GraphColouring	10	3	4	88.0	20.6	3	50.5	50.5	3	3.3	3.3
HanoiTower	10	2	7	126.0	49.8	2	214.0	214.0	7	52.5	18.3
KnightTour	10	6	10	14.3	0.3	6	93.5	93.5	10	16.0	0.6
Labyrinth	10	8	9	74.4	74.7	8	118.7	118.7	10	85.8	84.7
MazeGeneration	10	10	10	0.3	0.3	10	19.9	19.9	10	2.7	2.7
MultiContextSystemQuerying	10	10	10	5.1	5.1	10	122.4	122.4	10	9.4	9.4
Numberlink	10	6	8	21.1	0.6	6	24.3	24.3	7	8.7	5.5
PackingProblem	10	0	0	-	-	0	-	-	0	-	-
SokobanDecision	10	5	10	101.5	2.8	5	212.8	212.8	7	97.8	14.4
Solitaire	10	2	2	124.9	124.9	3	183.1	198.0	4	8.7	6.0
WeightAssignmentTree	10	1	5	119.2	22.4	1	297.3	297.3	3	282.3	97.9
Total	**120**	**58**	**80**	**62.9**	**20.5**	**59**	**124.1**	**95.6**	**76**	**68.7**	**34.6**

and also the improvement on the average execution time is sensible, with a percentage gain of around 64% on instances solved by all systems. On the other hand, clasp is faster than WASP 2.0, with a percentage gain of around 41 % on the same instances. Moreover, clasp solved 4 instances more than WASP 2.0.

Analyzing the results in more detail, there are some specific benchmarks where WASP 2.0 and clasp exhibit significantly performances. Two of these problems are SokobanDecision and WeightAssignmentTree, where clasp solved 3 and 2 instances more than WASP 2.0, respectively, while WASP 2.0 solved 2 instances more than clasp in Solitaire. We also note that the performance of WASP deteriored in DisjunctiveScheduling. This is due to the initial steps of the computation, and in particular to the simplification procedure, which in this case removes 80% of clauses and 99% of atoms. However, there are cases in which simplifications play a crucial role to improve performance of the answer set search procedure. For example, in HanoiTower, where WASP 2.0 performs better than other systems, more than half of the variables are removed in a few seconds.

Related Work

WASP 1.0 is inspired by several techniques used in SAT solving that were first introduced for Constraint Satisfaction and QBF solving.

Some of these techniques were already adapted in non-disjunctive ASP solvers like Smodels$_{cc}$ (Ward and Schlipf 2004), clasp (Gebser et al. 2007), Smodels (Simons, Niemelä, and Soininen 2002), Cmodels3 (Lierler and Maratea 2004), and DLV (Ricca, Faber, and Leone 2006). More in detail, WASP 2.0 differs from Cmodels3 (Lierler and Maratea 2004) that are based on a rewriting into a propositional formula and an external SAT solver. WASP 2.0 differs from DLV (Alviano et al. 2011) and the Smodels variants, which features a native implementation of all inference rules. Our new solver is more similar to clasp, but there are differences concerning the restart policy, constraint deletion and branching heuristics. WASP 2.0 adopts as default a policy based on the sequence of thresholds introduced in (Luby, Sinclair, and Zuckerman 1993), whereas clasp employs by default a different policy based on geometric series. Concerning deletion of learned constraints, WASP 2.0 adopts a criterion inspired by MiniSAT, while clasp implements a technique introduced in Glucose (Audemard and Simon 2009). Moreover, clasp adopts a branching heuristic based on BerkMin (Goldberg and Novikov 2002) with a variant of the MOMS criterion which estimates the effect of the candidate literals in short clauses.

Conclusion

In this paper we reported on the recent improvement of the ASP solver WASP 1.0. We described the main improvements on the evaluation procedure focusing on the improvements to the core evaluation algorithms working on normal programs. The new solver was compared with both its predecessor and the latest version of clasp on on benchmarks belonging to the NP class, where WASP 1.0 was not competitive. The result is very encouraging, since WASP 2.0 improves substantially w.r.t. WASP 1.0 and is often competitive with clasp.

Future work concerns the reengineering of disjunctive rules, aggregates, and weak constraints, as well as the introduction of a native implementation of choice rules.

References

Alviano, M.; Faber, W.; Leone, N.; Perri, S.; Pfeifer, G.; and Terracina, G. 2011. The disjunctive datalog system DLV. In Gottlob, G., ed., *Datalog 2.0*, volume 6702. Springer Berlin/Heidelberg. 282–301.

Alviano, M.; Dodaro, C.; Faber, W.; Leone, N.; and Ricca, F. 2013. Wasp: A native asp solver based on constraint learning. In Cabalar, P., and Son, T. C., eds., *LPNMR*, volume 8148 of *LNCS*, 54–66. Springer.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solvers. In Boutilier, C., ed., *IJCAI*, 399–404.

Calimeri, F.; Ianni, G.; and Ricca, F. 2014. The third open

answer set programming competition. *Theory and Practice of Logic Programming* 14(1):117–135.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem Proving. *Communications of the ACM* 5:394–397.

Eén, N., and Biere, A. 2005. Effective preprocessing in sat through variable and clause elimination. In *SAT*, volume 3569 of *LNCS*, 61–75. Springer.

Eén, N., and Sörensson, N. 2003. An extensible sat-solver. In Giunchiglia, E., and Tacchella, A., eds., *SAT*, volume 2919 of *LNCS*, 502–518. Springer.

Gaschnig, J. 1979. *Performance measurement and analysis of certain search algorithms*. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA. Technical Report CMU-CS-79-124.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *IJCAI*, 386–392. Morgan Kaufmann Publishers.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in *gringo* series 3. In Delgrande, J. P., and Faber, W., eds., *LPNMR*, volume 6645 of *LNCS*, 345–351. Springer.

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

Goldberg, E., and Novikov, Y. 2002. BerkMin: A Fast and Robust Sat-Solver. In *Design, Automation and Test in Europe Conference and Exposition (DATE 2002)*, 142–149. Paris, France: IEEE Computer Society.

Gomes, C. P.; Selman, B.; and Kautz, H. A. 1998. Boosting Combinatorial Search Through Randomization. In *Proceedings of AAAI/IAAI 1998*, 431–437. AAAI Press.

Lierler, Y., and Maratea, M. 2004. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In Lifschitz, V., and Niemelä, I., eds., *Proceedings of LPNMR*, volume 2923 of *LNAI*, 346–350. Springer.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.* 47:173–180.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th DAC*, 530–535. Las Vegas, NV, USA: ACM.

Ricca, F.; Faber, W.; and Leone, N. 2006. A Backjumping Technique for Disjunctive Logic Programming. 19(2):155–172.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138:181–234.

Ward, J., and Schlipf, J. S. 2004. Answer Set Programming with Clause Learning. In Lifschitz, V., and Niemelä, I., eds., *Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7)*, volume 2923 of *LNAI*, 302–313. Springer.

Zhang, L.; Madigan, C. F.; Moskewicz, M. W.; and Malik, S. 2001. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proceedings of the ICCAD*, 279–285.

On Strong and Default Negation in Logic Program Updates

Martin Slota
CENTRIA
New University of Lisbon

Martin Baláž
Faculty of Mathematics, Physics and Informatics
Comenius University

João Leite
CENTRIA
New University of Lisbon

Abstract

Existing semantics for answer-set program updates fall into two categories: either they consider only *strong negation* in heads of rules, or they primarily rely on *default negation* in heads of rules and optionally provide support for strong negation by means of a syntactic transformation.

In this paper we pinpoint the limitations of both these approaches and argue that both types of negation should be first-class citizens in the context of updates. We identify principles that plausibly constrain their interaction but are not simultaneously satisfied by any existing rule update semantics. Then we extend one of the most advanced semantics with direct support for strong negation and show that it satisfies the outlined principles as well as a variety of other desirable properties.

1 Introduction

The increasingly common use of rule-based knowledge representation languages in highly dynamic and information-rich contexts, such as the Semantic Web (Berners-Lee, Hendler, and Lassila 2001), requires standardised support for updates of knowledge represented by rules. Answer-set programming (Gelfond and Lifschitz 1988; 1991) forms the natural basis for investigation of rule updates, and various approaches to answer-set program updates have been explored throughout the last 15 years (Leite and Pereira 1998; Alferes et al. 1998; 2000; Eiter et al. 2002; Leite 2003; Sakama and Inoue 2003; Alferes et al. 2005; Banti et al. 2005; Zhang 2006; Šefránek 2006; Delgrande, Schaub, and Tompits 2007; Osorio and Cuevas 2007; Šefránek 2011; Krümpelmann 2012).

The most straightforward kind of conflict arising between an original rule and its update occurs when the original conclusion logically contradicts the newer one. Though the technical realisation and final result may differ significantly, depending on the particular rule update semantics, this kind of conflict is resolved by letting the newer rule prevail over the older one. Actually, under most semantics, this is also the *only* type of conflict that is subject to automatic resolution (Leite and Pereira 1998; Alferes et al. 2000; Eiter et al. 2002; Alferes et al. 2005; Banti et al. 2005; Delgrande, Schaub, and Tompits 2007; Osorio and Cuevas 2007).

From this perspective, allowing for both *strong* and *default negation* to appear in heads of rules is essential for an expressive and universal rule update framework (Leite 2003). While strong negation is the natural candidate here, used to express that an atom *becomes explicitly false*, default negation allows for more fine-grained control: the atom only *ceases to be true*, but its truth value may not be known after the update. The latter also makes it possible to move between any pair of epistemic states by means of updates, as illustrated in the following example:

Example 1.1 (Railway crossing (Leite 2003))**.** *Suppose that we use the following logic program to choose an action at a railway crossing:*

$$\mathsf{cross} \leftarrow \neg\mathsf{train}. \quad \mathsf{wait} \leftarrow \mathsf{train}. \quad \mathsf{listen} \leftarrow {\sim}\mathsf{train}, {\sim}\neg\mathsf{train}.$$

The intuitive meaning of these rules is as follows: one should cross *if there is evidence that no train is approaching;* wait *if there is evidence that a train is approaching;* listen *if there is no such evidence.*

Consider a situation where a train is approaching, represented by the fact $(\mathsf{train}.)$*. After this train has passed by, we want to update our knowledge to an epistemic state where we lack evidence with regard to the approach of a train. If this was accomplished by updating with the fact* $(\neg\mathsf{train}.)$*, we would cross the tracks at the subsequent state, risking being killed by another train that was approaching. Therefore, we need to express an update stating that all past evidence for an atom is to be removed, which can be accomplished by allowing default negation in heads of rules. In this scenario, the intended update can be expressed by the fact* $({\sim}\mathsf{train}.)$*.*

With regard to the support of negation in rule heads, existing rule update semantics fall into two categories: those that only allow for strong negation, and those that primarily consider default negation. As illustrated above, the former are unsatisfactory as they render many belief states unreachable by updates. As for the latter, they optionally provide support for strong negation by means of a syntactic transformation.

Two such transformations are known from the literature, both of them based on the principle of coherence: if an atom p is true, its strong negation $\neg p$ cannot be true simultaneously, so ${\sim}\neg p$ must be true, and also vice versa, if $\neg p$ is true, then so is ${\sim}p$. The first transformation, introduced in (Alferes and Pereira 1996), encodes this principle directly

by adding, to both the original program and its update, the following two rules for every atom p:

$$\sim\neg p \leftarrow p. \qquad\qquad \sim p \leftarrow \neg p.$$

This way, every conflict between an atom p and its strong negation $\neg p$ directly translates into two conflicts between the objective literals p, $\neg p$ and their default negations. However, the added rules lead to undesired side effects that stand in direct opposition with basic principles underlying updates. Specifically, despite the fact that the empty program does not encode any change in the modelled world, the stable models assigned to a program may change after an update by the empty program.

This undesired behaviour is addressed in an alternative transformation from (Leite 2003) that encodes the coherence principle more carefully. Nevertheless, this transformation also leads to undesired consequences, as demonstrated in the following example:

Example 1.2 (Faulty sensor)**.** *Suppose that we collect data from sensors and, for security reasons, multiple sensors are used to supply information about the critical fluent p. In case of a malfunction of one of the sensors, we may end up with an inconsistent logic program consisting of the following two facts:*

$$p. \qquad\qquad \neg p.$$

At this point, no stable model of the program exists and action needs to be taken to find out what is wrong. If a problem is found in the sensor that supplied the first fact $(p.)$, after the sensor is repaired, this information needs to be reset by updating the program with the fact $(\sim p.)$. Following the universal pattern in rule updates, where recovery from conflicting states is always possible, we expect that this update is sufficient to assign a stable model to the updated program. However, the transformational semantics for strong negation defined in (Leite 2003) still does not provide any stable model – we remain without a valid epistemic state when one should in fact exist.

In this paper we address the issues with combining strong and default negation in the context of rule updates. Based on the above considerations, we formulate a generic desirable principle that is violated by the existing approaches. Then we show how two distinct definitions of one of the most well-behaved rule update semantics (Alferes et al. 2005; Banti et al. 2005) can be equivalently extended with support for strong negation. The resulting semantics not only satisfies the formulated principle, but also retains the formal and computational properties of the original semantics. More specifically, our main contributions are as follows:

- based on Example 1.2, we introduce the *early recovery principle* that captures circumstances under which a stable model after a rule update should exist;
- we extend the *well-supported semantics for rule updates* (Banti et al. 2005) with direct support for strong negation;
- we define a fixpoint characterisation of the new semantics, based on the *refined dynamic stable model* semantics for rule updates (Alferes et al. 2005);
- we show that the defined semantics enjoy the early recovery principle as well as a range of desirable properties for rule updates known from the literature.

This paper is organised as follows: In Sect. 2 we present the syntax and semantics of logic programs, generalise the well-supported semantics from the class of normal programs to extended ones and define the rule update semantics from (Alferes et al. 2005; Banti et al. 2005). Then, in Sect. 3, we formally establish the early recovery principle, define the new rule update semantics for strong negation and show that it satisfies the principle. In Sect. 4 we introduce other established rule update principles and show that the proposed semantics satisfies them. We discuss our findings and conclude in Sect. 5.[1]

2 Background

In this section we introduce the necessary technical background and generalise the well-supported semantics (Fages 1991) to the class of extended programs.

2.1 Logic Programs

In the following we present the syntax of non-disjunctive logic programs with both strong and default negation in heads and bodies of rules, along with the definition of stable models of such programs from (Leite 2003) that is equivalent to the original definitions based on reducts (Gelfond and Lifschitz 1988; 1991; Inoue and Sakama 1998). Furthermore, we define an alternative characterisation of the stable model semantics: the well-supported models of normal logic programs (Fages 1991).

We assume that a countable set of propositional atoms $\mathcal{A}$ is given and fixed. An *objective literal* is an atom $p \in \mathcal{A}$ or its strong negation $\neg p$. We denote the set of all objective literals by $\mathcal{L}$. A *default literal* is an objective literal preceded by $\sim$ denoting default negation. A *literal* is either an objective or a default literal. We denote the set of all literals by $\mathcal{L}^*$. As a convention, double negation is absorbed, so that $\neg\neg p$ denotes the atom p and $\sim\sim l$ denotes the objective literal l. Given a set of literals S, we introduce the following notation: $S^+ = \{ l \in \mathcal{L} \mid l \in S \}$, $S^- = \{ l \in \mathcal{L} \mid \sim l \in S \}$, $\sim S = \{ \sim L \mid L \in S \}$.

An *extended rule* is a pair $\pi = (\mathsf{H}_\pi, \mathsf{B}_\pi)$ where H_π is a literal, referred to as the *head of* π, and B_π is a finite set of literals, referred to as the *body of* π. Usually we write π as $(\mathsf{H}_\pi \leftarrow \mathsf{B}_\pi^+, \sim\mathsf{B}_\pi^-.)$. A *generalised rule* is an extended rule that contains no occurrence of $\neg$, i.e., its head and body consist only of atoms and their default negations. A *normal rule* is a generalised rule that has an atom in the head. A *fact* is an extended rule whose body is empty and a *tautology* is any extended rule π such that $\mathsf{H}_\pi \in \mathsf{B}_\pi$. An *extended (generalised, normal) program* is a set of extended (generalised, normal) rules.

An *interpretation* is a consistent subset of the set of objective literals, i.e., a subset of $\mathcal{L}$ does not contain both p an $\neg p$ for any atom p. The satisfaction of an objective literal

[1] An extended version of this paper with all the proofs is available as (Slota, Baláž, and Leite 2014).

l, default literal $\sim l$, set of literals S, extended rule π and extended program P in an interpretation J is defined in the usual way: $J \models l$ iff $l \in J$; $J \models \sim l$ iff $l \notin J$; $J \models S$ iff $J \models L$ for all $L \in S$; $J \models \pi$ iff $J \models \mathsf{B}_\pi$ implies $J \models \mathsf{H}_\pi$; $J \models P$ iff $J \models \pi$ for all $\pi \in P$. Also, J is a *model of* P if $J \models P$, and P is *consistent* if it has a model.

Definition 2.1 (Stable model)**.** *Let P be an extended program. The set $[\![P]\!]_{\mathsf{SM}}$ of* stable models of P *consists of all interpretations J such that*

$$J^* = \mathsf{least}(P \cup \mathsf{def}(J))$$

where $\mathsf{def}(J) = \{ \sim l. \mid l \in \mathcal{L} \setminus J \}$, $J^* = J \cup \sim(\mathcal{L} \setminus J)$ *and* $\mathsf{least}(\cdot)$ *denotes the least model of the argument program in which all literals are treated as propositional atoms.*

A *level mapping* is a function that maps every atom to a natural number. Also, for any default literal $\sim p$, where $p \in \mathcal{A}$, and finite set of atoms and their default negations S, $\ell(\sim p) = \ell(p)$, $\ell^{\downarrow}(S) = \min\{\, \ell(L) \mid L \in S \,\}$ and $\ell^{\uparrow}(S) = \max\{\, \ell(L) \mid L \in S \,\}$.

Definition 2.2 (Well-supported model of a normal program)**.** *Let P be a normal program and ℓ a level mapping. An interpretation $J \subseteq \mathcal{A}$ is a* well-supported model of P w.r.t. ℓ *if the following conditions are satisfied:*

1. *J is a model of P;*
2. *For every atom $p \in J$ there exists a rule $\pi \in P$ such that*

$$\mathsf{H}_\pi = p \wedge J \models \mathsf{B}_\pi \wedge \ell(\mathsf{H}_\pi) > \ell^{\uparrow}(\mathsf{B}_\pi) \ .$$

The set $[\![P]\!]_{\mathsf{WS}}$ of well-supported models of P *consists of all interpretations $J \subseteq \mathcal{A}$ such that J is a well-supported model of P w.r.t. some level mapping.*

As shown in (Fages 1991), well-supported models coincide with stable models:

Proposition 2.3 ((Fages 1991))**.** *Let P be a normal program. Then, $[\![P]\!]_{\mathsf{WS}} = [\![P]\!]_{\mathsf{SM}}$.*

2.2 Well-supported Models for Extended Programs

The well-supported models defined in the previous section for normal logic programs can be generalised in a straightforward manner to deal with strong negation while maintaining their tight relationship with stable models (c.f. Proposition 2.3). This will come useful in Subsect. 2.3 and Sect. 3 when we discuss adding support for strong negation to semantics for rule updates.

We extend level mappings from atoms and their default negations to all literals: An *(extended) level mapping* ℓ maps every objective literal to a natural number. Also, for any default literal $\sim l$ and finite set of literals S, $\ell(\sim l) = \ell(p)$, $\ell^{\downarrow}(S) = \min\{\, \ell(L) \mid L \in S \,\}$ and $\ell^{\uparrow}(S) = \max\{\, \ell(L) \mid L \in S \,\}$.

Definition 2.4 (Well-supported model of an extended program)**.** *Let P be an extended program and ℓ a level mapping. An interpretation J is a* well-supported model of P w.r.t. ℓ *if the following conditions are satisfied:*

1. *J is a model of P;*
2. *For every objective literal $l \in J$ there exists a rule $\pi \in P$ such that*

$$\mathsf{H}_\pi = l \wedge J \models \mathsf{B}_\pi \wedge \ell(\mathsf{H}_\pi) > \ell^{\uparrow}(\mathsf{B}_\pi) \ .$$

The set $[\![P]\!]_{\mathsf{WS}}$ of well-supported models of P *consists of all interpretations J such that J is a well-supported model of P w.r.t. some level mapping.*

We obtain a generalisation of Prop. 2.3 to the class of extended programs:

Proposition 2.5. *Let P be an extended program. Then, $[\![P]\!]_{\mathsf{WS}} = [\![P]\!]_{\mathsf{SM}}$.*

2.3 Rule Updates

We turn our attention to rule updates, starting with one of the most advanced rule update semantics, the *refined dynamic stable models* for sequences of generalised programs (Alferes et al. 2005), as well as the equivalent definition of *well-supported models* (Banti et al. 2005). Then we define the transformations for adding support for strong negation to such semantics (Alferes and Pereira 1996; Leite 2003).

A rule update semantics provides a way to assign stable models to a pair or sequence of programs where each component represents an update of the preceding ones. Formally, a *dynamic logic program* (DLP) is a finite sequence of extended programs and by $\mathsf{all}(\boldsymbol{P})$ we denote the multiset of all rules in the components of $\boldsymbol{P}$. A rule update semantics S assigns a *set of S-models*, denoted by $[\![\boldsymbol{P}]\!]_{\mathsf{S}}$, to $\boldsymbol{P}$.

We focus on semantics based on the causal rejection principle (Leite and Pereira 1998; Alferes et al. 2000; Eiter et al. 2002; Leite 2003; Alferes et al. 2005; Banti et al. 2005; Osorio and Cuevas 2007) which states that a rule is *rejected* if it is in a direct conflict with a more recent rule. The basic type of conflict between rules π and σ occurs when their heads contain complementary literals, i.e. when $\mathsf{H}_\pi = \sim \mathsf{H}_\sigma$. Based on such conflicts and on a stable model candidate, a *set of rejected rules* can be determined and it can be verified that the candidate is indeed stable w.r.t. the remaining rules.

We define the most mature of these semantics, providing two equivalent definitions: the *refined dynamic stable models* (Alferes et al. 2005), or *RD-semantics*, defined using a fixpoint equation, and the *well-supported models* (Banti et al. 2005), or *WS-semantics*, based on level mappings.

Definition 2.6 (RD-semantics (Alferes et al. 2005))**.** *Let $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ be a DLP without strong negation. Given an interpretation J, the multisets of rejected rules* $\mathsf{rej}_{\geq}(\boldsymbol{P}, J)$ *and of default assumptions* $\mathsf{def}(\boldsymbol{P}, J)$ *are defined as follows:*

$$\mathsf{rej}_{\geq}(\boldsymbol{P}, J) = \{\pi \in P_i | i < n \wedge \exists j \geq i \ \exists \sigma \in P_j : \mathsf{H}_\pi = \sim \mathsf{H}_\sigma \wedge J \models \mathsf{B}_\sigma\},$$

$$\mathsf{def}(\boldsymbol{P}, J) = \{(\sim l.) | l \in \mathcal{L} \wedge \neg(\exists \pi \in \mathsf{all}(\boldsymbol{P}) : \mathsf{H}_\pi = l \wedge J \models \mathsf{B}_\pi)\}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{RD}}$ of RD-models of $\boldsymbol{P}$ *consists of all interpretations J such that*

$$J^* = \mathsf{least}\left([\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}_{\geq}(\boldsymbol{P}, J)] \cup \mathsf{def}(\boldsymbol{P}, J)\right)$$

where J^* and $\mathsf{least}(\cdot)$ *are defined as before.*

Definition 2.7 (WS-semantics (Banti et al. 2005)). *Let $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ be a DLP without strong negation. Given an interpretation J and a level mapping ℓ, the multiset of rejected rules* $\mathsf{rej}_\ell(\boldsymbol{P}, J)$ *is defined as follows:*

$$\mathsf{rej}_\ell(\boldsymbol{P}, J) = \{\pi \in P_i | i < n \wedge \exists j > i \; \exists \sigma \in P_j : \mathsf{H}_\pi = {\sim}\mathsf{H}_\sigma \wedge J \models \mathsf{B}_\sigma \wedge \ell(\mathsf{H}_\sigma) > \ell^\uparrow(\mathsf{B}_\sigma)\}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{WS}}$ *of* WS-models of $\boldsymbol{P}$ *consists of all interpretations J such that for some level mapping ℓ, the following conditions are satisfied:*

1. *J is a model of* $\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}_\ell(\boldsymbol{P}, J)$*;*
2. *For every $l \in J$ there exists some rule $\pi \in \mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}_\ell(\boldsymbol{P}, J)$ such that*

$$\mathsf{H}_\pi = l \wedge J \models \mathsf{B}_\pi \wedge \ell(\mathsf{H}_\pi) > \ell^\uparrow(\mathsf{B}_\pi) \; .$$

Unlike most other rule update semantics, these semantics can properly deal with tautological and other irrelevant updates, as illustrated in the following example:

Example 2.8 (Irrelevant updates). *Consider the DLP $\boldsymbol{P} = \langle P, U \rangle$ where programs P, U are as follows:*

$$\begin{array}{lll} P: & \mathsf{day} \leftarrow {\sim}\mathsf{night}. & \mathsf{stars} \leftarrow \mathsf{night}, {\sim}\mathsf{cloudy}. \\ & \mathsf{night} \leftarrow {\sim}\mathsf{day}. & {\sim}\mathsf{stars}. \\ U: & \mathsf{stars} \leftarrow \mathsf{stars}. & \end{array}$$

Note that program P has the single stable model $J_1 = \{\, \mathsf{day} \,\}$ and U contains a single tautological rule, i.e. it does not encode any change in the modelled domain. Thus, we expect that $\boldsymbol{P}$ also has the single stable model J_1. Nevertheless, many rule update semantics, such as those introduced in (Leite and Pereira 1998; Alferes et al. 2000; Eiter et al. 2002; Leite 2003; Sakama and Inoue 2003; Zhang 2006; Osorio and Cuevas 2007; Delgrande, Schaub, and Tompits 2007; Krümpelmann 2012), are sensitive to this or other tautological updates, introducing or eliminating models of the original program.

In this case, the unwanted model candidate is $J_2 = \{\, \mathsf{night}, \mathsf{stars} \,\}$ and it is neither an RD- nor a WS-model of $\boldsymbol{P}$, though the reasons for this are technically different under these two semantics. It is not difficult to verify that, given an arbitrary level mapping ℓ, the respective sets of rejected rules and the set of default assumptions are as follows:

$$\begin{aligned} \mathsf{rej}_{\geq}(\boldsymbol{P}, J_2) &= \{\, (\mathsf{stars} \leftarrow \mathsf{night}, {\sim}\mathsf{cloudy}.), ({\sim}\mathsf{stars}.) \,\}, \\ \mathsf{rej}_{\ell}(\boldsymbol{P}, J_2) &= \emptyset, \\ \mathsf{def}(\boldsymbol{P}, J_2) &= \{\, ({\sim}\mathsf{cloudy}.), ({\sim}\mathsf{day}.) \,\}. \end{aligned}$$

Note that $\mathsf{rej}_\ell(\boldsymbol{P}, J_2)$ is empty because, independently of ℓ, no rule π in U satisfies the condition $\ell(\mathsf{H}_\pi) > \ell^\uparrow(\mathsf{B}_\pi)$, so there is no rule that could reject another rule. Thus, the atom stars *belongs to J_2^* but does not belong to* $\mathsf{least}([\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}_{\geq}(\boldsymbol{P}, J_2)] \cup \mathsf{def}(\boldsymbol{P}, J_2))$*, so J_2 is not an RD-model of $\boldsymbol{P}$. Furthermore, no model of* $\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}_\ell(\boldsymbol{P}, J_2)$ *contains* stars*, so J_2 cannot be a WS-model of $\boldsymbol{P}$.*

Furthermore, the resilience of RD- and WS-semantics is not limited to empty and tautological updates, but extends to other irrelevant updates as well (Alferes et al. 2005; Banti et al. 2005). For example, consider the DLP $\boldsymbol{P}' = \langle P, U' \rangle$ where $U' = \{\, (\mathsf{stars} \leftarrow \mathsf{venus}.), (\mathsf{venus} \leftarrow \mathsf{stars}.) \,\}$. Though the updating program contains non-tautological rules, it does not provide a bottom-up justification of any model other than J_1 and, indeed, J_1 is the only RD- and WS-model of $\boldsymbol{P}'$.

We also note that the two presented semantics for DLPs without strong negation provide the same result regardless of the particular DLP to which they are applied.

Proposition 2.9 ((Banti et al. 2005)). *Let $\boldsymbol{P}$ be a DLP without strong negation. Then, $[\![\boldsymbol{P}]\!]_{\mathsf{WS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}}$.*

In case of the stable model semantics for a single program, strong negation can be reduced away by treating all objective literals as atoms and adding, for each atom p, the integrity constraint $(\leftarrow \; p, \neg p.)$ to the program (Gelfond and Lifschitz 1991). However, this transformation does not serve its purpose when adding support for strong negation to causal rejection semantics for DLPs because integrity constraints have empty heads, so according to these rule update semantics, they cannot be used to reject any other rule. For example, a DLP such as $\langle \{\, p., \neg p. \,\}, \{\, p. \,\} \rangle$ would remain without a stable model even though the DLP $\langle \{\, p., {\sim}p. \,\}, \{\, p. \,\} \rangle$ does have a stable model.

To capture the conflict between opposite objective literals l and $\neg l$ in a way that is compatible with causal rejection semantics, a slightly modified syntactic transformation can be performed, translating such conflicts into conflicts between objective literals and their default negations. Two such transformations have been suggested in the literature (Alferes and Pereira 1996; Leite 2003), both based on the principle of coherence. For any extended program P and DLP $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ they are defined as follows:

$$\begin{aligned} P^\dagger &= P \cup \{{\sim}\neg l \leftarrow l. | l \in \mathcal{L}\}, \\ \boldsymbol{P}^\dagger &= \left\langle P_i^\dagger \right\rangle_{i<n}, \\ P^\ddagger &= P \cup \{{\sim}\neg \mathsf{H}_\pi \leftarrow \mathsf{B}_\pi. | \pi \in P \wedge \mathsf{H}_\pi \in \mathcal{L}\}, \\ \boldsymbol{P}^\ddagger &= \left\langle P_i^\ddagger \right\rangle_{i<n}. \end{aligned}$$

These transformations lead to four possibilities for defining the semantics of an arbitrary DLP $\boldsymbol{P}$: $[\![\boldsymbol{P}^\dagger]\!]_{\mathsf{RD}}$, $[\![\boldsymbol{P}^\ddagger]\!]_{\mathsf{RD}}$, $[\![\boldsymbol{P}^\dagger]\!]_{\mathsf{WS}}$ and $[\![\boldsymbol{P}^\ddagger]\!]_{\mathsf{WS}}$. We discuss these in the following section.

3 Direct Support for Strong Negation in Rule Updates

The problem with existing semantics for strong negation in rule updates is that semantics based on the first transformation ($\boldsymbol{P}^\dagger$) assign too many models to some DLPs, while semantics based on the second transformation ($\boldsymbol{P}^\ddagger$) sometimes do not assign any model to a DLP that should have one. The former is illustrated in the following example:

Example 3.1 (Undesired side effects of the first transformation). *Consider the DLP $\boldsymbol{P}_1 = \langle P, U \rangle$ where $P = \{\, p., \neg p. \,\}$ and $U = \emptyset$. Since P has no stable model and U does not encode any change in the represented domain, it should follow that $\boldsymbol{P}_1$ has no stable model either. However, $[\![\boldsymbol{P}_1^\dagger]\!]_{\mathsf{RD}} =$*

$[\![\boldsymbol{P}_1^\dagger]\!]_{\mathsf{WS}} = \{\{p\},\{\neg p\}\}$*, i.e. two models are assigned to* $\boldsymbol{P}_1$ *when using the first transformation to add support for strong negation. To verify this, observe that* $\boldsymbol{P}_1^\dagger = \langle P^\dagger, U^\dagger\rangle$ *where*

$$P^\dagger: \quad p. \quad \neg p. \quad \sim p \leftarrow \neg p. \quad \sim\neg p \leftarrow p. \qquad U^\dagger: \quad \sim p \leftarrow \neg p. \quad \sim\neg p \leftarrow p.$$

Consider the interpretation $J_1 = \{p\}$*. It is not difficult to verify that*

$$\mathsf{rej}_{\geq}(\boldsymbol{P}_1^\dagger, J_1) = \{\neg p., \sim\neg p \leftarrow p.\} \ ,$$
$$\mathsf{def}(\boldsymbol{P}_1^\dagger, J_1) = \emptyset \ ,$$

so it follows that

$$\mathsf{least}\left(\left[\mathsf{all}(\boldsymbol{P}_1^\dagger) \setminus \mathsf{rej}_{\geq}(\boldsymbol{P}_1^\dagger, J_1)\right] \cup \mathsf{def}(\boldsymbol{P}_1^\dagger, J_1)\right) = \{p, \sim\neg p\} = J_1^*.$$

In other words, J_1 *belongs to* $[\![\boldsymbol{P}_1^\dagger]\!]_{\mathsf{RD}}$ *and in an analogous fashion it can be verified that* $J_2 = \{\neg p\}$ *also belongs there. A similar situation occurs with* $[\![\boldsymbol{P}_1^\dagger]\!]_{\mathsf{WS}}$ *since the rules that were added to the more recent program can be used to reject facts in the older one.*

Thus, the problem with the first transformation is that an update by an empty program, which does not express any change in the represented domain, may affect the original semantics. This behaviour goes against basic and intuitive principles underlying updates, grounded already in the classical belief update postulates (Keller and Winslett 1985; Katsuno and Mendelzon 1991) and satisfied by virtually all belief update operations (Herzig and Rifi 1999) as well as by the vast majority of existing rule update semantics, including the original RD- and WS-semantics.

This undesired behaviour can be corrected by using the second transformation instead. The more technical reason is that it does not add any rules to a program in the sequence unless that program already contains some original rules. However, its use leads to another problem: sometimes *no model* is assigned when in fact a model should exist.

Example 3.2 (Undesired side effects of the second transformation)**.** *Consider again Example 1.2, formalised as the DLP* $\boldsymbol{P}_2 = \langle P, V\rangle$ *where* $P = \{p., \neg p.\}$ *and* $V = \{\sim p.\}$*. It is reasonable to expect that since* V *resolves the conflict present in* P*, a stable model should be assigned to* $\boldsymbol{P}_2$*. However,* $[\![\boldsymbol{P}_2^\ddagger]\!]_{\mathsf{RD}} = [\![\boldsymbol{P}_2^\ddagger]\!]_{\mathsf{WS}} = \emptyset$*. To verify this, observe that* $\boldsymbol{P}_2^\ddagger = \langle P^\ddagger, V^\ddagger\rangle$ *where*

$$P^\ddagger: \quad p. \quad \neg p. \quad \sim p. \quad \sim\neg p. \qquad V^\ddagger: \quad \sim p.$$

Given an interpretation J *and level mapping* ℓ*, we conclude that* $\mathsf{rej}_\ell(\boldsymbol{P}_2^\ddagger, J) = \{p.\}$*, so the facts* $(\neg p.)$ *and* $(\sim\neg p.)$ *both belong to the program*

$$\mathsf{all}(\boldsymbol{P}_2^\ddagger) \setminus \mathsf{rej}_\ell(\boldsymbol{P}_2^\ddagger, J) \ .$$

Consequently, this program has no model and it follows that J *cannot belong to* $[\![\boldsymbol{P}_2^\ddagger]\!]_{\mathsf{WS}}$*. Similarly it can be shown that* $[\![\boldsymbol{P}_2^\ddagger]\!]_{\mathsf{RD}} = \emptyset$*.*

Based on this example, in the following we formulate a generic *early recovery principle* that formally identifies conditions under which *some* stable model should be assigned to a DLP. For the sake of simplicity, we concentrate on DLPs of length 2 which are composed of facts. We discuss a generalisation of the principle to DLPs of arbitrary length and containing other rules than just facts in Sect. 5. After introducing the principle, we define a semantics for rule updates which directly supports both strong and default negation and satisfies the principle.

We begin by defining, for every objective literal l, the sets of literals $\bar{l}$ and $\overline{\sim l}$ as follows:

$$\bar{l} = \{\sim l, \neg l\} \qquad \text{and} \qquad \overline{\sim l} = \{l\} \ .$$

Intuitively, for every literal L, $\overline{L}$ denotes the set of literals that are in conflict with L. Furthermore, given two sets of facts P and U, we say that U *solves all conflicts in* P if for each pair of rules $\pi, \sigma \in P$ such that $\mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi}$ there is a fact $\rho \in U$ such that either $\mathsf{H}_\rho \in \overline{\mathsf{H}_\pi}$ or $\mathsf{H}_\rho \in \overline{\mathsf{H}_\sigma}$.

Considering a rule update semantics S, the new principle simply requires that when U solves all conflicts in P, S will assign *some model* to $\langle P, U\rangle$. Formally:

Early recovery principle: If P is a set of facts and U is a consistent set of facts that solves all conflicts in P, then $[\![\langle P, U\rangle]\!]_{\mathsf{S}} \neq \emptyset$.

We conjecture that rule update semantics should generally satisfy the above principle. In contrast with the usual behaviour of belief update operators, the nature of existing rule update semantics ensures that recovery from conflict is always possible, and this principle simply formalises and sharpens the sufficient conditions for such recovery.

Our next goal is to define a semantics for rule updates that not only satisfies the outlined principle, but also enjoys other established properties of rule updates that have been identified over the years. Similarly as for the original semantics for rule updates, we provide two equivalent definitions, one based on a fixed point equation and the other one on level mappings.

To directly accommodate strong negation in the RD-semantics, we first need to look more closely at the set of rejected rules $\mathsf{rej}_{\geq}(\boldsymbol{P}, J)$, particularly at the fact that it allows conflicting rules within the same component of $\boldsymbol{P}$ to reject one another. This behaviour, along with the constrained set of defaults $\mathsf{def}(\boldsymbol{P}, J)$, is used to prevent tautological and other irrelevant cyclic updates from affecting the semantics. However, in the presence of strong negation, rejecting conflicting rules within the same program has undesired side effects. For example, the early recovery principle requires that some model be assigned to the DLP $\langle\{p., \neg p.\}, \{\sim p\}\rangle$ from Example 3.2, but if the rules in the initial program reject each other, then the only possible stable model to assign is $\emptyset$. However, such a stable model would violate the causal rejection principle since it does not satisfy the initial rule $(\neg p.)$ and there is no rule in the updating program that overrides it.

To overcome the limitations of this approach to the prevention of tautological updates, we disentangle rule rejection per se from ensuring that rejection is done without

cyclic justifications. We introduce the set of rejected rules $\mathsf{rej}^{\neg}_{>}(\boldsymbol{P}, S)$ which directly supports strong negation and does not allow for rejection within the same program. Prevention of cyclic rejections is done separately by using a customised immediate consequence operator $T_{\boldsymbol{P},J}$. Given a stable model candidate J, instead of verifying that J^* is the least fixed point of the usual consequence operator, as done in the RD-semantics using $\mathsf{least}(\cdot)$, we verify that J^* is the least fixed point of $T_{\boldsymbol{P},J}$.

Definition 3.3 (Extended RD-semantics)**.** *Let $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ be a DLP. Given an interpretation J and a set of literals S, the multiset of rejected rules $\mathsf{rej}^{\neg}_{>}(\boldsymbol{P}, S)$, the remainder $\mathsf{rem}(\boldsymbol{P}, S)$ and the consequence operator $T_{\boldsymbol{P},J}$ are defined as follows:*

$$\mathsf{rej}^{\neg}_{>}(\boldsymbol{P}, S) = \{\pi \in P_i | i < n \wedge \exists j > i\ \exists \sigma \in P_j : \mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi} \wedge \mathsf{B}_\sigma \subseteq S\},$$

$$\mathsf{rem}(\boldsymbol{P}, S) = \mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}^{\neg}_{>}(\boldsymbol{P}, S)\ ,$$

$$T_{\boldsymbol{P},J}(S) = \big\{\, \mathsf{H}_\pi \mid \pi \in (\mathsf{rem}(\boldsymbol{P}, J^*) \cup \mathsf{def}(J)) \wedge \mathsf{B}_\pi \subseteq S \wedge \neg \big(\exists \sigma \in \mathsf{rem}(\boldsymbol{P}, S) : \mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi} \wedge \mathsf{B}_\sigma \subseteq J^*\big) \,\big\}.$$

Furthermore, $T^0_{\boldsymbol{P},J}(S) = S$ and for every $k \geq 0$, $T^{k+1}_{\boldsymbol{P},J}(S) = T_{\boldsymbol{P},J}(T^k_{\boldsymbol{P},J}(S))$. The set $[\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{RD}}$ of extended RD-models of $\boldsymbol{P}$ *consists of all interpretations J such that*

$$J^* = \bigcup_{k \geq 0} T^k_{\boldsymbol{P},J}(\emptyset)\ .$$

Adding support for strong negation to the WS-semantics is done by modifying the set of rejected rules $\mathsf{rej}_\ell(\boldsymbol{P}, J)$ to account for the new type of conflict. Additionally, in order to ensure that rejection of a literal L cannot be based on the assumption that some conflicting literal $L' \in \overline{L}$ is true, a rejecting rule σ must satisfy the stronger condition $\ell^{\downarrow}(\overline{L}) > \ell^{\uparrow}(\mathsf{B}_\sigma)$. Finally, to prevent defeated rules from affecting the resulting models, we require that all supporting rules belong to $\mathsf{rem}(\boldsymbol{P}, J^*)$.

Definition 3.4 (Extended WS-semantics)**.** *Let $\boldsymbol{P} = \langle P_i \rangle_{i<n}$ be a DLP. Given an interpretation J and a level mapping ℓ, the multiset of rejected rules $\mathsf{rej}^{\neg}_{\ell}(\boldsymbol{P}, J)$ is defined by:*

$$\mathsf{rej}^{\neg}_{\ell}(\boldsymbol{P}, J) = \{\pi \in P_i | i < n \wedge \exists j > i\ \exists \sigma \in P_j : \mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi} \wedge J \models \mathsf{B}_\sigma \wedge \ell^{\downarrow}(\overline{\mathsf{H}_\pi}) > \ell^{\uparrow}(\mathsf{B}_\sigma)\}.$$

The set $[\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{WS}}$ of extended WS-models of $\boldsymbol{P}$ *consists of all interpretations J such that for some level mapping ℓ, the following conditions are satisfied:*

1. *J is a model of $\mathsf{all}(\boldsymbol{P}) \setminus \mathsf{rej}^{\neg}_{\ell}(\boldsymbol{P}, J)$;*
2. *For every $l \in J$ there exists some rule $\pi \in \mathsf{rem}(\boldsymbol{P}, J^*)$ such that*

$$\mathsf{H}_\pi = l \wedge J \models \mathsf{B}_\pi \wedge \ell(\mathsf{H}_\pi) > \ell^{\uparrow}(\mathsf{B}_\pi)\ .$$

The following theorem establishes that the two defined semantics are equivalent:

Theorem 3.5. *Let $\boldsymbol{P}$ be a DLP. Then, $[\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{WS}} = [\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{RD}}$.*

Also, on DLPs without strong negation they coincide with the original semantics.

Theorem 3.6. *Let $\boldsymbol{P}$ be a DLP without strong negation. Then, $[\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{WS}} = [\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{RD}} = [\![\boldsymbol{P}]\!]_{\mathsf{WS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}}$.*

Furthermore, unlike the transformational semantics for strong negation, the new semantics satisfy the early recovery principle.

Theorem 3.7. *The extended RD-semantics and extended WS-semantics satisfy the early recovery principle.*

4 Properties

In this section we take a closer look at the formal and computational properties of the proposed rule update semantics.

The various approaches to rule updates (Leite and Pereira 1998; Alferes et al. 2000; Eiter et al. 2002; Leite 2003; Sakama and Inoue 2003; Alferes et al. 2005; Banti et al. 2005; Zhang 2006; Šefránek 2006; Osorio and Cuevas 2007; Delgrande, Schaub, and Tompits 2007; Šefránek 2011; Krümpelmann 2012) share a number of basic characteristics. For example, all of them generalise stable models, i.e., the models they assign to a sequence $\langle P \rangle$ (of length 1) are exactly the stable models of P. Similarly, they adhere to the principle of primacy of new information (Dalal 1988), so models assigned to $\langle P_i \rangle_{i<n}$ satisfy the latest program P_{n-1}. However, they also differ significantly in their technical realisation and classes of supported inputs, and desirable properties such as immunity to tautologies are violated by many of them.

Table 1 lists many of the generic properties proposed for rule updates that have been identified and formalised throughout the years (Leite and Pereira 1998; Eiter et al. 2002; Leite 2003; Alferes et al. 2005). The rule update semantics we defined in the previous section enjoys all of them.

Theorem 4.1. *The extended RD-semantics and extended WS-semantics satisfy all properties listed in Table 1.*

Our semantics also retains the same computational complexity as the stable models.

Theorem 4.2. *Let $\boldsymbol{P}$ be a DLP. The problem of deciding whether some $J \in [\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{WS}}$ exists is NP-complete. Given a literal L, the problem of deciding whether for all $J \in [\![\boldsymbol{P}]\!]^{\neg}_{\mathsf{WS}}$ it holds that $J \models L$ is coNP-complete.*

5 Concluding Remarks

In this paper we have identified shortcomings in the existing semantics for rule updates that fully support both strong and default negation, and proposed a generic *early recovery principle* that captures them formally. Subsequently, we provided two equivalent definitions of a new semantics for rule updates.

We have shown that the newly introduced rule update semantics constitutes a strict improvement upon the state of the art in rule updates as it enjoys the following combination of characteristics, unmatched by any previously existing semantics:

- It allows for both strong and default negation in heads of rules, making it possible to move between any pair of epistemic states by means of updates;

Table 1: Desirable properties of rule update semantics

Generalisation of stable models	$[\![\langle P\rangle]\!]_{\mathsf{s}} = [\![P]\!]_{\mathsf{SM}}$.
Primacy of new information	If $J \in [\![\langle P_i\rangle_{i<n}]\!]_{\mathsf{s}}$, then $J \models P_{n-1}$.
Fact update	A sequence of consistent sets of facts $\langle P_i\rangle_{i<n}$ has the single model $\{\, l \in \mathfrak{L} \mid \exists i < n : (l.) \in P_i \wedge (\forall j > i : \{\, \neg l., \sim l.\,\} \cap P_j = \emptyset)\,\}$.
Support	If $J \in [\![\boldsymbol{P}]\!]_{\mathsf{s}}$ and $l \in J$, then there is some rule $\pi \in \mathsf{all}(\boldsymbol{P})$ such that $\mathsf{H}_\pi = l$ and $J \models \mathsf{B}_\pi$.
Idempotence	$[\![\langle P, P\rangle]\!]_{\mathsf{s}} = [\![\langle P\rangle]\!]_{\mathsf{s}}$.
Absorption	$[\![\langle P, U, U\rangle]\!]_{\mathsf{s}} = [\![\langle P, U\rangle]\!]_{\mathsf{s}}$.
Augmentation	If $U \subseteq V$, then $[\![\langle P, U, V\rangle]\!]_{\mathsf{s}} = [\![\langle P, V\rangle]\!]_{\mathsf{s}}$.
Non-interference	If U and V are over disjoint alphabets, then $[\![\langle P, U, V\rangle]\!]_{\mathsf{s}} = [\![\langle P, V, U\rangle]\!]_{\mathsf{s}}$.
Immunity to empty updates	If $P_j = \emptyset$, then $[\![\langle P_i\rangle_{i<n}]\!]_{\mathsf{s}} = \left[\!\!\left[\langle P_i\rangle_{i<n \wedge i\neq j}\right]\!\!\right]_{\mathsf{s}}$.
Immunity to tautologies	If $\langle Q_i\rangle_{i<n}$ is a sequence of sets of tautologies, then $[\![\langle P_i \cup Q_i\rangle_{i<n}]\!]_{\mathsf{s}} = [\![\langle P_i\rangle_{i<n}]\!]_{\mathsf{s}}$.
Causal rejection principle	For every $i < n$, $\pi \in P_i$ and $J \in [\![\langle P_i\rangle_{i<n}]\!]_{\mathsf{s}}$, if $J \not\models \pi$, then there exists some $\sigma \in P_j$ with $j > i$ such that $\mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi}$ and $J \models \mathsf{B}_\sigma$.

- It satisfies the *early recovery principle* which guarantees the existence of a model whenever all conflicts in the original program are satisfied;
- It enjoys all rule update principles and desirable properties reported in Table 1;
- It does not increase the computational complexity of the stable model semantics upon which it is based.

However, the early recovery principle, as it is formulated in Sect. 3, only covers a single update of a set of facts by another set of facts. Can it be generalised further without rendering it too strong? Certain caution is appropriate here, since in general the absence of a stable model can be caused by odd cycles or simply by the fundamental differences between different approaches to rule update, and the purpose of this principle is not to choose which approach to take.

Nevertheless, one generalisation that should cause no harm is the generalisation to iterated updates, i.e. to sequences of sets of facts. Another generalisation that appears very reasonable is the generalisation to *acyclic DLPs*, i.e. DLPs such that $\mathsf{all}(\boldsymbol{P})$ is an acyclic program. An acyclic program has at most one stable model, and if we guarantee that all potential conflicts within it certainly get resolved, we can safely conclude that the rule update semantics should assign some model to it. We formalise these ideas in what follows.

We say that a program P is *acyclic* (Apt and Bezem 1991) if for some level mapping ℓ, such that for every $l \in \mathfrak{L}$, $\ell(l) = \ell(\neg l)$, and every rule $\pi \in P$ it holds that $\ell(\mathsf{H}_\pi) > \ell^{\uparrow}(\mathsf{B}_\pi)$. Given a DLP $\boldsymbol{P} = \langle P_i\rangle_{i<n}$, we say that *all conflicts in $\boldsymbol{P}$ are solved* if for every $i < n$ and each pair of rules $\pi, \sigma \in P_i$ such that $\mathsf{H}_\sigma \in \overline{\mathsf{H}_\pi}$ there is some $j > i$ and a fact $\rho \in P_j$ such that either $\mathsf{H}_\rho \in \overline{\mathsf{H}_\pi}$ or $\mathsf{H}_\rho \in \overline{\mathsf{H}_\sigma}$.

Generalised early recovery principle: If $\mathsf{all}(\boldsymbol{P})$ is acyclic and all conflicts in $\boldsymbol{P}$ are solved, then $[\![\boldsymbol{P}]\!]_{\mathsf{s}} \neq \emptyset$.

Note that this generalisation of the early recovery principle applies to a much broader class of DLPs than the original one. We illustrate this in the following example:

Example 5.1 (Recovery in a stratified program)**.** *Consider the following programs programs P, U and V:*

$$
\begin{array}{lllll}
P: & p \leftarrow q, \sim r. & \sim p \leftarrow s. & q. & s \leftarrow q. \\
U: & \neg p. & & r \leftarrow q. & \neg r \leftarrow q, s. \\
V: & & & \sim r. &
\end{array}
$$

Looking more closely at program P, we see that atoms q and s are derived by the latter two rules inside it while atom r is false by default since there is no rule that could be used to derive its truth. Consequently, the bodies of the first two rules are both satisfied and as their heads are conflicting, P has no stable model. The single conflict in P is solved after it is updated by U, but then another conflict is introduced due to the latter two rules in the updating program. This second conflict can be solved after another update by V. Consequently, we expect that some stable model be assigned to the DLP $\langle P, U, V\rangle$.

The original early recovery principle does not impose this because the DLP in question has more than two components and the rules within it are not only facts. However, the DLP is acyclic, as shown by any level mapping ℓ with $\ell(p) = 3$, $\ell(q) = 0$, $\ell(r) = 2$ and $\ell(s) = 1$, so the generalised early recovery principle does apply. Furthermore, we also find the single extended RD-model of $\langle P, U, V\rangle$ is $\{\, \neg p, q, \neg r, s\,\}$, i.e. the semantics respects the stronger principle in this case.

Moreover, as established in the following theorem, it is no coincidence that the extended RD-semantics respects the stronger principle in the above example – the principle is generally satisfied by the semantics introduced in this paper.

Theorem 5.2. *The extended RD-semantics and extended WS-semantics satisfy the generalised early recovery principle.*

Both the original and the generalised early recovery principle can guide the future addition of full support for both kinds of negations in other approaches to rule updates, such as those proposed in (Sakama and Inoue 2003; Zhang 2006; Delgrande, Schaub, and Tompits 2007; Krümpelmann 2012), making it possible to reach any belief state by updating the current program. Furthermore, adding support for strong negation is also interesting in the context of recent results on program revision and updates that are performed on the *semantic level*, ensuring syntax-independence of the respective methods (Delgrande et al. 2013; Slota and Leite 2014; 2012a; 2010), in the context of finding suitable condensing operators (Slota and Leite 2013), and unifying with updates in classical logic (Slota and Leite 2012b).

Acknowledgments

João Leite was partially supported by Fundação para a Ciência e a Tecnologia under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010). Martin Slota was partially supported by Fundação para a Ciência e a Tecnologia under project "ASPEN – Answer Set Programming with BoolEaN Satisfiability" (PTDC/EIA-CCO/110921/2009). The collaboration between the co-authors resulted from the Slovak–Portuguese bilateral project "ReDIK – Reasoning with Dynamic Inconsistent Knowledge", supported by APVV agency under SK-PT-0028-10 and by Fundação para a Ciência e a Tecnologia (FCT/2487/3/6/2011/S).

References

Alferes, J. J., and Pereira, L. M. 1996. Update-programs can update programs. In Dix, J.; Pereira, L. M.; and Przymusinski, T. C., eds., *Non-Monotonic Extensions of Logic Programming (NMELP '96), Selected Papers*, volume 1216 of *Lecture Notes in Computer Science*, 110–131. Bad Honnef, Germany: Springer.

Alferes, J. J.; Leite, J. A.; Pereira, L. M.; Przymusinska, H.; and Przymusinski, T. C. 1998. Dynamic logic programming. In Cohn, A. G.; Schubert, L. K.; and Shapiro, S. C., eds., *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*, 98–111. Morgan Kaufmann.

Alferes, J. J.; Leite, J. A.; Pereira, L. M.; Przymusinska, H.; and Przymusinski, T. C. 2000. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming* 45(1-3):43–70.

Alferes, J. J.; Banti, F.; Brogi, A.; and Leite, J. A. 2005. The refined extension principle for semantics of dynamic logic programming. *Studia Logica* 79(1):7–32.

Apt, K. R., and Bezem, M. 1991. Acyclic programs. *New Generation Computing* 9(3/4):335–364.

Banti, F.; Alferes, J. J.; Brogi, A.; and Hitzler, P. 2005. The well supported semantics for multidimensional dynamic logic programs. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, volume 3662 of *Lecture Notes in Computer Science*, 356–368. Diamante, Italy: Springer.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The semantic web. *Scientific American* 284(5):28–37.

Dalal, M. 1988. Investigations into a theory of knowledge base revision. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988)*, 475–479. St. Paul, MN, USA: AAAI Press / The MIT Press.

Delgrande, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2013. A model-theoretic approach to belief change in answer set programming. *ACM Transactions on Computational Logic (TOCL)* 14(2):14:1–14:46.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2007. A preference-based framework for updating logic programs. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, volume 4483 of *Lecture Notes in Computer Science*, 71–83. Tempe, AZ, USA: Springer.

Eiter, T.; Fink, M.; Sabbatini, G.; and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming (TPLP)* 2(6):721–777.

Fages, F. 1991. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. *New Generation Computing* 9(3/4):425–444.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, 1070–1080. Seattle, Washington: MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3-4):365–385.

Herzig, A., and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence* 115(1):107–138.

Inoue, K., and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming* 35(1):39–78.

Katsuno, H., and Mendelzon, A. O. 1991. On the difference between updating a knowledge base and revising it. In Allen, J. F.; Fikes, R.; and Sandewall, E., eds., *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 387–394. Cambridge, MA, USA: Morgan Kaufmann Publishers.

Keller, A. M., and Winslett, M. 1985. On the use of an extended relational model to handle changing incomplete information. *IEEE Transactions on Software Engineering* 11(7):620–633.

Krümpelmann, P. 2012. Dependency semantics for sequences of extended logic programs. *Logic Journal of the IGPL* 20(5):943–966.

Leite, J. A., and Pereira, L. M. 1998. Generalizing updates: From models to programs. In Dix, J.; Pereira, L. M.;

and Przymusinski, T. C., eds., *Proceedings of the 3rd International Workshop on Logic Programming and Knowledge Representation (LPKR '97), October 17, 1997, Port Jefferson, New York, USA*, volume 1471 of *Lecture Notes in Computer Science*, 224–246. Springer.

Leite, J. A. 2003. *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover*. IOS Press.

Osorio, M., and Cuevas, V. 2007. Updates in answer set programming: An approach based on basic structural properties. *Theory and Practice of Logic Programming* 7(4):451–479.

Sakama, C., and Inoue, K. 2003. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming (TPLP)* 3(6):671–713.

Šefránek, J. 2006. Irrelevant updates and nonmonotonic assumptions. In Fisher, M.; van der Hoek, W.; Konev, B.; and Lisitsa, A., eds., *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, volume 4160 of *Lecture Notes in Computer Science*, 426–438. Liverpool, UK: Springer.

Šefránek, J. 2011. Static and dynamic semantics: Preliminary report. *Mexican International Conference on Artificial Intelligence* 36–42.

Slota, M., and Leite, J. 2010. On semantic update operators for answer-set programs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 957–962. IOS Press.

Slota, M., and Leite, J. 2012a. Robust equivalence models for semantic updates of answer-set programs. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 158–168. Rome, Italy: AAAI Press.

Slota, M., and Leite, J. 2012b. A unifying perspective on knowledge updates. In del Cerro, L. F.; Herzig, A.; and Mengin, J., eds., *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings*, volume 7519 of *Lecture Notes in Computer Science*, 372–384. Springer.

Slota, M., and Leite, J. 2013. On condensing a sequence of updates in answer-set programming. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI.

Slota, M., and Leite, J. 2014. The rise and fall of semantic rule updates based on se-models. *Theory and Practice of Logic Programming* FirstView:1–39.

Slota, M.; Baláž, M.; and Leite, J. 2014. On strong and default negation in logic program updates (extended version). *CoRR* abs/1404.6784.

Zhang, Y. 2006. Logic program-based updates. *ACM Transactions on Computational Logic* 7(3):421–472.

Inference in the FO(C) Modelling Language

Bart Bogaerts and **Joost Vennekens** and **Marc Denecker**
Department of Computer Science, KU Leuven
{bart.bogaerts, joost.vennekens, marc.denecker}@cs.kuleuven.be

Jan Van den Bussche
Hasselt University & transnational University of Limburg
jan.vandenbussche@uhasselt.be

Abstract

Recently, FO(C), the integration of C-LOG with classical logic, was introduced as a knowledge representation language. Up to this point, no systems exist that perform inference on FO(C), and very little is known about properties of inference in FO(C). In this paper, we study both of the above problems. We define normal forms for FO(C), one of which corresponds to FO(ID). We define transformations between these normal forms, and show that, using these transformations, several inference tasks for FO(C) can be reduced to inference tasks for FO(ID), for which solvers exist. We implemented this transformation and hence, created the first system that performs inference in FO(C). We also provide results about the complexity of reasoning in FO(C).

1 Introduction

Knowledge Representation and Reasoning is a subfield of Artificial Intelligence concerned with two tasks: defining modelling languages that allow intuitive, clear, representation of knowledge and developing inference tools to reason with this knowledge. Recently, C-LOG was introduced with a strong focus on the first of these two goals (Bogaerts et al. in press 2014). C-LOG has an expressive recursive syntax suitable for expressing various forms of non-monotonic reasoning: disjunctive information in the context of closed world assumptions, non-deterministic inductive constructions, causal processes, and ramifications. C-LOG allows for example nested occurrences of causal rules.

It is straightforward to integrate first-order logic (FO) with C-LOG, offering an expressive modelling language in which causal processes as well as assertional knowledge in the form of axioms and constraints can be naturally expressed. We call this integration FO(C).[1] FO(C) fits in the FO($\cdot$) research project (Denecker 2012), which aims at integrating expressive language constructs with a Tarskian model semantics in a unified language.

An example of a C-LOG expression is the following

$$\left\{ \begin{array}{l} \textbf{All}\, p[\mathrm{Apply}(p) \wedge \mathrm{PassedTest}(p)] : \mathrm{PermRes}(p). \\ (\textbf{Select}\, p[\mathrm{Participate}(p)] : \mathrm{PermRes}(p)) \leftarrow \mathrm{Lott}. \end{array} \right\}$$

This describes that all persons who pass a naturalisation test obtain permanent residence in the U.S., and that one person who participates in the green card lottery also obtains residence. The person that is selected for the lottery can either be one of the persons that also passed the naturalisation test, or someone else. There are local closed world assumptions: in the example, the endogenous predicate $\mathrm{PermRes}$ only holds for the people passing the test and at most one extra person. We could add an FO constraint to this theory, for example $\forall p : \mathrm{Participate}(p) \Rightarrow \mathrm{Apply}(p)$. This results in a FO(C) theory; a structure is a model of this theory if it is a model of the C-LOG expression and no-one participates in the lottery without applying the normal way.

So far, very little is known about inference in FO(C). No systems exist to reason with FO(C), and complexity of inference in FO(C) has not been studied. This paper studies both of the above problems.

The rest of this paper is structured as follows: in Section 2, we repeat some preliminaries, including a very brief overview of the semantics of FO(C). In Section 3 we define normal forms on FO(C) and transformations between these normal forms. We also argue that one of these normal forms corresponds to FO(ID) (Denecker and Ternovska 2008) and hence, that IDP (De Cat et al. 2014) can be seen as the first FO(C)-solver. In Section 4 we give an example that illustrates both the semantics of FO(C) and the transformations. Afterwards, in Section 5, we define inference tasks for FO(C) and study their complexity. We conclude in Section 6.

2 Preliminaries

We assume familiarity with basic concepts of FO. Vocabularies, formulas, and terms are defined as usual. A Σ-structure I interprets all symbols (including variable symbols) in Σ; D^I denotes the domain of I and σ^I, with σ a symbol in Σ, the interpretation of σ in I. We use $I[\sigma : v]$ for the structure J that equals I, except on σ: $\sigma^J = v$. *Domain atoms* are atoms of the form $P(\bar{d})$ where the d_i are domain elements. We use restricted quantifications, see e.g. (Preyer and Peter 2002). In FO, these are formulas of the form $\forall x[\psi] : \varphi$ or $\exists x[\psi] : \varphi$, meaning that φ holds for all (resp. for some) x such that ψ holds. The above expressions are syntactic sugar for $\forall x : \psi \Rightarrow \varphi$ and $\exists x : \psi \wedge \varphi$, but such a reduction is not possible for other restricted quantifiers in C-LOG. We call ψ the *qualification* and φ the *assertion* of the restricted quantifications. From now on, let Σ be a relational vocabulary, i.e., Σ consists only of predicate, constant

[1] Previously, this language was called FO(C-LOG)

and variable symbols.

Our logic has a standard, two-valued Tarskian semantics, which means that models represent possible states of affairs. Three-valued logic with partial domains is used as a technical device to express intermediate stages of causal processes. A truth-value is one of the following: $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, where $\mathbf{f}^{-1} = \mathbf{t}, \mathbf{t}^{-1} = \mathbf{f}$ and $\mathbf{u}^{-1} = \mathbf{u}$. Two partial orders are defined on truth values: the precision order $\leq_p$, given by $\mathbf{u} \leq_p \mathbf{t}$ and $\mathbf{u} \leq_p \mathbf{f}$ and the truth order $\mathbf{f} \leq \mathbf{u} \leq \mathbf{t}$. Let D be a set, a *partial set* $\mathcal{S}$ in D is a function from D to truth values. We identify a partial set with a tuple $(\mathcal{S}_{ct}, \mathcal{S}_{pt})$ of two sets, where the *certainly true set* $\mathcal{S}_{ct}$ is $\{x \mid \mathcal{S}(x) = \mathbf{t}\}$ and the *possibly true set* $\mathcal{S}_{pt}$ is $\{x \mid \mathcal{S}(x) \neq \mathbf{f}\}$. The union, intersection, and subset-relation of partial sets are defined pointwise. For a truth value v, we define the restriction of a partial set $\mathcal{S}$ to this truth-value, denoted $r(\mathcal{S}, v)$, as the partial set mapping every $x \in D$ to $\min_{\leq}(\mathcal{S}(x), v)$. Every set S is also a partial set, namely the tuple (S, S).

A *partial Σ-structure* I consists of 1) a *domain* D^I: a partial set of elements, and 2) a mapping associating a value to each symbol in Σ; for constants and variables, this value is in D^I_{ct}, for predicate symbols of arity n, this is a partial set P^I in $(D^I_{\text{pt}})^n$. We often abuse notation and use the domain D as if it were a predicate. A partial structure I is *two-valued* if for all predicates P (including D), $P^I_{\text{ct}} = P^I_{\text{pt}}$. There is a one-to-one correspondence between two-valued partial structures and structures. If I and J are two partial structures with the same interpretation for constants, we call I more precise than J ($I \geq_p J$) if for all its predicates P (including D), $P^I_{\text{ct}} \supseteq P^J_{\text{ct}}$ and $P^I_{\text{pt}} \subseteq P^J_{\text{pt}}$.

Definition 2.1. *We define the value of an FO formula φ in a partial structure I inductively based on the Kleene truth tables (Kleene 1938).*

- $P(\overline{t})^I = P^I(\overline{t}^I)$,
- $(\neg\varphi)^I = ((\varphi)^I)^{-1}$
- $(\varphi \wedge \psi)^I = \min_{\leq}\left(\varphi^I, \psi^I\right)$
- $(\varphi \vee \psi)^I = \max_{\leq}\left(\varphi^I, \psi^I\right)$
- $(\forall x : \varphi)^I = \min_{\leq}\left\{\max(D^I(d)^{-1}, \varphi^{I[x:d]}) \mid d \in D^I_{\text{pt}}\right\}$
- $(\exists x : \varphi)^I = \max_{\leq}\left\{\min(D^I(d), \varphi^{I[x:d]}) \mid d \in D^I_{\text{pt}}\right\}$

In what follows we briefly repeat the syntax and formal semantics of C-LOG. For more details, an extensive overview of the informal semantics of CEEs, and examples of CEEs, we refer to (Bogaerts et al. in press 2014).

2.1 Syntax of C-LOG

Definition 2.2. Causal effect expressions *(CEE) are defined inductively as follows:*

- *if $P(\overline{t})$ is an atom, then $P(\overline{t})$ is a CEE,*
- *if φ is an FO formula and C' is a CEE, then $C' \leftarrow \varphi$ is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 $\mathbf{And}$ C_2 is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 $\mathbf{Or}$ C_2 is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then $\mathbf{All}\, x[\varphi] : C'$ is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then $\mathbf{Select}\, x[\varphi] : C'$ is a CEE,*
- *if x is a variable and C' is a CEE, then $\mathbf{New}\, x : C'$ is a CEE.*

We call a CEE an *atom-* (respectively *rule-*, **And**-, **Or**-, **All**-, **Select**- or **New**-*expression*) if it is of the corresponding form. We call a predicate symbol P *endogenous* in C if P occurs as the symbol of a (possibly nested) atom-expression in C. All other symbols are called *exogenous* in C. An occurrence of a variable x is *bound* in a CEE if it occurs in the scope of a quantification over that variable ($\forall x$, $\exists x$, $\mathbf{All}\, x$, $\mathbf{Select}\, x$, or $\mathbf{New}\, x$) and *free* otherwise. A variable is *free* in a CEE if it has free occurrences. A *causal theory*, or C-LOG *theory* is a CEE without free variables. By abuse of notation, we often represent a causal theory as a finite set of CEEs; the intended causal theory is the **And**-conjunction of these CEEs. We often use Δ for a causal theory and C, C', C_1 and C_2 for its subexpressions. We stress that the connectives in CEEs differ from their FO counterparts. E.g., in the example in the introduction, the CEE expresses that there is a cause for several persons to become American (those who pass the test and maybe one extra lucky person). This implicitly also says that every person without cause for becoming American is not American. As such C-LOG-expressions are highly non-monotonic.

2.2 Semantics of C-LOG

Definition 2.3. *Let Δ be a causal theory; we associate a parse-tree with Δ. An* occurrence *of a CEE C in Δ is a node in the parse tree of Δ labelled with C. The* variable context *of an occurrence of a CEE C in Δ is the sequence of quantified variables as they occur on the path from Δ to C in the parse-tree of Δ. If $\overline{x}$ is the variable context of C in Δ, we denote C as $C\langle\overline{x}\rangle$ and the length of $\overline{x}$ as n_C.*

For example, the variable context of $P(x)$ in $\mathbf{Select}\, y[Q(y)] : \mathbf{All}\, x[Q(x)] : P(x)$ is $[y, x]$. *Instances* of an occurrence $C\langle\overline{x}\rangle$ correspond to assignments $\overline{d}$ of domain elements to $\overline{x}$.

Definition 2.4. *Let Δ be a causal theory and D a set. A Δ-*selection *ζ in D consists of*

- *for every occurrence C of a* **Select***-expression in Δ, a total function $\zeta^{sel}_C : D^{n_C} \to D$,*
- *for every occurrence C of a* **Or***-expression in Δ, a total function $\zeta^{or}_C : D^{n_C} \to \{1, 2\}$,*
- *for every occurrence C of a* **New***-expression in Δ, an injective partial function $\zeta^{new}_C : D^{n_C} \to D$.*

such that furthermore the images of all functions ζ^{new}_C are disjoint (i.e., such that every domain element can be created only once).

The initial elements *of ζ are those that do not occur as image of one of the ζ^{new}_C-functions: $\zeta^{in} = D \setminus \cup_C image(\zeta^{new}_C)$, where the union ranges over all occurrences of* **New***-expressions.*

The effect set of a CEE in a partial structure is a partial set: it contains information on everything that is caused and everything that might be caused. For defining the semantics a new, unary predicate $\mathcal{U}$ is used.

Definition 2.5. *Let Δ be a CEE and J a partial structure. Suppose ζ is a Δ-selection in a set $D \supseteq D^J_{\text{pt}}$. Let C be an occurrence of a CEE in Δ. The* effect set of *C with respect to J and ζ is a partial set of domain atoms, defined recursively:*

- *If C is $P(\overline{t})$, then* $\text{eff}_{J,\zeta}(C) = \{P(\overline{t}^J)\}$,
- *if C is C_1* **And** *C_2, then* $\text{eff}_{J,\zeta}(C) = \text{eff}_{J,\zeta}(C_1) \cup \text{eff}_{J,\zeta}(C_2)$,
- *if C is $C' \leftarrow \varphi$, then* $\text{eff}_{J,\zeta}(C) = r(\text{eff}_{J,\zeta}(C'), \varphi^J)$,
- *if C is* **All** *$x[\varphi] : C'$, then*

$$\text{eff}_{J,\zeta}(C) = \bigcup \Big\{ r\big(\text{eff}_{J',\zeta}(C'), \min_{\leq}(D^J(d), \varphi^{J'})\big) \mid d \in D^J_{\text{pt}} \text{ and } J' = J[x:d] \Big\}$$

- *if $C\langle\overline{y}\rangle$ is C_1* **Or** *C_2, then*
  - $\text{eff}_{J,\zeta}(C) = \text{eff}_{J,\zeta}(C_1)$ *if* $\zeta^{or}_C(\overline{y}^J) = 1$,
  - *and* $\text{eff}_{J,\zeta}(C) = \text{eff}_{J,\zeta}(C_2)$ *otherwise*
- *if $C\langle\overline{y}\rangle$ is* **Select** *$x[\varphi] : C'$, let $e = \zeta^{sel}_C(\overline{y}^J)$, $J' = J[x:e]$ and $v = \min_{\leq}(D^J(e), \varphi^{J'})$. Then* $\text{eff}_{J,\zeta}(C) = r(\text{eff}_{J,\zeta}(C'), v)$,
- *if $C\langle\overline{y}\rangle$ is* **New** *$x : C'$, then*
  - $\text{eff}_{J,\zeta}(C) = \emptyset$ *if $\zeta^{new}_C(\overline{y}^J)$ does not denote,*
  - *and* $\text{eff}_{J,\zeta}(C) = \{\mathcal{U}(\zeta^{new}_C(\overline{y}^J))\} \cup \text{eff}_{J',\zeta}(C')$, *where $J' = J[x : \zeta^{new}_C(\overline{y}^J)]$ otherwise,*

An instance of an occurrence of a CEE in Δ is relevant *if it is encountered in the evaluation of* $\text{eff}_{I,\zeta}(\Delta)$. *We say that C* succeeds[2] *with ζ in J if for all relevant occurrences $C\langle\overline{y}\rangle$ of* **Select***-expressions, $\zeta^{sel}_C(\overline{y}^J)$ satisfies the qualification of C and for all relevant instances $C\langle\overline{y}\rangle$ of* **New***-expressions, $\zeta^{new}_C(\overline{y}^J)$ denotes.*

Given a structure I (and a Δ-selection ζ), two lattices are defined: $L^{\Sigma}_{I,\zeta}$ denotes the set of all Σ-structures J with $\zeta^{in} \subseteq D^J \subseteq D^I$ such that for all exogenous symbols σ of arity n: $\sigma^J = \sigma^I \cap (D^J)^n$. This set is equipped with the truth order. And L^{Σ}_I denotes the sublattice of $L^{\Sigma}_{I,\zeta}$ consisting of all structures in $L^{\Sigma}_{I,\zeta}$ with domain equal to D^I.

A partial structure corresponds to an element of the bilattice $(L^{\Sigma}_{I,\zeta})^2$; the bilattice is equipped with the precision order.

Definition 2.6. *Let I be a structure and ζ a Δ-selection in D^I. The* partial immediate causality operator *A_ζ is the operator on $(L^{\Sigma}_{I,\zeta})^2$ that sends partial structure J to a partial structure J' such that*

- $D^{J'}(d) = \mathbf{t}$ *if $d \in \zeta^{in}$ and* $D^{J'}(d) = \text{eff}_{J,\zeta}(\Delta)(\mathcal{U}(d))$ *otherwise*
- *for endogenous symbols P,* $P(\overline{d})^{J'} = \text{eff}_{J,\zeta}(\Delta)(P(\overline{d}))$.

Such operators have been studied intensively in the field of Approximation Fixpoint Theory (Denecker, Bruynooghe, and Vennekens 2012); and for such operators, the well-founded fixpoint has been defined in (Denecker, Bruynooghe, and Vennekens 2012). The semantics of C-LOG is defined in terms of this well-founded fixpoint in (Bogaerts et al. in press 2014):

Definition 2.7. *Let Δ be a causal theory. We say that structure I is a* model *of Δ (notation $I \models \Delta$) if there exists a Δ-selection ζ such that (I,I) is the well-founded fixpoint of A_ζ, and Δ succeeds with ζ in I.*

FO(C) is the integration of FO and C-LOG. An FO(C) theory consists of a set of causal theories and FO sentences. A structure I is a model of an FO(C) theory if it is a model of all its causal theories and FO sentences. In this paper, we assume, without loss of generality, that an FO(C) theory $\mathcal{T}$ has exactly one causal theory.

3 A Transformation to DefF

In this section we present normal forms for FO(C) and transformations between these normal forms. The transformations we propose preserve equivalence modulo newly introduced predicates:

Definition 3.1. *Suppose $\Sigma \subseteq \Sigma'$ are vocabularies, $\mathcal{T}$ is an* FO(C) *theory over Σ and $\mathcal{T}'$ is an* FO(C) *theory over Σ'. We call $\mathcal{T}$ and $\mathcal{T}'$* Σ-equivalent *if each model of $\mathcal{T}$, can be extended to a model of $\mathcal{T}'$ and the restriction of each model of $\mathcal{T}'$ to Σ is a model of $\mathcal{T}$.*

From now on, we use **All** $\overline{x}[\varphi] : C'$, where $\overline{x}$ is a tuple of variables as syntactic sugar for **All** $x_1[\mathbf{t}] :$ **All** $x_2[\mathbf{t}] : \ldots$ **All** $x_n[\varphi] : C'$, and similar for **Select**-expressions. If $\overline{x}$ is a tuple of length 0, **All** $\overline{x}[\varphi] : C'$ is an abbreviation for $C' \leftarrow \varphi$. It follows directly from the definitions that **And** and **Or** are associative, hence we use C_1 **And** C_2 **And** C_3 as an abbreviation for $(C_1$ **And** $C_2)$ **And** C_3 and for C_1 **And** $(C_2$ **And** $C_3)$, and similar for **Or**-expressions.

3.1 Normal Forms

Definition 3.2. *Let C be an occurrence of a CEE in C'. The* nesting depth *of C in C' is the depth of C in the parse-tree of C'. In particular, the nesting depth of C' in C' is always 0. The* height *of C' is the maximal nesting depth of occurrences of CEEs in C'. In particular, the height of atom-expressions is always 0.*

Example 3.3. Let Δ be A **And** ((**All** $x[P(x)] : Q(x)$) **Or** B). The nesting depth of B in Δ is 2 and the height of Δ is 3.

Definition 3.4. *A* C-LOG *theory is* creation-free *if it does not contain any* **New***-expressions, it is* deterministic *if it is creation-free and it does not contain any* **Select** *or* **Or***-expressions. An* FO(C) *is* creation-free *(resp.* deterministic*) if its (unique)* C-LOG *theory is.*

Definition 3.5. *A* C-LOG *theory is in* Nesting Normal Form (NestNF) *if it is of the form C_1* **And** *C_2* **And** *C_3* **And** *$\ldots$ where each of the C_i is of the form* **All** *$\overline{x}[\varphi_i] : C'_i$ and each of the C'_i has height at most one. A* C-LOG *theory Δ is in* Definition Form (DefF) *if it is in NestNF and each of the C'_i have height zero, i.e., they are atom-expressions. An* FO(C) *theory is NestNF (respectively DefF) if its corresponding* C-LOG *theory is.*

[2] Previously, we did not say that C "succeeds", but that the effect set "is a possible effect set". We believe this new terminology is more clear.

Theorem 3.6. *Every* FO(C) *theory over* Σ *is* Σ*-equivalent with an* FO(C) *theory in DefF.*

We will prove this result in 3 parts: in Section 3.4, we show that every FO(C) theory can be transformed to NestNF, in Section 3.3, we show that every theory in NestNF can be transformed into a deterministic theory and in Section 3.2, we show that every deterministic theory can be transformed to DefF. The FO sentences in an FO(C) theory do not matter for the normal forms, hence most results focus on the C-LOG part of FO(C) theories.

3.2 From Deterministic FO(C) to DefF

Lemma 3.7. *Let* Δ *be a* C-LOG *theory. Suppose* C *is an occurrence of an expression* $\mathbf{All}\,\overline{x}[\varphi] : C_1\ \mathbf{And}\ C_2$*. Let* Δ' *be the causal theory obtained from* Δ *by replacing* C *with* $(\mathbf{All}\,\overline{x}[\varphi] : C_1)\ \mathbf{And}\ (\mathbf{All}\,\overline{x}[\varphi] : C_2)$*. Then* Δ *and* Δ' *are equivalent.*

Proof. It is clear that Δ and Δ' have the same selection functions. Furthermore, it follows directly from the definitions that given such a selection, the defined operators are equal. □

Repeated applications of the above lemma yield:

Lemma 3.8. *Every deterministic* FO(C) *theory is equivalent with an* FO(C) *theory in DefF.*

3.3 From NestNF to Deterministic FO(C)

Lemma 3.9. *If* $\mathcal{T}$ *is an* FO(C) *theory in NestNF over* Σ*, then* $\mathcal{T}$ *is* Σ*-equivalent with a deterministic* FO(C) *theory.*

We will prove Lemma 3.9 using a strategy that replaces a Δ-selection by an interpretation of new predicates (one per occurrence of a non-deterministic CEE). The most important obstacle for this transformation are **New**-expressions. In deterministic C-LOG, no constructs influence the domain. This has as a consequence that the immediate causality operator for a deterministic C-LOG theory is defined in a lattice of structures with fixed domain, while in general, the operator is defined in a lattice with variable domains. In order to bridge this gap, we use two predicates to describe the domain, $\mathcal{S}$ are the initial elements and $\mathcal{U}$ are the created, the union of the two is the domain. Suppose a C-LOG theory Δ over vocabulary Σ is given.

Definition 3.10. *We define the* Δ-selection vocabulary Σ^s_Δ *as the vocabulary consisting of:*

- *a unary predicate* $\mathcal{S}$*,*
- *for every occurrence* C *of a* **Or***-expression in* Δ*, a new* n_C*-ary predicate* $\mathrm{Choose1}_C$*,*
- *for every occurrence* C *of a* **Select***-expression in* Δ*, a new* (n_C+1)*-ary predicate* Sel_C*,*
- *for every occurrence* C *of a* **New***-expression in* Δ*, a new* (n_C+1)*-ary predicate* Create_C*,*

Intuitively, a Σ^s_Δ-structure corresponds to a Δ-selection: $\mathcal{S}$ correspond to ζ^{in}, $\mathrm{Choose1}_C$ to ζ^{or}_C, Sel_C to ζ^{sel}_C and Create_C to ζ^{new}_C.

Lemma 3.11. *There exists an FO theory* S_Δ *over* Σ^s_Δ *such that there is a one-to-one correspondence between* Δ*-selections in* D *and models of* S_Δ *with domain* D*.*

Proof. This theory contains sentences that express that Sel_C is functional, and that Create_C is a partial function. It is straightforward to do this in FO (with among others, constraints such as $\forall\overline{x} : \exists y : \mathrm{Sel}_C(\overline{x}, y)$). Furthermore, it is also easy to express that the Create_C functions are injective, and that different **New**-expressions create different elements. Finally, this theory relates $\mathcal{S}$ to the Create_C expressions: $\forall y : \mathcal{S}(y) \Leftrightarrow \neg \bigvee_C (\exists\overline{x} : \mathrm{Create}_C(\overline{x}, y))$ where the disjunction ranges over all occurrences C of **New**-expressions. □

The condition that a causal theory succeeds can also be expressed as an FO theory. For that, we need one more definition.

Definition 3.12. *Let* Δ *be a causal theory in NestNF and let* C *be one of the* C'_i *in definition 3.5, then we call* φ_i *(again, from definition 3.5) the* relevance condition *of* C *and denote it* Rel_C*.*

In what follows, we define one more extended vocabulary. First, we use it to express the constraints that Δ succeeds and afterwards, for the actual transformation.

Definition 3.13. *The* Δ-transformed vocabulary Σ^t_Δ *is the disjoint union of* Σ *and* Σ^s_Δ *extended with the unary predicate symbol* $\mathcal{U}$*.*

Lemma 3.14. *Suppose* Δ *is a causal theory in NestNF, and* ζ *is a* Δ*-selection with corresponding* Σ^s_Δ*-structure* M*. There exists an FO theory* Succ_Δ *such that for every (two-valued) structure* I *with* $I|_{\Sigma^s_\Delta} = M$*,* Δ *succeeds with respect to* I *and* ζ *iff* $I \models \mathrm{Succ}_\Delta$*.*

Proof. Δ is in NestNF; for every of the C'_i (as in Definition 3.5), $\mathrm{Rel}_{C'_i}$ is true in I if and only if C'_i is relevant. Hence, for Succ_Δ we can take the FO theory consisting of the following sentences:

- $\forall\overline{x} : \mathrm{Rel}_C \Rightarrow \exists y : \mathrm{Create}_C(\overline{x}, y)$, for all **New**-expressions $C\langle\overline{x}\rangle$ in Δ,
- $\forall\overline{x} : \mathrm{Rel}_C \Rightarrow \exists y : (\mathrm{Sel}_C(\overline{x}, y) \wedge \psi)$, for all **Select**-expressions $C\langle\overline{x}\rangle$ of the form $\mathbf{Select}\ y[\psi] : C'$ in Δ. □

Now we describe the actual transformation: we translate every quantification into a relativised version, make explicit that a **New**-expression causes an atom $\mathcal{U}(d)$, and eliminate all non-determinism using the predicates in Σ^s_Δ.

Definition 3.15. *Let* Δ *be a* C-LOG *theory over* Σ *in NestNF. The* transformed theory Δ^t *is the theory obtained from* Δ *by applying the following transformation:*

- *first replacing all quantifications* $\alpha x[\psi] : \chi$*, where* $\alpha \in \{\forall, \exists, \mathbf{Select}, \mathbf{All}\}$ *by* $\alpha x[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge \psi] : \chi$
- *subsequently replacing each occurrence* $C\langle\overline{x}\rangle$ *of an expression* $\mathbf{New}\,y : C'$ *by* $\mathbf{All}\,y[\mathrm{Create}_C(\overline{x}, y)] : \mathcal{U}(y)\ \mathbf{And}\ C'$*,*
- *replacing every occurrence* $C\langle\overline{x}\rangle$ *of an expression* $C_1\ \mathbf{Or}\ C_2$ *by* $(C_1 \leftarrow \mathrm{Choose1}_C(\overline{x}))\mathbf{And}(C_2 \leftarrow \neg\mathrm{Choose1}_C(\overline{x}))$*,*

- *and replacing every occurrence $C\langle\overline{x}\rangle$ of an expression* $\mathbf{Select}\, y[\varphi] : C'$ *by* $\mathbf{All}\, y[\varphi \wedge \mathrm{Sel}_C(\overline{x}, y)] : C'$.

Given a structure I and a Δ-selection ζ, there is an obvious lattice morphism $m_\zeta : L^{\Sigma}_{I,\zeta} \to L^{\Sigma^t_\Delta}_{I}$ mapping a structure J to the structure J' with domain $D^{J'} = D^I$ interpreting all symbols in Σ^s_Δ according to ζ (as in Lemma 3.11), all symbols in Σ (except for the domain) the same as I and interpreting $\mathcal{U}$ as $D^J \setminus \mathcal{S}^{J'}$. m_ζ can straightforwardly be extended to a bilattice morphism.

Lemma 3.16. *Let ζ be a Δ-selection for Δ and A_ζ and A be the partial immediate causality operators of Δ and Δ^t respectively. Let J be any partial structure in $(L^{\Sigma}_{I,\zeta})^2$. Then $m_\zeta(A_\zeta(J)) = A(m_\zeta(J))$.*

Idea of the proof. $\mathbf{New}$-expressions $\mathbf{New}\, y : C'$ in Δ have been replaced by $\mathbf{All}$ expressions causing two subexpressions: $\mathcal{U}(y)$ and the C' for exactly the y's that are created according to ζ. Furthermore, the relativisation of all other quantifications guarantees that we correctly evaluate all quantifications with respect to the domain of J, encoded in $\mathcal{S} \cup \mathcal{U}$.

Furthermore, all non-deterministic expressions have been changed into $\mathbf{All}$-expressions that are conditionalised by the Δ-selection; this does not change the effect set; thus, the operators correspond. □

Lemma 3.17. *Let ζ, A_ζ and A be as in lemma 3.16. If I is the well-founded model of A_ζ, $m_\zeta(I)$ is the well-founded model of A.*

Proof. Follows directly from lemma 3.16: the mapping $J \mapsto m_\zeta(J)$ is an isomorphism between $L^{\Sigma}_{I,\zeta}$ and the sublattice of $L^{\Sigma^t_\Delta}_{I,\zeta'}$ consisting of those structures such that the interpretations of $\mathcal{S}$ and $\mathcal{U}$ have an empty intersection. As this isomorphism maps A_ζ to A, their well-founded models must agree. □

Lemma 3.18. *Let Δ be a causal theory in NestNF, ζ a Δ-selection for Δ and I a Σ-structure. Then $I \models \Delta$ if and only if $m_\zeta(I) \models \Delta^t$ and $m_\zeta(I) \models \mathrm{S}_\Delta$ and $m_\zeta(I) \models \mathrm{Succ}_\Delta$.*

Proof. Follows directly from Lemmas 3.17, 3.11 and 3.14. □

Proof of Lemma 3.9. Let Δ be the C-LOG theory in $\mathcal{T}$. We can now take as deterministic theory the theory consisting of Δ^t, all FO sentences in $\mathcal{T}$, and the sentence $\mathrm{S}_\Delta \wedge \mathrm{Succ}_\Delta \wedge \forall x : \mathcal{S}(x) \Leftrightarrow \neg\mathcal{U}(x)$, where the last formula excludes all structures not of the form $m_\zeta(I)$ for some I (the created elements $\mathcal{U}$ and the initial elements $\mathcal{S}$ should form a partition of the domain). □

3.4 From General FO(C) to NestNF

In the following definition we use $\Delta[C'/C]$ for the causal theory obtained from Δ by replacing the occurrence of a CEE C by C'.

Definition 3.19. *Suppose $C\langle\overline{x}\rangle$ is an occurrence of a CEE in Δ. With* $\mathtt{Unnest}(\Delta, C)$ *we denote the causal theory $\Delta[P(\overline{x})/C]$* $\mathbf{And}\, \mathbf{All}\, \overline{x}[P(\overline{x})] : C$ *where P is a new predicate symbol.*

Lemma 3.20. *Every* FO(C) *theory is Σ-equivalent with an* FO(C) *theory in NestNF.*

Proof. First, we claim that for every C-LOG theory over Σ, Δ and $\mathtt{Unnest}(\Delta, C)$ are Σ-equivalent. It is easy to see that the two theories have the same Δ-selections. Furthermore, the operator for $\mathtt{Unnest}(\Delta, C)$ is a part-to-whole monotone fixpoint extension[3] (as defined in (Vennekens et al. 2007)) of the operator for Δ. In (Vennekens et al. 2007) it is shown that in this case, their well-founded models agree, which proves our claim. The lemma now follows by repeated applications of the claim. □

Proof of Theorem 3.6. Follows directly by combining lemmas 3.20, 3.9 and 3.8. For transformations only defined on C-LOG theories, the extra FO part remains unchanged. □

3.5 FO(C) and FO(ID)

An inductive definition (ID) (Denecker and Ternovska 2008) is a set of rules of the form $\forall\overline{x} : P(\bar{t}) \leftarrow \varphi$, an FO($ID$) theory is a set of FO sentences and IDs, and an $\exists$SO(ID) theory is a theory of the form $\exists\bar{P} : \mathcal{T}$, where $\mathcal{T}$ is an FO(ID) theory. A causal theory in DefF corresponds exactly to an ID: the CEE $\mathbf{All}\, \overline{x}[\varphi] : P(\bar{t})$ corresponds to the above rule and the $\mathbf{And}$-conjunction of such CEEs to the set of corresponding rules. The partial immediate consequence operator for IDs defined in (Denecker and Ternovska 2008) is exactly the partial immediate causality operator for the corresponding C-LOG theory. Combining this with Theorem 3.6, we find (with $\bar{P}$ the introduced symbols):

Theorem 3.21. *Every* FO(C) *theory is equivalent with an* $\exists$SO(ID) *formula of the form $\exists\bar{P} : \{\Delta, \mathcal{T}\}$, where Δ is an ID and $\mathcal{T}$ is an FO sentence.*

Theorem 3.21 implies that we can use reasoning engines for FO(ID) in order to reason with FO(C), as long as we are careful with the newly introduced predicates. We implemented a prototype of this transformation in the IDP system (De Cat et al. 2014), it can be found at (Bogaerts 2014).

4 Example: Natural Numbers

Example 4.1. Let Σ be a vocabulary consisting of predicates $\mathrm{Nat}/1, \mathrm{Succ}/2$ and $\mathrm{Zero}/1$ and suppose $\mathcal{T}$ is the following theory:

$$\left\{ \begin{array}{l} \mathbf{New}\, x : \mathrm{Nat}(x)\, \mathbf{And}\, \mathrm{Zero}(x) \\ \mathbf{All}\, x[\mathrm{Nat}(x)] : \mathbf{New}\, y : \mathrm{Nat}(y)\, \mathbf{And}\, \mathrm{Succ}(x, y) \end{array} \right\}$$

[3] Intuitively, a part-to-whole fixpoint extension means that all predicates only depend positively on the newly introduced predicates

This theory defines a process creating the natural numbers. Transforming it to NestNF yields:

$$\left\{\begin{array}{l}\mathbf{New}\, x : T_1(x)\\ \mathbf{All}\, x[T_1(x)] : \mathrm{Nat}(x)\\ \mathbf{All}\, x[T_1(x)] : \mathrm{Zero}(x)\\ \mathbf{All}\, x[\mathrm{Nat}(x)] : \mathbf{New}\, y : T_2(x,y)\\ \mathbf{All}\, x,y[T_2(x,y)] : \mathrm{Nat}(y)\\ \mathbf{All}\, x,y[T_2(x,y)] : \mathrm{Succ}(x,y),\end{array}\right\}$$

where T_1 and T_2 are auxiliary symbols. Transforming the resulting theory into deterministic C-LOG requires the addition of more auxiliary symbols $\mathcal{S}/1, \mathcal{U}/1, \mathrm{Create}_1/1$ and $\mathrm{Create}_2/2$ and results in the following C-LOG theory (together with a set of FO-constraints):

$$\left\{\begin{array}{l}\mathbf{All}\, x[\mathrm{Create}_1(x)] : \mathcal{U}(x)\ \mathbf{And}\ T_1(x)\\ \mathbf{All}\, x[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge T_1(x)] : \mathrm{Nat}(x)\\ \mathbf{All}\, x[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge T_1(x)] : \mathrm{Zero}(x)\\ \mathbf{All}\, x,y[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge \mathrm{Nat}(x) \wedge \mathrm{Create}_2(x,y)] :\\ \quad \mathcal{U}(y)\ \mathbf{And}\ T_2(x,y)\\ \mathbf{All}\, x,y[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge (\mathcal{U}(y) \vee \mathcal{S}(y)) \wedge T_2(x,y)] :\\ \quad \mathrm{Nat}(y)\\ \mathbf{All}\, x,y[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge (\mathcal{U}(y) \vee \mathcal{S}(y)) \wedge T_2(x,y)] :\\ \quad \mathrm{Succ}(x,y)\end{array}\right\}$$

This example shows that the proposed transformation is in fact too complex. E.g., here, almost all occurrences of $\mathcal{U}(x) \vee \mathcal{S}(x)$ are not needed. This kind of redundancies can be eliminated by executing the three transformations (from Sections 3.2, 3.3 and 3.4) simultaneously. In that case, we would get the simpler deterministic theory:

$$\left\{\begin{array}{l}\mathbf{All}\, x[\mathrm{Create}_1(x)] : \mathrm{Nat}(x)\ \mathbf{And}\ \mathrm{Zero}(x)\ \mathbf{And}\ \mathcal{U}(x)\\ \mathbf{All}\, x,y[(\mathcal{U}(x) \vee \mathcal{S}(x)) \wedge \mathrm{Nat}(x) \wedge \mathrm{Create}_2(x,y)] :\\ \quad \mathrm{Nat}(y)\ \mathbf{And}\ \mathrm{Succ}(x,y)\ \mathbf{And}\ \mathcal{U}(y)\end{array}\right\}$$

with several FO sentences:

$\forall x : \mathcal{U}(x) \Leftrightarrow \neg\mathcal{S}(x)$
$\forall y : \mathcal{S}(y) \Leftrightarrow \neg(\mathrm{Create}_1(y) \vee \exists x : \mathrm{Create}_2(x,y))$.
$\exists x : \mathrm{Create}_1(x)$.
$\forall x,y : \mathrm{Create}_1(x) \wedge \mathrm{Create}_1(y) \Rightarrow x = y$.
$\forall x,y,z : \mathrm{Create}_2(x,y) \wedge \mathrm{Create}_1(x,z) \Rightarrow y = z$.
$\forall x,y,z : \mathrm{Create}_1(y) \wedge \mathrm{Create}_1(x,z) \Rightarrow y = z$.
$\forall x[\mathrm{Nat}(x)] : \exists y : \mathrm{Create}_2(x,y)$.

These sentences express the well-known constraints on $\mathbb{N}$: there is at least one natural number (identified by Create_1), and every number has a successor. Furthermore the initial element and the successor elements are unique, and all are different. Natural numbers are defined as zero and all elements reachable from zero by the successor relation. The theory we started from is much more compact and much more readable than any FO(ID) theory defining natural numbers. This shows the Knowledge Representation power of C-LOG.

5 Complexity Results

In this section, we provide complexity results. We focus on the C-LOG fragment of FO(C) here, since complexity for FO is well-studied. First, we formally define the inference methods of interest.

5.1 Inference Tasks

Definition 5.1. *The* model checking *inference takes as input a* C-LOG *theory Δ and a finite (two-valued) structure I. It returns true if $I \models \Delta$ and false otherwise.*

Definition 5.2. *The* model expansion *inference takes as input a* C-LOG *theory Δ and a partial structure I with finite two-valued domain. It returns a model of Δ more precise than I if one exists and "unsat" otherwise.*

Definition 5.3. *The* endogenous model expansion *inference is a special case of model expansion where I is two-valued on exogenous symbols of Δ and completely unknown on endogenous symbols.*

The next inference is related to database applications. In the database world, languages with object creation have also been defined (Abiteboul, Hull, and Vianu 1995). A query in such a language can create extra objects, but the interpretation of exogenous symbols (tables in the database) is fixed, i.e., exogenous symbols are always false on newly created elements.

Definition 5.4. *The* unbounded query *inference takes as input a* C-LOG *theory Δ, a partial structure I with finite two-valued domain such that I is two-valued on exogenous symbols of Δ and completely unknown on endogenous symbols of Δ, and a propositional atom P. This inference returns true if there exist i) a structure J, with $D^J \supseteq D^I$, $\sigma^J = \sigma^I$ for exogenous symbols σ, and $P^J = \mathbf{t}$ and ii) a Δ-selection ζ in D^J with $\zeta^{in} = D^I$, such that J is a model of Δ with Δ-selection ζ. It returns false otherwise.*

5.2 Complexity of Inference Tasks

In this section, we study the datacomplexity of the above inference tasks, i.e., the complexity for fixed Δ.

Lemma 5.5. *For a finite structure I, computing $A_\zeta(I)$ is polynomial in the size of I and ζ.*

Proof. In order to compute $A_\zeta(I)$, we need to evaluate a fixed number of FO-formulas a polynomial number of times (with exponent in the nesting depth of Δ). As evaluating a fixed FO formula in the context of a partial structure is polynomial, the result follows. □

Theorem 5.6. *For a finite structure I, the task of computing the A_ζ-well-founded model of Δ in the lattice $L^{\Sigma}_{I,\zeta}$ is polynomial in the size of I and ζ.*

Proof. Calculating the well-founded model of an approximator can be done with a polynomial number of applications of the approximator. Furthermore, Lemma 5.5 guarantees that each of these applications is polynomial as well. □

Theorem 5.7. *Model expansion for* C-LOG *is NP-complete.*

Proof. After guessing a model and a Δ-selection, Theorem 5.6 guarantees that checking that this is the well-founded model is polynomial. Lemma 3.14 shows that checking whether Δ succeeds is polynomial as well. Thus, model expansion is in NP.

NP-hardness follows from the fact that model expansion for inductive definitions is NP-hard and inductive definitions are shown to be a subclass of C-LOG theories, as argued in Section 3.5. □

Example 5.8. We show how the SAT-problem can be encoded as model checking for C-LOG. Consider a vocabulary Σ_{IN}^{SAT} with unary predicates Cl and PS and with binary predicates Pos and Neg. Every SAT-problem can be encoded as a Σ_{IN}^{SAT}-structure: Cl and PS are interpreted as the sets of clauses and propositional symbols respectively, $\mathrm{Pos}(c,p)$ (respectively $\mathrm{Neg}(c,p)$) holds if clause c contains the literal p (respectively $\neg p$).

We now extend Σ_{IN}^{SAT} to a vocabulary Σ_{ALL}^{SAT} with unary predicates Tr and Fa and a propositional symbol Sol. Tr and Fa encode an assignment of values (true or false) to propositional symbols, Sol means that the encoded assignment is a solution to the SAT problem. Let Δ_{SAT} be the following causal theory:

$$\begin{aligned}&\mathbf{All}\, p[\mathrm{PS}(p)] : \mathrm{Tr}(p)\ \mathbf{Or}\ \mathrm{Fa}(p)\\ &\mathrm{Sol} \leftarrow \forall c[\mathrm{Cl}(c)] : \exists p :\\ &\qquad (\mathrm{Pos}(c,p) \wedge \mathrm{Tr}(p) \vee (\mathrm{Neg}(c,p) \wedge \mathrm{Fa}(p))\end{aligned}$$

The first rules guesses an assignment. The second rule says that Sol holds if every clause has at least one true literal. Model expansion of that theory with a structure interpreting Σ_{IN}^{SAT} according to a SAT problem and interpreting Sol as true, is equivalent with solving that SAT problem, hence model expansion is NP-hard (which we already knew). In order to show that model *checking* is NP-hard, we add the following CEE to the theory Δ_{SAT}.

$$(\mathbf{All}\, p[\mathrm{PS}(p)] : \mathrm{Tr}(p)\ \mathbf{And}\ \mathrm{Fa}(p)) \leftarrow \mathrm{Sol}$$

Basically, this rules tells us to forget the assignment once we have derived that it is a model (i.e., we hide the witness of the NP problem). Now, the original SAT problem has a solution if and only if the structure interpreting symbols in Σ_{IN}^{SAT} according to a SAT problem and interpreting all other symbols as constant true is a model of the extended theory. Hence:

Theorem 5.9. *Model checking for* C-LOG *is NP-complete.*

Model checking might be a hard task but in certain cases (including for Δ_{SAT}) endogenous model expansion is not. The results in Theorem 5.6 can sometimes be used to generate models, if we have guarantees to end in a state where Δ succeeds.

Theorem 5.10. *If Δ is a total*[4] *causal theory without* **New** *and* **Select***-expressions, endogenous model expansion is in P.*

[4]A causal theory is *total* if for every Δ-selection ζ, $w(A_\zeta)$ is two-valued, i.e., roughly, if it does not contain relevant loops over negation.

Note that Theorem 5.10 does not contradict Example 5.8 since in that example, Sol is interpreted as true in the input structure, i.e., the performed inference is not endogenous model expansion. It is future work to generalise Theorem 5.10, i.e., to research which are sufficient restrictions on Δ such that model expansion is in P.

It is a well-known result in database theory that query languages combining recursion and object-creation are computationally complete (Abiteboul, Hull, and Vianu 1995); C-LOG can be seen as such a language.

Theorem 5.11. *Unbounded querying can simulate the language* $\mathrm{while}_{\mathrm{new}}$ *from (Abiteboul, Hull, and Vianu 1995).*

Proof. We already showed that we can create the natural numbers in C-LOG. Once we have natural numbers and the successor function Succ, we add one extra argument to every symbol (this argument represents time). Now, we encode the looping construct from $\mathrm{while}_{\mathrm{new}}$ as follows. An expression of the form `while P do s` corresponds to the CEE: $\mathbf{All}\, t[P(t)] : C$, where C is the translation of the expression s. An expression `P = new Q` corresponds to a CEE (where the variable t should be bound by a surrounding while).

$$\mathbf{All}\, \overline{x}, t'[\mathrm{Succ}(t,t')] : \mathbf{New}\, y : P(\overline{x}, y, t') \leftarrow Q(\overline{x}, t).\ \square$$

Now, it follows immediately from (Abiteboul, Hull, and Vianu 1995) that

Corollary 5.12. *For every decidable class $\mathcal{S}$ of finite structures closed under isomorphism, there exists a Δ such that unbounded exogenous model generation returns true with input I iff $I \in \mathcal{S}$.*

6 Conclusion

In this paper we presented several normal forms for FO(C). We showed that every FO(C) theory can be transformed to a Σ-equivalent deterministic FO(C) theory and to a Σ-equivalent FO(C) theory in NestNF or in DefF. Furthermore, as FO(C) theories in DefF correspond exactly to FO(ID), these transformations reduce inference for FO(C) to FO(ID). We implemented a prototype of this above transformation, resulting in the first FO(C) solver. We also gave several complexity results for inference in C-LOG. All of these results are valuable from a theoretical point of view, as they help to characterise FO(C), but also from a practical point of view, as they provide more insight in FO(C).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Bogaerts, B.; Vennekens, J.; Denecker, M.; and Van den Bussche, J. (in press) 2014. FO(C): A knowledge representation language of causality. *Theory and Practice of Logic Programming (TPLP)* (Online-Supplement, Technical Communication ICLP14).

Bogaerts, B. 2014. IDP-CLog. `http://dtai.cs.kuleuven.be/krr/files/software/various/idp-clog.tar.gz`.

De Cat, B.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2014. Predicate logic as a modelling language: The IDP system. *CoRR* abs/1401.6312.

Denecker, M., and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)* 9(2):14:1–14:52.

Denecker, M.; Bruynooghe, M.; and Vennekens, J. 2012. Approximation fixpoint theory and the semantics of logic and answers set programs. In Erdem, E.; Lee, J.; Lierler, Y.; and Pearce, D., eds., *Correct Reasoning*, volume 7265 of *Lecture Notes in Computer Science*. Springer.

Denecker, M. 2012. The FO(·) knowledge base system project: An integration project (invited talk). In *ASPOCP*.

Kleene, S. C. 1938. On notation for ordinal numbers. *The Journal of Symbolic Logic* 3(4):pp. 150–155.

Preyer, G., and Peter, G. 2002. *Logical Form and Language*. Clarendon Press.

Vennekens, J.; Mariën, M.; Wittocx, J.; and Denecker, M. 2007. Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae* 79(1-2):187–208.

FO(C) and Related Modelling Paradigms

Bart Bogaerts and **Joost Vennekens** and **Marc Denecker**
Department of Computer Science, KU Leuven
{bart.bogaerts, joost.vennekens, marc.denecker}@cs.kuleuven.be

Jan Van den Bussche
Hasselt University & transnational University of Limburg
jan.vandenbussche@uhasselt.be

Abstract

Recently, C-LOG was introduced as a language for modelling causal processes. Its formal semantics has been defined, but the study of this language is far from finished. In this paper, we compare C-LOG to other declarative modelling languages. More specifically, we compare to first-order logic (FO), and argue that C-LOG and FO are orthogonal and that their integration, FO(C), is a knowledge representation language that allows for clear and succinct models. We compare FO(C) to E-disjunctive logic programming with the stable semantics, and define a fragment on which both semantics coincide. Furthermore, we discuss object-creation in FO(C), relating it to mathematics, business rules systems, and data base systems.

1 Introduction

Previous work introduced C-LOG (Bogaerts et al. in press 2014a), an expressive language construct to describe causal processes, and FO(C), its integration with classical logic. In that work, it is indicated that C-LOG shows similarities to many other languages and it is suggested that C-LOG could serve as a tool to study the semantical relationship between these languages. In this paper, we take the first steps for such a study: we discuss the relationship of FO(C) with other paradigms and through this discussion, provide a comprehensive overview of the informal semantics of FO(C).

C-LOG and FO are syntactically very similar, but semantically very different languages. In this paper we formalise the semantical relationship between C-LOG and FO, and argue how their integration, FO(C), is a rich language in which knowledge can be represented succinctly and clearly.

We explain how modelling in FO(C) relates to the "generate, define, and test" methodology used in answer set programming. We discuss how FO(C) relates to disjunctive logic programs with existential quantification in rule heads (You, Zhang, and Zhang 2013), both informally and formally, and we identify a subset of E-disjunctive logic programs on which stable semantics corresponds to the FO(C) semantics. We also discuss four important knowledge representation constructs that FO(C) adds with respect to E-disjunctive logic programs: *nested rules* (in fact, arbitrary nesting of expressions), *dynamic choice*, *object creation*, and *a more modular semantics*.

Furthermore, we discuss object-creation in related paradigms. One of those discussed paradigms is the field of deductive databases, where extensions of Datalog have been defined. In (Abiteboul and Vianu 1991), rules with existentially quantified head variables are used for object creation. It is remarkable to see how the same extension of logic programs is used sometimes (e.g., in (You, Zhang, and Zhang 2013)) for selection, and sometimes (e.g., in (Abiteboul and Vianu 1991)) for object-creation. Consider for example a rule

$$\forall X : \exists Y : P(X,Y) \text{ :- } q(X).$$

Viewing this rule as a rule in an E-disjunctive logic program, it corresponds to the C-LOG expression

$$\mathbf{All}\, X[q(X)] : \mathbf{Select}\, Y[\mathbf{t}] : P(X,Y),$$

where for every X satisfying q, one existing value Y is selected, and $P(X,Y)$ is caused. The selected Y can be different or equal for different X's. On the other hand, in case this same rule occurs in a LogicBlox (Green, Aref, and Karvounarakis 2012) specification, it corresponds to the C-LOG expression

$$\mathbf{All}\, X[q(X)] : \mathbf{New}\, Y : P(X,Y),$$

where for every X satisfying q a new value Y is invented. Thus implying among others that all of these values are different. The explicit distinction C-LOG makes between object-creation and selection is necessary for studying the relationship between these languages.

The rest of this paper is structured as follows. In Section 2 we give preliminaries, including the syntax and informal semantics of C-LOG. In Sections 3 and 4, we focus on the creation-free fragment of C-LOG, i.e., on expressions without the **New**-operator: first, we compare C-LOG to FO and discuss the integration of these two; afterwards, we compare C-LOG to E-disjunctive logic programs. In Section 5, we discuss object-creation in C-LOG by providing simple intuitive examples and relating the **New**-operator to other languages with similar forms of object-creation. We conclude in Section 6.

2 C-LOG

We assume familiarity with the basics of first-order logic. Vocabularies, formulas, and terms are defined as usual. We

use **t** for truth and **f** for falsity. $\sigma^{\mathcal{I}}$ denotes the interpretation of symbol σ in structure $\mathcal{I}$. *Domain atoms* are atoms of the form $P(\bar{d})$ where the d_i are domain elements. We use restricted quantifications (Preyer and Peter 2002), e.g., in FO, these are formulas of the form $\forall x[\psi] : \varphi$ or $\exists x[\psi] : \varphi$, meaning that φ holds for all (resp. for a) x such that ψ holds. The above expressions are syntactic sugar for $\forall x : \psi \Rightarrow \varphi$ and $\exists x : \psi \wedge \varphi$, but such a reduction is not possible for other restricted quantifiers in C-LOG. We call ψ the *qualification* and φ the *assertion* of the restricted quantifications. From now on, let Σ be a relational vocabulary, i.e., Σ consists only of predicate, constant and variable symbols.

In what follows we briefly repeat the syntax and informal semantics of C-LOG. For more details and an extensive overview of the formal semantics of C-LOG, we refer to (Bogaerts et al. in press 2014a).

2.1 Syntax of C-LOG

Definition 2.1. Causal effect expressions *(CEE) are defined inductively as follows:*

- *if $P(\bar{t})$ is an atom, then $P(\bar{t})$ is a CEE,*
- *if φ is an FO formula and C' is a CEE, then $C' \leftarrow \varphi$ is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 $\mathbf{And}$ C_2 is a CEE,*
- *if C_1 and C_2 are CEEs, then C_1 $\mathbf{Or}$ C_2 is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then $\mathbf{All}\, x[\varphi] : C'$ is a CEE,*
- *if x is a variable, φ is a first-order formula and C' is a CEE, then $\mathbf{Select}\, x[\varphi] : C'$ is a CEE,*
- *if x is a variable and C' is a CEE, then $\mathbf{New}\, x : C'$ is a CEE.*

We call a CEE an *atom-expression* (respectively *rule-*, **And**-, **Or**-, **All**-, **Select**- or **New**-*expression*) if it is of the corresponding form. We use $\mathbf{All}\, \bar{x}[\varphi] : C$ as an abbreviation for $\mathbf{All}\, x_1[\mathbf{t}] : \ldots \mathbf{All}\, x_n[\varphi] : C$ and similar for **Select**-expressions. We call a predicate symbol P *endogenous* in C if P occurs as the symbol of a (possibly nested) atom-expression in C, i.e., if P occurs in C but not only in first-order formulas. All other symbols are called *exogenous* in C. An occurrence of a variable x is *bound* in a CEE if it occurs in the scope of a quantification over that variable ($\forall x$, $\exists x$, $\mathbf{All}\, x$, $\mathbf{Select}\, x$, or $\mathbf{New}\, x$) and *free* otherwise. A variable is *free* in a CEE if it has free occurrences. A *causal theory*, or C-LOG *theory* is a CEE without free variables. We often represent a causal theory as a set of CEEs; the intended causal theory is the **And**-conjunction of these CEEs.

2.2 Informal Semantics of C-LOG

In this section, we discuss the informal semantics of CEEs. We repeat the driving principles on a simple example—one without non-determinism—and discuss more complex expressions afterwards.

Driving Principles Following the philosophy of (Vennekens, Denecker, and Bruynooghe 2009), the semantics of C-LOG is based on two principles that are common in causal modelling. The first is the distinction between *endogenous* and *exogenous* properties, i.e., those whose value is determined by the causal laws in the model and those whose value is not, respectively (Pearl 2000). The second is the *default-deviant* assumption, used also by, e.g., (Hall 2004; Hitchcock 2007). The idea here is to assume that each endogenous property of the domain has some "natural" state, that it will be in whenever nothing is acting upon it. For ease of notation, C-LOG identifies the default state with falsity, and the deviant state with truth. For example, consider the following simplified model of a bicycle, in which a pair of gear wheels can be put in motion by pedalling:

$$Turn(BigGear) \leftarrow Pedal. \quad (1)$$

$$Turn(BigGear) \leftarrow Turn(SmallGear). \quad (2)$$

$$Turn(SmallGear) \leftarrow Turn(BigGear). \quad (3)$$

Here, $Pedal$ is exogenous, while $Turn(BigGear)$ and $Turn(SmallGear)$ are endogenous. The semantics of this causal model is given by a straightforward "execution" of the rules. The domain starts out in an initial state, in which all endogenous atoms have their default value *false* and the exogenous atom $Pedal$ has some fixed value. If $Pedal$ is true, then the first rule is applicable and may be fired ("$Pedal$ causes $Turn(BigGear)$") to produce a new state of the domain in which $Turn(BigGear)$ now has its deviant value *true*. In this way, we construct the following sequence of states (we abbreviate symbols by their first letter):

$$\{P\} \rightarrow \{P, T(B)\} \rightarrow \{P, T(B), T(S)\} \quad (4)$$

In general, given a causal theory Δ, a causal process is a (possibly transfinite) sequence of intermediate states, starting from the default state such that, at each state, the effects described by Δ take place. This notion of causal process is based on the following principles:

- The principle of *sufficient causation* states that if the precondition to a causal law is satisfied, then the event that it triggers must eventually happen. For example, the process described in (4) cannot stop after the first step: there is a cause for $Turn(SmallGear)$, hence this should eventually happen.
- The principle of *universal causation* states that all changes to the state of the domain must be triggered by a causal law whose precondition is satisfied. For example, the small gear can only turn if the big gear turns.
- The principle of *no self-causation* states that nothing can happen based on itself. E.g., if rule (1) would be excluded from the causal theory, the gears cannot start rotating by themselves.

Complex Expressions A (possibly infinite) structure is a model of a causal theory Δ if it is the final state of a (non-deterministic) causal processes described by Δ. In order to define these processes correctly, one should know the events that take place in every state. We call the set of those events the *effect set* of the causal theory. There are two kinds of effects that can be described by a causal theory: 1) flipping an atom from its default to its deviant state and 2) creating a new domain element. We now explain in a compositional

way what the effect set of a causal theory is in a given state of affairs, which we represent as usual by a structure.

The effect of an atom-expression A is that A is flipped to its deviant state. A conditional effect, i.e., a rule expression, causes the effect set of its head if its body is satisfied in the current state, and nothing otherwise. The effect set described by an **And**-expression is the union of the effect sets of its two subexpressions; an **All**-expression $\mathbf{All}\, x[\varphi] : C'$ causes the union of all effect sets of $C'(x)$ for those x's that satisfy φ. An expression $C_1\, \mathbf{Or}\, C_2$ non-deterministically causes either the effect set of C_1 or the effect set of C_2; a **Select**-expression $\mathbf{Select}\, x[\varphi] : C'$ causes the effect set of C' for a non-deterministically chosen x that satisfies φ. An object-creating CEE $\mathbf{New}\, x : C'$ causes the creation of a new domain element n and the effect set of $C'(n)$.

Informally, CEEs only cause changes to the state once (for each of its instantiations), e.g., a **Select**-expression $\mathbf{Select}\, x[\varphi] : C'$ causes the effect set of C' for a non-deterministically chosen x once, and cannot cause C' for another x afterwards.

Example 2.2. Permanent residence in the United States can be obtained in several ways. One way is passing the naturalisation test. Another way is by playing the "Green Card Lottery", where each year a number of lucky winners are randomly selected and granted permanent residence. We model this as follows:

$$\left\{ \begin{array}{l} \mathbf{All}\, p[Apply(p) \wedge PassedTest(p)] : PermRes(p) \\ (\mathbf{Select}\, p[Play(p)] : PermRes(p)) \leftarrow Lottery. \end{array} \right\}$$

The first CEE describes the "normal" way to obtain permanent residence; the second rule expresses that one winner is selected among everyone who plays the lottery. If $\mathcal{I}$ is a structure in which $Lottery$ holds, due to the non-determinism, there are many possible effect sets of the above CEE, namely the sets $\{PermRes(p) \mid p \in Apply^{\mathcal{I}} \wedge p \in PassedTest^{\mathcal{I}}\} \cup \{PermRes(d)\}$ for some $d \in Play^{\mathcal{I}}$.

Models of this causal theory are structures such that everyone who applies and passes the test has permanent residence, and in case the lottery happens, one random person who played the lottery as well, and such that furthermore no-one else obtains permanent residence. The principle of sufficient causation guarantees a form of closed world assumption: you can only obtain residence if there is a rule that causes you to obtain this nationality. The two CEEs are considered independent: the winner could be one of the people that obtained it through standard application, as well as someone else, i.e., the semantics allows both minimal and non-minimal models.

Note that in the above, there is a great asymmetry between $Play(p)$, which occurs as a qualification of **Select**-expression, and $PermRes(p)$, which occurs as a caused atom. This means that the effect will never cause atoms of the form $Play(p)$, but only atoms of the form $PermRes(p)$. This is one of the cases where the qualification of an expression cannot simply be eliminated.

Example 2.3. Hitting the "send" button in your mail application causes the creation of a new package containing a specific mail. That package is put on a channel and will be received some (unknown) time later. As long as the package is not received, it stays on the channel. In C-LOG, we model this as follows:

$$\left\{ \begin{array}{l} \mathbf{All}\, m, t[Mail(m) \wedge HitSend(m,t)] : \mathbf{New}\, p : \\ \quad Pack(p)\, \mathbf{And}\, Cont(p,m)\, \mathbf{And} \\ \quad OnCh(p, t+1)\, \mathbf{And} \\ \quad \mathbf{Select}\, d[d > 0] : Received(p, t+d) \\ \mathbf{All}\, p, t[Pack(p) \wedge OnCh(p,t) \wedge \neg Received(p,t)] : \\ \quad OnCh(p, t+1) \end{array} \right\}$$

Suppose an interpretation $HitSend^{\mathcal{I}} = \{(MyMail, 0)\}$ is given. A causal process then unfolds as follows: it starts in the initial state, where all endogenous predicates are false. The effect set of the above causal effect in that state consists of 1) the creation of one new domain element, say $\_p$, and 2) the caused atoms $Pack(\_p)$, $Cont(\_p, MyMail)$, $OnCh(\_p, 1)$ and $Received(\_p, 7)$, where instead of 7, we could have chosen any number greater than zero. Next, it continues, and in every step t, before receiving the package, an extra atom $OnCh(p, t+1)$ is caused. Finally, in the seventh step, no more atoms are caused; the causal process ends. The final state is a model of the causal theory.

2.3 FO(C)

First-order logic and C-LOG have a straightforward integration, FO(C). Theories in this logic are sets of FO sentences and causal theories. A model of such a theory is a structure that is a model of each of its expressions (of each of its CEEs and sentences). An illustration is the mail protocol from Example 2.3, which we can extend with the "observation" that at some time, two packages are on the channel:

$$\exists t, p_1, p_2[p_1 \neq p_2] : OnCh(p_1, t) \wedge OnCh(p_2, t).$$

Models of this theory represent states of affairs where at least once two packages are on the channel simultaneously. This entirely differs from **And**-conjoining our CEE with

$$\mathbf{Select}\, t, p_1, p_2[p_1 \neq p_2] : OnCh(p_1, t)\, \mathbf{And}\, OnCh(p_2, t).$$

The resulting CEE would have unintended models in which two packages suddenly appear on the channel for no reason. Note that in the definitions of C-LOG, we restricted attention to relational vocabularies. All the theory can straightforwardly be generalised as long as function symbols do not occur as endogenous symbols in CEEs, i.e., if they only occur in FO sentences or as exogenous symbols in causal theories.

3 C-LOG, FO, and FO(C)

There is an obvious syntactical correspondence between FO and creation-free C-LOG (C-LOG without **New**-expressions): **And** corresponds to $\wedge$, **Or** to $\vee$, $\leftarrow$ to $\Leftarrow$, **All** to $\forall$, and **Select** to $\exists$. As already mentioned above, expressions in C-LOG have an entirely different meaning than the corresponding FO expression. A C-LOG expression describes a process in which more and more facts are caused, while an FO expression describes a truth. For example $P\, \mathbf{Or}\, Q$ describes a process that picks either P or Q

and makes one of them true, hence its models are structures in which exactly one of the two holds. On the other hand, the FO sentence $P \vee Q$ has more models, namely also one in which both hold. We generalise this observation:

Theorem 3.1. *Let Δ be a creation-free causal theory over Σ and $\mathcal{T}_\Delta$ the corresponding FO theory (the theory obtained from Δ by replacing* **All** *by $\forall$,* **Select** *by $\exists$,* **Or** *by $\vee$,* **And** *by $\wedge$, and $\leftarrow$ by $\Leftarrow$). Then for every Σ-structure $\mathcal{I}$, if $\mathcal{I} \models \Delta$, then also $\mathcal{I} \models \mathcal{T}_\Delta$.*

The reverse often does not hold: there is no obvious way to translate any FO formula to a C-LOG expression. In some cases, it is possible to find an inverse transformation, for example for positive (negation-free) FO theories. This would yield a constructive way to create models for a positive FO theory, which is not a surprising, nor a very interesting result; another constructive way to get a model of such a theory would be to make everything true. But it is interesting to view C-LOG theories as a constructive way to create a certain structure. This shows that modelling in C-LOG is orthogonal to modelling in FO. In FO, by default everything is open, every atom can be true or false arbitrarily. Every constraint removes worlds from the set of possible worlds. In C-LOG on the other hand, all endogenous symbols are by default false. Adding extra rules to a C-LOG theory can result in more models (when introducing extra non-determinism), or modify worlds. In some cases, one of the approaches is more natural than the other.

Consider for example a steel oven scheduling problem. For every block of steel, we should find a time t to put that block in the oven and at time $t + D$, where D is some fixed delay, we take the block out. In C-LOG this is modelled as

$$\mathbf{All}\, b[Block(b)] : \mathbf{Select}\, t[\mathbf{t}] : In(b,t)\, \mathbf{And}\, Out(b,t+D),$$

but to model this in FO we would get one similar constraint together with several constraints guaranteeing uniqueness:

$$\begin{aligned}
&\forall b[Block(b)] : \exists t : In(b,t) \wedge Out(b,t+D)\\
&\forall b,t,t'[Block(b)] : In(b,t) \wedge In(b,t') \Rightarrow t = t'\\
&\forall b,t,t'[Block(b)] : Out(b,t) \wedge Out(b,t') \Rightarrow t = t'\\
&\forall x : (\exists t : In(x,t) \vee Out(x,t)) \Rightarrow Block(x)
\end{aligned}$$

Here, the approach in C-LOG is much more natural, as in this example it is clear how to construct a model, whereas to model it in FO, we should analyse all properties of models. On the other hand, if we extend this example with a constraint that no two blocks can enter the oven at the same time, this is easily expressible in FO:

$$\neg\exists t,b,b'[b \neq b'] : In(b,t) \wedge In(b',t),$$

while this is not naturally expressible in C-LOG. This shows the power of FO(C), the integration of FO and C-LOG. For example, the entire above scheduling problem would be modelled in FO(C) as follows (where we use "{" and "}" to separate the C-LOG theory from the FO sentences).

$$\left\{ \begin{array}{l} \mathbf{All}\, b[Block(b)] : \mathbf{Select}\, t[\mathbf{t}] : \\ \qquad In(b,t)\, \mathbf{And}\, Out(b,t+D) \end{array} \right\}$$
$$\neg\exists t,b,b'[b \neq b'] : In(b,t) \wedge In(b',t)$$

This is much more readable and much more concise than any pure C-LOG or FO expression that expresses the same knowledge. As can be seen, the integration of the orthogonal languages FO and C-LOG, FO(C) provides a great modelling flexibility.

4 FO(C) and ASP

The methodology from the previous section is very similar to the "generate, define, and test" (GDT) methodology used in Answer Set Programming (ASP). In that methodology, "generate" and "define" are constructive modules of ASP programs that describe which atoms can be true, while the "test" module corresponds to first-order sentences that constrain solutions. In (Denecker et al. 2012), it has been argued that GDT programs correspond to FO(ID) theories. Furthermore, in (Bogaerts et al. in press 2014a), we showed that FO(ID) is syntactically and semantically a sublanguage of FO(C). Here, we argue that a more general class of ASP programs can be seen as FO(C) theories.

E-disjunctive programs (You, Zhang, and Zhang 2013) are finite sets of rules of the form:

$$\forall\overline{x} : \exists\overline{y} : \alpha_1; \ldots; \alpha_m \text{ :- } \beta_1, \ldots, \beta_k, \text{not}\, \gamma_1, \ldots, \text{not}\, \gamma_n. \quad (5)$$

where the α_i, β_i and γ_i are atoms and variables in $\overline{y}$ only occur in the α_i. Given a structure $\mathcal{M}$, we define $\mathcal{M}^-$ as the literal set

$$\{\neg\alpha \mid \alpha \text{ is a domain atom on } dom(\mathcal{M}) \text{ and } \mathcal{M} \not\models \alpha\}.$$

A structure $\mathcal{M}$ is a stable model of E-disjunctive program $\mathcal{P}$ (denoted $\mathcal{M} \models \mathcal{P}$) if $\mathcal{M}$ is a minimal set X satisfying the condition: for any rule $r \in \mathcal{P}$ and any variable assignment η, if the literal set $X \cup \mathcal{M}^-$ logically entails $body(r)\eta$, then for some assignment θ, and for some α in the head of r, $(\alpha\eta|_{\overline{x}})\theta \in X$. A rule of the form (5) is called a *constraint* if $m = 0$.

Definition 4.1. *Let $\mathcal{P}$ be an E-disjunctive program. The* corresponding FO(C)*-theory is the theory $\mathcal{T}_\mathcal{P}$ with as* C-LOG *expression the* **And***-conjunction of all expressions*

$$\mathbf{All}\, \overline{x}[\beta_1 \wedge \cdots \wedge \neg\gamma_n] : \mathbf{Select}\, \overline{y}[\mathbf{t}] : \alpha_1\, \mathbf{Or}\, \ldots\, \mathbf{Or}\, \alpha_m$$

such that there is a rule of the form (5) with $m > 0$ in $\mathcal{P}$. $\mathcal{T}_\mathcal{P}$ has as FO part:

- *all sentences $\forall\overline{x} : \neg(\beta_1 \wedge \cdots \wedge \beta_k \wedge \neg\gamma_1 \wedge \ldots \wedge \neg\gamma_n)$ such that there is a rule of the form (5) with $m = 0$ (i.e., a constraint) in $\mathcal{P}$ and*
- *the sentences $\forall\overline{x} : \neg P(\overline{x})$ for symbols P that do not occur in the head of any rule in $\mathcal{P}$.*

The last type of constraint is a technical detail: in ASP, all symbols are endogenous, while in C-LOG, this is only the case for predicates occurring in "the head of rules".

The above syntactical correspondence does not always correspond to a semantical correspondence. Intuitively, an E-disjunctive rule r (roughly) means the following: if the body of r holds for an instantiation of $\overline{x}$, then we select one instantiation of the $\overline{y}$ and one disjunct; that disjunct is caused to be true for that instantiation. But, globally the selection should happen in such a way that the final model is minimal.

For example the program $\{p.\quad p; q.\}$ only has one stable model, namely $\{p\}$. The intuition behind it is that the first rule causes p to be true, and hence compromises the choice in the second rule. As p already holds, the global minimality condition ensures that the second rule is obliged to choose p as well, if possible. When we slightly modify the above program, by adding a constraint: $\{p.\quad p; q.\quad$:- not $q.\}$ suddenly, q can (and should) be chosen by the second rule, as $\{p\}$ no longer is a model of this theory. The above illustrates that there is a great interdependency between different rules and between rules and constraints: adding an extra rule or constraint changes the meaning of other rules. Below, we identify a fragment of E-disjunctive ASP in which this dependency is not too strong, and we show that for this fragment, the stable model semantics equals the FO(C) semantics. In order to do so, we introduce the following concepts:

Definition 4.2. *Let δ be a domain atom and r a rule in the form of (5). Suppose η is a variable assignment of the variables $\overline{x}$ and $\overline{y}$. We say that δ* occurs in *r at i for η if $\alpha_i\eta = \delta$. We say that δ* occurs in *r if there exist and i and an η such that r occurs at i for η.*

Definition 4.3. *We call a rule* disjunctive *if $\overline{y}$ is not the empty tuple or if $m > 1$.*

Definition 4.4. *An E-disjunctive program $\mathcal{P}$ is called* non-overlapping *if for every domain atom δ one of the following holds*

- *δ occurs only in non-disjunctive rules, or*
- *there are at most one rule r, one i, and one η such that δ occurs in r at i for η.*

The above condition states that domain atoms occurring in heads of disjunctive rules, cannot occur multiple times in rule heads. Intuitively, this guarantees that different choices do not interfere.

Theorem 4.5. *Let $\mathcal{P}$ be a non-overlapping E-disjunctive program without recursion over negation and $\mathcal{T}_\mathcal{P}$ the corresponding* FO(C) *theory. For every structure $\mathcal{I}$, $\mathcal{I} \models \mathcal{P}$ if and only if $\mathcal{I} \models \mathcal{T}_\mathcal{P}$.*

In Theorem 4.5, there is one extra condition on non-overlapping ASP programs to be equivalent to the corresponding FO(C) theory, namely that it does not contain recursion over negation, i.e., there are no rules of the form

$$p \text{ :- not } p'. \qquad p' \text{ :- not } p.$$

It has already been argued in (Denecker et al. 2012) that in practical applications recursion over negation is mostly for two purposes: 1) expressing constraints and 2) to "open" the predicate p, i.e., to encode that it can have arbitrary truth value. In this case, the predicate p' would not be used in the rest of the theory. This can as well be done with a rule $p; p'$. This last rule is equivalent to the above two in non-overlapping programs (or, if p and p' do not occur in other rule heads). In FO(C), we could either add the disjunctive rule, or simply omit this rule, since exogenous predicates are open anyway.

As already stated above, in case an ASP program is not non-overlapping, semantics might differ. However, we do have

Theorem 4.6. *Let $\mathcal{P}$ be any E-disjunctive program without recursion over negation and $\mathcal{T}_\mathcal{P}$ be the corresponding* FO(C) *theory. For every structure $\mathcal{I}$, if $\mathcal{I} \models \mathcal{P}$ then also $\mathcal{I} \models \mathcal{T}_\mathcal{P}$.*

The reverse does not hold, since C-LOG does not impose a global minimality condition. The difference in semantics is illustrated in the American Lottery example, which we resume below.

In the above, we argued that for many practical applications of E-disjunctive programs, semantics of FO(C) corresponds to the stable model semantics. This raises the question of relevance of FO(C). From a knowledge representation perspective, FO(C) adds several useful constructs with respect to E-disjunctive logic programs. Among these are nested rules (in fact, arbitrary nesting of expressions), dynamic choice, object creation, and a more modular semantics.

Nested causal rules occur in many places, for example, one could state that the electrician causes a causal link between a button and a light, e.g.,

$$(light \leftarrow button) \leftarrow electrician.$$

We found similar nested rules in (Kowalski and Sadri 2013). Of course, for simple examples this can also be expressed compactly in ASP, e.g. by

$$light \text{ :- } electrician, button.$$

but when causes and effects are more complex, translating them requires the introduction of auxiliary predicates, diminishing the readability of the resulting program.

Dynamic choices occur in many practical applications. Consider the following situation: a robot enters a room, opens some of the doors in this room, and then leaves by one of the doors that are open. The robot's leaving corresponds to a non-deterministic choice between a *dynamic* set of alternatives, which is determined by the robot's own actions, and therefore cannot be hard-coded into the head of a rule. In C-LOG, we would model this last choice as

$$\mathbf{Select}\, x[open(x)] : leave(x).$$

To model this in an E-disjunctive logic program, we need an extra auxiliary predicate, thus reducing readability:

$$\exists X : chosen(X).$$
$$\forall X : leave(X) \text{ :- } chosen(X).$$
$$\forall X \text{ :- } chosen(X); \text{not } open(X).$$

Modularity of the semantics has already been discussed above: The non-overlapping condition on ASP programs guarantees similar modularity. However, when the non-overlapping condition is violated, semantics of ASP programs are often less clear. Let us reconsider Example 2.2. The E-disjunctive program

$$\exists X : permres(X) \text{ :- } lottery.$$
$$\forall X : permres(X) \text{ :- } passtest(X).$$

is similar to

$$\left\{ \begin{array}{l} (\mathbf{Select}\, x[\mathbf{t}] : permres(x)) \leftarrow lottery \\ \mathbf{All}\, x[passtest(x)] : permres(x) \end{array} \right\}$$

Semantically, the first imposes a minimality condition: the lottery is always won by a person succeeding the test, if there exists one. On the other hand, in C-LOG the two rules are independent, and models might not be minimal. In this example, it is the latter that is intended. This illustrates modularity of C-LOG. The rule $(\mathbf{Select}\, x[\mathbf{t}] : permres(x)) \leftarrow lottery$ means that one person is selected randomly to obtain residence. Adding other rules does not change the meaning of this rule; causal effects do not interfere.

Object-creation in C-LOG is discussed in the next section.

5 Object-creation in C-LOG

Object creation is available in C-LOG through the **New**-operator. Like every language construct in C-LOG, the informal interpretation of an expression

$$\mathbf{New}\, x : P(x) \leftarrow \varphi$$

is defined in terms of causal processes. The above expression states that φ causes the creation of a new element and that for that new element, P is caused. Object-creation is also subject to the principles of sufficient causation, universal causation and no self-causation. In order to apply these principles, the domain of a structure is partitioned into two parts: the *initial* elements are those whose existence is not governed by the causal theory, they are exogenous and the *created* elements are those created by expressions in the causal theory, i.e., they are endogenous. For created elements, their default value is *not existing* and their deviant value is existing. Thus, at the start of a causal process, only the initial elements exist, as soon as the preconditions of a **New**-expressions are satisfied, an element is added to the domain. The principle of no self-causation takes these default and deviant values into account: an object cannot be created based on its own existence. Consider for example the following causal theory:

$$\begin{aligned}&\mathbf{Select}\, x[\mathbf{t}] : P(x)\\ &(\mathbf{New}\, y : Q(y)) \leftarrow \exists x : P(x)\\ &\mathbf{Select}\, x[\mathbf{t}] : R(x)\end{aligned}$$

The first and last expressions select one object randomly and cause P (respectively R) to hold for that object. The second expression creates a new element conditionally, only if there is at least one element satisfying P. In this example, the element selected for the first expression cannot be the one created in the second. **Select**-operators can only select existing elements and the object created in the second expression can only be created after the selection in the first rule, after there is some object satisfying P. For the last expression, any element can be selected. Hence, this causal theory has no models with only one domain element. A structure $\mathcal{I}$ with domain $\{A, B\}$ and with $P^{\mathcal{I}} = \{A\}$ and $Q^{\mathcal{I}} = R^{\mathcal{I}} = \{B\}$ is a model of the above causal theory. In this case, B is the unique created element, and A is initial, i.e., A is assumed to exist before the described causal process takes place. This illustrates that the **New**-operator is more than simply a **Select** together with unique name axioms: its semantics is really integrated in the underlying causal process. The behaviour of **New**-expressions can be simulated using **Select**-expressions if we make the two parts of the domain (initial and created elements) explicit and conditionalise all quantifications. A detailed discussion of this transformation is out of the scope of this paper.

Object creation occurs in many fields, of which we discuss some below.

5.1 Object-Creation in Database Systems

Object-creation has been studied intensively in the field of deductive databases. In (Abiteboul and Vianu 1991), various extensions of Datalog, are considered, resulting in nondeterministic semantics for queries and updates. One of the studied extensions is object creation (throught existential quantifications in rule heads). These and similar related extension have been implemented in several systems, including LogicBlox (Green, Aref, and Karvounarakis 2012). An example from the latter paper is the rule:

$$President(p), presidentOf[c] = p \leftarrow Country(c).$$

which means that for every country c, a new (anonymous) "derived entity" of type $President$ is created. Of course, the president of a country is not a new person, but the president is new with respect to the database, which does not contain any persons yet. Such rules with (implicit) existentially quantified head variables correspond to **New**-expressions. Here, it would translate to

$$\mathbf{All}\, c[Country(c)] : \mathbf{New}\, p : Pres(p)\, \mathbf{And}\, presOf(c, p).$$

This shows that in some rule-based paradigms, an existentially quantified head-variable corresponds to object-creation (**New**), while in other rule-based paradigms, such as ASP, we saw that an existentially quantified head variable corresponds to a selection. The relation between these paradigms has, to the best of our knowledge, not yet been studied thoroughly. We believe that FO(C), which makes an explicit distinction between selection and object-creation, is an interesting tool to study this relationship. This is future work.

Many other Datalog extensions with forms of object creation exist. For example (Van den Bussche and Paredaens 1995) discusses a version with creation of sets and compares its expressivity with simple object creation.

Object-creation also occurs in other database languages, such as for example the query language $\text{while}_{\text{new}}$ in (Abiteboul, Hull, and Vianu 1995). An expression

```
while R do (P = new Q)
```

in that language corresponds to a CEE.

$$\mathbf{All}\, t[R(t)] : \mathbf{All}\, \overline{x}[\mathbf{t}] : \mathbf{New}\, y : P(\overline{x}, y, t+1) \leftarrow Q(\overline{x}, t).$$

In fact in (Bogaerts et al. in press 2014b), it has been shown that C-LOG can "simulate" the entire language $\text{while}_{\text{new}}$.

5.2 Object-Creation in Mathematics

Object-creation also occurs in mathematics. The set of all natural numbers can be thought of as the set obtained by a process that first creates one element (zero) and for every element in this set, adds another element (its successor). In

C-LOG, the above natural language sentences can be modelled as follows

$$\textbf{New}\, x : (Nat(x)\, \textbf{And}\, Zero(x))$$
$$\textbf{All}\, x[Nat(x)] : \textbf{New}\, y : (Nat(y)\, \textbf{And}\, Succ(x, y)).$$

Models of the above theory are exactly those structures interpreting $Nat, Zero, Succ$ as the natural numbers, zero and the successor function (modulo isomorphism).

5.3 Object-Creation in Business Rules Systems

Business Rules (Business Rules Group 2000) engines are widely used in the industry. One big drawback of these systems is their inability to perform multiple forms of reasoning. For example, banks might use a Business Rules engine to decide whether someone is eligible for a loan. This approach can be very efficient, but as soon as one is not only interested in the above question, but also in explanations, or suggestions about what to change in order to become eligible, the application should be redesigned. Previous attempts to translate Business Rules applications into a logic with a Tarskian model semantics have been made in (Hertum et al. 2013). The conclusion of this study was that for such a transformation, we need object creation . We believe that C-LOG provides a suitable form of object-creation for this purpose. As an illustration, the JBoss manual (Browne 2009) contains the following rule:

```
when Order( customer == null )
then insertLogical(new
    ValidationResult(
   validation.customer.missing ));
```

This rule means that if an order is created without customer, a new *ValidationResult* is created with the message that the customer is missing. This can be translated to C-LOG as follows:

$$\textbf{All}\, y[Order(y) \wedge NoCustumer(y)] :$$
$$\textbf{New}\, x : ValidationR(x)\, \textbf{And}\, Message(x, \text{“...”}).$$

A more thorough study of the relationship between the operational semantics of Business Rules systems and the semantics of C-LOG is a topic for future work.

6 Conclusion

In this paper we compared FO(C) to other modelling paradigms. We discussed the semantical relationship between C-LOG and FO. We identified a fragment of E-disjunctive logic programs for which the stable model semantics corresponds to the semantics of FO(C), and argued how FO(C) enriches such programs with several useful modelling constructs. Furthermore, we argued that the object-creation in FO(C) corresponds to the object creation in many related language. Besides technical relationship between these languages, we believe that this discussion also provides insights in the semantics of FO(C).

References

Abiteboul, S., and Vianu, V. 1991. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.* 43(1):62–124.

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Bogaerts, B.; Vennekens, J.; Denecker, M.; and Van den Bussche, J. (in press) 2014a. FO(C): A knowledge representation language of causality. *Theory and Practice of Logic Programming (TPLP)* (Online-Supplement, Technical Communication ICLP14).

Bogaerts, B.; Vennekens, J.; Denecker, M.; and Van den Bussche, J. (in press) 2014b. Inference in the FO(C) modelling language. In *ECAI 2014 - 21th European Conference on Artificial Intelligence, Prague, Czech Republic, August 18-22, 2014, Proceedings*.

Browne, P. 2009. *JBoss Drools Business Rules*. From technologies to solutions. Packt Publishing, Limited.

Business Rules Group. 2000. Defining Business Rules $\sim$ What Are They Really? Technical report.

Denecker, M.; Lierler, Y.; Truszczynsky, M.; and Vennekens, J. 2012. A Tarskian informal semantics for answer set programming. In Dovier, A., and Santos Costa, V., eds., *Technical Communications of the 28th International Conference on Logic Programming*, 277–289. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Green, T. J.; Aref, M.; and Karvounarakis, G. 2012. Logicblox, platform and language: A tutorial. In Barceló, P., and Pichler, R., eds., *Datalog*, volume 7494 of *LNCS*, 1–8. Springer.

Hall, N. 2004. Two concepts of causation. In *Causation and Counterfactuals*.

Hertum, P. V.; Vennekens, J.; Bogaerts, B.; Devriendt, J.; and Denecker, M. 2013. The effects of buying a new car: an extension of the IDP knowledge base system. *TPLP* 13(4-5-Online-Supplement).

Hitchcock, C. 2007. Prevention, preemption, and the principle of sufficient reason. *Philosophical review* 116(4).

Kowalski, R. A., and Sadri, F. 2013. Towards a logic-based unifying framework for computing. *CoRR* abs/1301.6905.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Preyer, G., and Peter, G. 2002. *Logical Form and Language*. Clarendon Press.

Van den Bussche, J., and Paredaens, J. 1995. The expressive power of complex values in object-based data models. *Information and Computation* 120:220–236.

Vennekens, J.; Denecker, M.; and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming* 9(3):245–308.

You, J.-H.; Zhang, H.; and Zhang, Y. 2013. Disjunctive logic programs with existential quantification in rule heads. *Theory and Practice of Logic Programming* 13:563–578.

Belief Merging within Fragments of Propositional Logic

Nadia Creignou and **Odile Papini**
Aix Marseille Université, CNRS

Stefan Rümmele and **Stefan Woltran**
Vienna University of Technology

Abstract

Recently, belief change within the framework of fragments of propositional logic has gained increasing attention. Previous works focused on belief contraction and belief revision on the Horn fragment. However, the problem of belief merging within fragments of propositional logic has been neglected so far. This paper presents a general approach to define new merging operators derived from existing ones such that the result of merging remains in the fragment under consideration. Our approach is not limited to the case of Horn fragment but applicable to any fragment of propositional logic characterized by a closure property on the sets of models of its formulæ. We study the logical properties of the proposed operators in terms of satisfaction of merging postulates, considering in particular distance-based merging operators for Horn and Krom fragments.

Introduction

Belief merging consists in achieving a synthesis between pieces of information provided by different sources. Although these sources are individually consistent, they may mutually conflict. The aim of merging is to provide a consistent set of information, making maximum use of the information provided by the sources while not favoring any of them. Belief merging is an important issue in many fields of Artificial Intelligence (AI) (Bloch and (Eds) 2001) and symbolic approaches to multi-source fusion gave rise to increasing interest within the AI community since the 1990s (Baral, Kraus, and Minker 1991; Cholvy 1998; Lin 1996; Revesz 1993; 1997). One of today's major approaches is the problem of merging under (integrity) constraints in order to generalize both merging (without constraints) and revision (of old information by a new piece of information). For the latter the constraints then play the role of the new piece of information. Postulates characterizing the rational behavior of such merging operators, known as IC postulates, have been proposed by Konieczny and Pino Pérez (Konieczny and Pino Pérez 2002) in the same spirit as the seminal AGM (Alchourrón, Gärdenfors, and Makinson 1985) postulates for revision. Concrete merging operators have been proposed according to either semantic (model-based) or syntactic (formula-based) points of view in a classical logic setting (Chacón and Pino Pérez 2012). We focus here on the model-based approach of distance-based merging operators (Konieczny, Lang, and Marquis 2004; Konieczny and Pino Pérez 2002; Revesz 1997). These operators are parametrized by a distance which represents the closeness between interpretations and an aggregation function which captures the merging strategy and takes the origin of beliefs into account.

Belief change operations within the framework of fragments of classical logic constitute a vivid research branch. In particular, contraction (Booth et al. 2011; Delgrande and Wassermann 2013; Zhuang and Pagnucco 2012) and revision (Delgrande and Peppas 2011; Putte 2013; Zhuang, Pagnucco, and Zhang 2013) have been thoroughly analyzed in the literature. The study of belief change within language fragments is motivated by two central observations:

- In many applications, the language is restricted a priori. For instance, a rule-based formalization of expert's knowledge is much easier to handle for standard users. In case users want to revise or merge some sets of rules, they indeed expect that the outcome is still in the easy-to-read format they are used to.
- Many fragments of propositional logic allow for efficient reasoning methods. Suppose an agent has to make a decision according to a group of experts' beliefs. This should be done efficiently, therefore the expert's beliefs are stored as formulæ known to be in a tractable class. For making a decision, it is desired that the result of the change operation yields a set of formulæ in the same fragment. Hence, the agent still can use the dedicated solving method she is equipped with for this fragment.

Most of previous work has focused on the Horn fragment except (Creignou et al. 2014) that studied revision in any fragment of propositional logic. However, as far as we know, the problem of *belief merging* within *fragments of propositional logic* has been neglected so far.

The main obstacle hereby is that for a language fragment $\mathcal{L}'$, given n belief bases $K_1, \ldots, K_n \in 2^{\mathcal{L}'}$ and a constraint $\mu \in \mathcal{L}'$, there is no guarantee that the outcome of the merging, $\Delta_\mu(\{K_1, \ldots, K_n\})$, remains in $\mathcal{L}'$ as well. Let for example, $K_1 = \{a\}$, $K_2 = \{b\}$ and $\mu = \neg a \vee \neg b$ be two sets of formulæ and a formula expressed in the Horn fragment. Merging with typical distance-based operator proposed in (Konieczny and Pino Pérez 2002) does not remain in the Horn language fragment since the result of merging is equiv-

alent to $(a \vee b) \wedge (\neg a \vee \neg b)$, which is not equivalent to any Horn formula (see (Schaefer 1978)).

We propose the concept of *refinement* to overcome these problems. Refinements have been proposed for revision in (Creignou et al. 2014) and capture the intuition of adapting a given operator (defined for full classical logic) in order to become applicable within a fragment. The basic properties of a refinement aim to (i) guarantee the result of the change operation to be in the same fragment as the belief change scenario given and (ii) keep the behavior of the original operator unchanged in case it delivers a result which already fits in the fragment.

Refinements are interesting from different points of view. Several fragments can be treated in a uniform way and a general characterization of refinements is provided for any fragment. Defining and studying refinements of merging operators is not a straightforward extension of the revision case. It is more complex due to the nature of the merging operators. Even if the constraints play the role of the new piece of information in revision, model-based merging deals with multi-sets of models. Moreover applying this approach to different distance-based merging operators, each parameterized by a distance and an aggregation function, reveals that all the different parameters matter, thus showing a rich variety of behaviors for refined merging operators.

The main contributions of this paper are the following:

- We propose to adapt known belief merging operators to make them applicable in fragments of propositional logic. We provide natural criteria, which refined operators should satisfy. We characterize refined operators in a constructive way.
- This characterization allows us to study their properties in terms of the IC postulates (Konieczny and Pino Pérez 2002). On one hand we prove that the basic postulates (IC0–IC3) are preserved for any refinement for any fragment. On the other hand we show that the situation is more complex for the remaining postulates. We provide detailed results for the Horn and the Krom fragment in terms of two kinds of distance-based merging operators and three approaches for refinements.

Preliminaries

Propositional Logic. We consider $\mathcal{L}$ as the language of propositional logic over some fixed alphabet $\mathcal{U}$ of propositional atoms. A literal is an atom or its negation. A clause is a disjunction of literals. A clause is called *Horn* if at most one of its literals is positive; and *Krom* if it consists of at most two literals. We identify the following subsets of $\mathcal{L}$: $\mathcal{L}_{Horn}$ is the set of all formulæ in $\mathcal{L}$ being conjunctions of Horn clauses, and $\mathcal{L}_{Krom}$ is the set of all formulæ in $\mathcal{L}$ being conjunctions of Krom clauses. In what follows we sometimes just talk about arbitrary fragments $\mathcal{L}' \subseteq \mathcal{L}$. Hereby, we tacitly assume that any such fragment $\mathcal{L}' \subseteq \mathcal{L}$ contains at least the formula $\top$.

An interpretation is represented either by a set $\omega \subseteq \mathcal{U}$ of atoms (corresponding to the variables set to true) or by its corresponding characteristic bit-vector of length $|\mathcal{U}|$. For instance if we consider $\mathcal{U} = \{x_1, \ldots, x_6\}$, the interpretation $x_1 = x_3 = x_6 = 1$ and $x_2 = x_4 = x_5 = 0$ will be represented either by $\{x_1, x_3, x_6\}$ or by $(1, 0, 1, 0, 0, 1)$. As usual, if an interpretation ω satisfies a formula ϕ, we call ω a model of ϕ. By $\mathrm{Mod}(\phi)$ we denote the set of all models (over $\mathcal{U}$) of ϕ. Moreover, $\psi \models \phi$ if $\mathrm{Mod}(\psi) \subseteq \mathrm{Mod}(\phi)$ and $\psi \equiv \phi$ (ϕ and ψ are equivalent) if $\mathrm{Mod}(\psi) = \mathrm{Mod}(\phi)$.

A *base* K is a finite set of propositional formulæ $\{\varphi_1, \ldots, \varphi_n\}$. We shall often identify K via $\bigwedge K$, the conjunction of formulæ of K, i.e., $\bigwedge K = \varphi_1 \wedge \cdots \wedge \varphi_n$. Thus, a base K is said to be consistent if $\bigwedge K$ is consistent, $\mathrm{Mod}(K)$ is a shortcut for $\mathrm{Mod}(\bigwedge K)$, $K \models \phi$ stands for $\bigwedge K \models \phi$, etc. Given $\mathcal{L}' \subseteq \mathcal{L}$ we denote by $\mathcal{K}_{\mathcal{L}'}$ the set of bases restricted to formulæ from $\mathcal{L}'$. For fragments $\mathcal{L}' \subseteq \mathcal{L}$, we also use $T_{\mathcal{L}'}(K) = \{\phi \in \mathcal{L}' \mid K \models \phi\}$.

A *profile* E is a non-empty finite multiset of consistent bases $E = \{K_1, \ldots, K_n\}$ and represents a group of n agents having different beliefs. Given $\mathcal{L}' \subseteq \mathcal{L}$, we denote by $\mathcal{E}_{\mathcal{L}'}$ the set of profiles restricted to the use of formulæ from $\mathcal{L}'$. We denote $\bigwedge K_1 \wedge \ldots \wedge \bigwedge K_n$ by $\bigwedge E$. The profile is said to be consistent if $\bigwedge E$ is consistent. By abuse of notation we write $K \sqcup E$ to denote the multi-set union $\{K\} \sqcup E$. The multi-set consisting of the sets of models of the bases in a profile is denoted $\mathcal{M}od(E) = \{\mathrm{Mod}(K_1), \ldots, \mathrm{Mod}(K_n)\}$. Two profiles E_1 and E_2 are equivalent, denoted by $E_1 \equiv E_2$ if $\mathcal{M}od(E_1) = \mathcal{M}od(E_2)$. Finally, for a set of interpretations $\mathcal{M}$ and a profile E we define $\#(\mathcal{M}, E) = |\{i : \mathcal{M} \cap \mathrm{Mod}(K_i) \neq \emptyset\}|$.

Characterizable Fragments of Propositional Logic. Let $\mathcal{B}$ denote the set of all Boolean functions $\beta\colon \{0,1\}^k \to \{0,1\}$ that have the following two properties[1]:

- *symmetry*, i.e., for all permutations σ, $\beta(x_1, \ldots, x_k) = \beta(x_{\sigma(1)}, \ldots, x_{\sigma(k)})$ and
- 0- and 1-*reproduction*, i.e., for all $x \in \{0,1\}$, $\beta(x, \ldots, x) = x$.

Examples are the binary AND function denoted by $\wedge$ or the ternary MAJORITY function, $\mathrm{maj}_3(x, y, z) = 1$ if at least two of the variables x, y, and z are set to 1. We extend Boolean functions to interpretations by applying coordinate-wise the original function (recall that we consider interpretations also as bit-vectors). So, if $M_1, \ldots, M_k \in \{0,1\}^n$, then $\beta(M_1, \ldots, M_k)$ is defined by $(\beta(M_1[1], \ldots, M_k[1]), \ldots, \beta(M_1[n], \ldots, M_k[n]))$, where $M[i]$ is the i-th coordinate of the interpretation M.

Definition 1. *Given a set $\mathcal{M} \subseteq 2^{\mathcal{U}}$ of interpretations and $\beta \in \mathcal{B}$, we define $Cl_\beta(\mathcal{M})$, the* closure *of $\mathcal{M}$ under β, as the smallest set of interpretations that contains $\mathcal{M}$ and that is closed under β, i.e., if $M_1, \ldots, M_k \in Cl_\beta(\mathcal{M})$, then also $\beta(M_1, \ldots, M_k) \in Cl_\beta(\mathcal{M})$.*

Let us mention some easy properties of such a closure: (i) monotonicity; (ii) if $|\mathcal{M}| = 1$, then $Cl_\beta(\mathcal{M}) = \mathcal{M}$; (iii) $Cl_\beta(\emptyset) = \emptyset$.

Definition 2. *Let $\beta \in \mathcal{B}$. A set $\mathcal{L}' \subseteq \mathcal{L}$ of propositional formulæ is a* β-fragment *(or* characterizable fragment*) if:*

[1] these properties are also known as anonimity and unanimity.

1. *for all* $\psi \in \mathcal{L}'$, $\mathrm{Mod}(\psi) = Cl_\beta(\mathrm{Mod}(\psi))$
2. *for all* $\mathcal{M} \subseteq 2^{\mathcal{U}}$ *with* $\mathcal{M} = Cl_\beta(\mathcal{M})$ *there exists a* $\psi \in \mathcal{L}'$ *with* $\mathrm{Mod}(\psi) = \mathcal{M}$
3. *if* $\phi, \psi \in \mathcal{L}'$ *then* $\phi \wedge \psi \in \mathcal{L}'$.

It is well-known that $\mathcal{L}_{Horn}$ is an $\wedge$-fragment and $\mathcal{L}_{Krom}$ is a maj_3-fragment (see e.g. (Schaefer 1978)).

Logical Merging Operators. Belief merging aims at combining several pieces of information coming from different sources. Merging operators we consider are functions from the set of profiles and the set of propositional formulæ to the set of bases, i.e., $\Delta\colon \mathcal{E}_\mathcal{L} \times \mathcal{L} \to \mathcal{K}_\mathcal{L}$. For $E \in \mathcal{E}_\mathcal{L}$ and $\mu \in \mathcal{L}$ we will write $\Delta_\mu(E)$ instead of $\Delta(E, \mu)$; the formula μ is referred to as the *integrity constraint* (IC) and restricts the result of the merging.

As for belief revision some logical properties that one could expect from any reasonable merging operator have been stated. See (Konieczny and Pino Pérez 2002) for a detailed discussion. Intuitively $\Delta_\mu(E)$ is the "closest" belief base to the profile E satisfying the integrity constraint μ. This is what the following postulates try to capture.

(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 $\Delta_\mu(E) = \bigwedge E \wedge \mu$
(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 if and only if $\Delta_\mu(\{K_1, K_2\}) \wedge K_2$ 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 $\Delta_{\mu_1 \wedge \mu_2}(E) \models \Delta_{\mu_1}(E)$

Similarly to belief revision, a representation theorem (Konieczny and Pino Pérez 2002) shows that a merging operator corresponds to a family of total preorders over interpretations. More formally, for $E \in \mathcal{E}_\mathcal{L}$, $\mu \in \mathcal{L}$ and $\leq_E$ a total preorder over interpretations, a model-based operator is defined by $\mathrm{Mod}(\Delta_\mu(E)) = min(\mathrm{Mod}(\mu), \leq_E)$. The model-based merging operators select interpretations that are the "closest" to the original belief bases.

Distance-based operators where the notion of closeness stems from the definition of a distance (or a pseudo-distance[2]) between interpretations and from an aggregation function have been proposed in (Konieczny and Pino Pérez 2002; 2011). An aggregation function f is a function mapping for any positive integer n each n-tuple of positive reals into a positive real such that for any $x_1, \ldots, x_n, x, y \in R^+$, if $x \leq y$, then $f(x_1, \ldots, x, \ldots, x_n) \leq f(x_1, \ldots, y, \ldots, x_n)$, $f(x_1, \ldots, x_n) = 0$ if and only if $x_1 = \ldots = x_n = 0$ and $f(x) = x$. Let $E = \{K_1, \ldots, K_n\} \in \mathcal{E}_\mathcal{L}$, $\mu \in \mathcal{L}$, d be a distance and f be an aggregation function, we consider the family of $\Delta_\mu^{d,f}$ merging operators defined by $\mathrm{Mod}(\Delta_\mu^{d,f}(E)) = min(\mathrm{Mod}(\mu), \leq_E)$ where $\leq_E$ is a total preorder over the set $2^{\mathcal{U}}$ of interpretations defined as follows:

- $d(\omega, K_i) = min_{\omega' \models K_i} d(\omega, \omega')$,
- $d(\omega, E) = f(d(\omega, K_1), \ldots, d(\omega, K_n))$, and
- $\omega \leq_E \omega'$ if $d(\omega, E) \leq d(\omega', E)$.

Definition 3. *A* counting distance *between interpretations is a function* $d : 2^{\mathcal{U}} \times 2^{\mathcal{U}} \to \mathbb{R}^+$ *defined for every pair of interpretations* (ω, ω') *by* $d(\omega, \omega') = g(|(\omega \setminus \omega') \cup (\omega' \setminus \omega)|)$, *where* $g : \mathbb{N} \to \mathbb{R}^+$ *is a nondecreasing function such that* $g(n) = 0$ *if and only if* $n = 0$. *If* $g(n) = g(1)$ *for every* $n \neq 0$, *we call* d *a* drastic distance *and denote it via* d_D. *If* $g(n) = n$ *for all* n, *we call* d *the* Hamming distance *and denote it via* d_H. *If for every interpretations* w, w' *and* w'' *we have* $d(w, w') \leq d(w, w'') + d(w'', w')$, *then we say that the distance* d *satisfies the triangular inequality.*

Observe that a counting distance is indeed a pseudo-distance, and both, the Hamming distance and drastic distance satisfy the triangular inequality.

As aggregation functions, we consider here Σ, the sum aggregation function, and the aggregation function GMax defined as follows. Let $E = \{K_1, \ldots, K_n\} \in \mathcal{E}_\mathcal{L}$ and ω, ω' be two interpretations. Let $(d_1^\omega, \ldots, d_n^\omega)$, where $d_j^\omega = d_H(\omega, K_j)$, be the vector of distances between ω and the n belief bases in E. Let L_ω^E be the vector obtained from $(d_1^\omega, \ldots, d_n^\omega)$ by ranking it in decreasing order. The aggregation function GMax is defined by $\mathrm{GMax}(d_1^\omega, \ldots, d_n^\omega) = L_\omega^E$, with $\mathrm{GMax}(d_1^\omega, \ldots, d_n^\omega) \leq \mathrm{GMax}(d_1^{\omega'}, \ldots, d_n^{\omega'})$ if $L_\omega^E \leq_{lex} L_{\omega'}^E$, where $\leq_{lex}$ denotes the lexicographical ordering.

In this paper we focus on the $\Delta^{d,\Sigma}$ and $\Delta^{d,\mathrm{GMax}}$ operators where d is an arbitrary counting distance. These operators are known to satisfy the postulates (IC0)–(IC8), as shown in (Konieczny, Lang, and Marquis 2004) generalizing more specific results from (Konieczny and Pino Pérez 2002; Lin and Mendelzon 1998). Finally, we define certain concepts for merging operators and fragments.

Definition 4. *A* basic *(merging) operator for* $\mathcal{L}' \subseteq \mathcal{L}$ *is any function* $\Delta : \mathcal{E}_{\mathcal{L}'} \times \mathcal{L}' \to \mathcal{K}_{\mathcal{L}'}$ *satisfying* $\mathrm{Mod}(\Delta_\mu(\{\{\top\}\})) = \mathrm{Mod}(\mu)$ *for each* $\mu \in \mathcal{L}'$. *We say that* Δ satisfies *an* (IC) *postulate* (IC_i) *(*$i \in \{0, \ldots, 8\}$*)* in $\mathcal{L}'$ *if the respective postulate holds when restricted to formulæ from* $\mathcal{L}'$.

Refined Operators

Let us consider a simple example to illustrate the problem of standard operators when applied within a fragment of propositional logic.

Example 1. *Let* $\mathcal{U} = \{a, b\}$, $E = \{K_1, K_2\} \in \mathcal{E}_{\mathcal{L}_{Horn}}$ *and* $\mu \in \mathcal{L}_{Horn}$ *such that* $\mathrm{Mod}(K_1) = \{\{a\}, \{a, b\}\}$, $\mathrm{Mod}(K_2) = \{\{b\}, \{a, b\}\}$, *and* $\mathrm{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}\}$. *Consider the distance-based merging operators,* $\Delta^{d_H,\Sigma}$ *and* $\Delta^{d_H,\mathrm{GMax}}$. *The following table gives the distances between*

[2] Let $\omega, \omega' \in \mathcal{W}$, a pseudo-distance is such that $d(\omega, \omega') = d(\omega', \omega)$ and $d(\omega, \omega') = 0$ if and only if $\omega = \omega'$.

the interpretations of μ and the belief bases, and the result of the aggregation functions Σ and GMax.

$2^{\mathcal{U}}$	K_1	K_2	Σ	GMax
$\emptyset$	1	1	2	$(1,1)$
$\{a\}$	0	1	1	$(1,0)$
$\{b\}$	1	0	1	$(1,0)$

Hence, we have $\mathrm{Mod}(\Delta_\mu^{d_H,\Sigma}(E)) = \mathrm{Mod}(\Delta_\mu^{d_H,\mathrm{GMax}}(E)) = \{\{a\},\{b\}\}$. *Thus, for instance, we can give $\phi = (a \vee b) \wedge (\neg a \vee \neg b)$ as a result of the merging for both operators. However, there is no $\psi \in \mathcal{L}_{Horn}$ with $\mathrm{Mod}(\psi) = \{\{a\},\{b\}\}$ (each $\psi \in \mathcal{L}_{Horn}$ satisfies the following closure property in terms of its set of models: for every $I, J \in \mathrm{Mod}(\psi)$, also $I \cap J \in \mathrm{Mod}(\psi)$)). Thus, the result of the operator has to be "refined", such that it fits into the Horn fragment. On the other hand, it holds that $\mu \in \mathcal{L}_{Krom}$, $E \in \mathcal{E}_{\mathcal{L}_{Krom}}$ and also the result ϕ is in Krom. This shows that different fragments behave differently on certain instances. Nonetheless, we aim for a uniform approach for refining merging operators.*

We are interested in the following: Given a known merging operator Δ and a fragment $\mathcal{L}'$ of propositional logic, how can we adapt Δ to a new merging operator $\Delta^\star$ such that, for each $E \in \mathcal{E}_{\mathcal{L}'}$ and $\mu \in \mathcal{L}'$, $\Delta^\star_\mu(E) \in \mathcal{K}_{\mathcal{L}'}$? Let us define a few natural desiderata for $\Delta^\star$ inspired by the work on belief revision. See (Creignou et al. 2014) for a discussion.

Definition 5. *Let $\mathcal{L}'$ be a fragment of classical logic and Δ a merging operator. We call an operator $\Delta^\star : \mathcal{E}_{\mathcal{L}'} \times \mathcal{L}' \to \mathcal{K}_{\mathcal{L}'}$ a* Δ-refinement for $\mathcal{L}'$ *if it satisfies the following properties, for each $E, E_1, E_2 \in \mathcal{E}_{\mathcal{L}'}$ and $\mu, \mu_1, \mu_2 \in \mathcal{L}'$.*

1. *consistency: $\Delta_\mu(E)$ is consistent if and only if $\Delta^\star_\mu(E)$ is consistent*
2. *equivalence: if $E_1 \equiv E_2$ and $\Delta_{\mu_1}(E_1) \equiv \Delta_{\mu_2}(E_2)$ then $\Delta^\star_{\mu_1}(E_1) \equiv \Delta^\star_{\mu_2}(E_2)$*
3. *containment: $T_{\mathcal{L}'}(\Delta_\mu(E)) \subseteq T_{\mathcal{L}'}(\Delta^\star_\mu(E))$*
4. *invariance: If $\Delta_\mu(E) \in \mathcal{K}_{\langle\mathcal{L}'\rangle}$, then $T_{\mathcal{L}'}(\Delta^\star_\mu(E)) \subseteq T_{\mathcal{L}'}(\Delta_\mu(E))$, where $\langle\mathcal{L}'\rangle$ denotes the set of formulæ in $\mathcal{L}$ for which there exists an equivalent formula in $\mathcal{L}'$.*

Next we introduce examples of refinements that fit Definition 5.

Definition 6. *Let Δ be a merging operator and $\beta \in \mathcal{B}$. We define the Cl_β-*based *refined operator Δ^{Cl_β} as:*

$$\mathrm{Mod}(\Delta_\mu^{Cl_\beta}(E)) = Cl_\beta(\mathcal{M}).$$

where $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E))$.

We define the Min-based *refined operator Δ^{Min} as:*

$$\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E)) = \begin{cases} \mathcal{M} & \text{if } Cl_\beta(\mathcal{M}) = \mathcal{M}, \\ \{\mathrm{Min}(\mathcal{M})\} & \text{otherwise}, \end{cases}$$

where Min *is a function that selects the minimum from a set of interpretations with respect to a given and fixed order.*

We define the Min/Cl_β-based *refined operator $\Delta^{\mathrm{Min}/Cl_\beta}$ as:*

$$\Delta_\mu^{\mathrm{Min}/Cl_\beta}(E) = \begin{cases} \Delta_\mu^{\mathrm{Min}}(E) & \text{if } \#(\mathcal{M}, E) = 0 \\ \Delta_\mu^{Cl_\beta}(E) & \text{otherwise}. \end{cases}$$

The intuition behind the last refinement is to ensure a certain form of fairness, i.e. if no model is selected from the profile, this carries over to the refinement.

Proposition 1. *For any merging operator $\Delta : \mathcal{E}_\mathcal{L} \times \mathcal{L} \to \mathcal{K}_\mathcal{L}$, $\beta \in \mathcal{B}$ and $\mathcal{L}' \subseteq \mathcal{L}$ a β-fragment, the operators Δ^{Cl_β}, Δ^{Min} and $\Delta_\mu^{\mathrm{Min}/Cl_\beta}$ are Δ-refinements for $\mathcal{L}'$.*

Proof. Let $\mu \in \mathcal{L}'$, $E \in \mathcal{E}_{\mathcal{L}'}$ and $\beta \in \mathcal{B}$. We show that each operator yields a base from $\mathcal{K}_{\mathcal{L}'}$ and moreover satisfies consistency, equivalence, containment and invariance, cf. Definition 5.

Δ^{Cl_β}: $\Delta_\mu^{Cl_\beta}(E) \in \mathcal{L}'$ since by assumption $\mathcal{L}'$ is a β-fragment and thus closed under β. Consistency holds since $\mathrm{Mod}(\Delta_\mu^{Cl_\beta}(E)) = Cl_\beta(\mathrm{Mod}(\Delta_\mu(E)))$ and $Cl_\beta(\mathcal{M}) = \emptyset$ iff $\mathcal{M} = \emptyset$. Equivalence holds since $\mathrm{Mod}(\Delta_{\mu_1}(E_1)) = \mathrm{Mod}(\Delta_{\mu_2}(E_2))$ implies $Cl_\beta(\mathrm{Mod}(\Delta_{\mu_1}(E_1))) = Cl_\beta(\mathrm{Mod}(\Delta_{\mu_2}(E_2)))$. Containment: let $\phi \in T_{\mathcal{L}'}(\Delta_\mu(E))$, i.e. $\phi \in \mathcal{L}'$ and $\mathrm{Mod}(\Delta_\mu(E)) \subseteq \mathrm{Mod}(\phi)$. By monotonicity of Cl_β, then $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq Cl_\beta(\mathrm{Mod}(\phi))$. Since $\phi \in \mathcal{L}'$ then $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq \mathrm{Mod}(\phi)$ therefore $\phi \in T_{\mathcal{L}'}(\Delta_\mu^{Cl_\beta}(E))$. Invariance: let $\phi \in T_{\mathcal{L}'}(\Delta_\mu^{Cl_\beta}(E))$, i.e. $\phi \in \mathcal{L}'$ and $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq \mathrm{Mod}(\phi)$. By hypothesis $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \supseteq \mathrm{Mod}(\Delta_\mu(E))$, therefore $\phi \in T_{\mathcal{L}'}(\Delta_\mu(E))$.

Δ^{Min}: if $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E))) = Cl_\beta(\mathrm{Mod}(\Delta_\mu(E)))$ (i.e. $\Delta_\mu(E) \in \mathcal{K}_{\langle\mathcal{L}'\rangle}$) then Δ^{Min} satisfies all the required properties as shown above; otherwise consistency, equivalence and containment hold since $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E))) = \{\mathrm{Min}(\mathrm{Mod}(\Delta_\mu(E)))\}$. Moreover, by definition each fragment contains a formula ϕ with $\mathrm{Mod}(\phi) = \{\omega\}$ where ω is an arbitrary interpretation. $\Delta_\mu(E) \in \mathcal{L}'$ thus also holds in this case.

$\Delta^{\mathrm{Min}/Cl_\beta}$: satisfies the required properties since Δ^{Cl_β} and Δ^{Min} satisfy them. □

Example 2. *Consider the profile E, the integrity constraint μ given in Example 1, the distance-based merging operator $\Delta^{d_H,\Sigma}$, and let β be the binary AND function. Let us have the following order over the set of interpretations on $\{a,b\}$: $\emptyset < \{a\} < \{b\} < \{a,b\}$. The result of merging is* $\mathrm{Mod}(\Delta_\mu^{d_H,\Sigma}(E)) = \{\{a\},\{b\}\}$. *The* Min*-based $\Delta^{d_H,\Sigma}$-refined operator, denoted by Δ^{Min}, is such that* $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E)) = \{\{a\}\}$. *The Cl_β-based $\Delta^{d_H,\Sigma}$-refined operator, denoted by Δ^{Cl_β}, is such that* $\mathrm{Mod}(\Delta_\mu^{Cl_\beta}(E)) = \{\{a\},\{b\},\emptyset\}$. *The same result is achieved by the* Min/Cl_β*-based $\Delta^{d_H,\Sigma}$-refined operator since $\#(\mathrm{Mod}(\Delta_\mu^{d_H,\Sigma}(E)), E) = 2$.*

In what follows we show how to capture not only a particular refined operator but characterize the class of *all* refined operators.

Definition 7. *Given $\beta \in \mathcal{B}$, we define a* β-mapping, *f_β, as an application which to every set of models $\mathcal{M}$ and every multi-set of sets of models $\mathcal{X}$ associates a set of models $f_\beta(\mathcal{M}, \mathcal{X})$ such that:*

1. *$Cl_\beta(f_\beta(\mathcal{M}, \mathcal{X})) = f_\beta(\mathcal{M}, \mathcal{X})$ ($f_\beta(\mathcal{M}, \mathcal{X})$ is closed under β)*

2. $f_\beta(\mathcal{M}, \mathcal{X}) \subseteq Cl_\beta(\mathcal{M})$
3. *if* $\mathcal{M} = Cl_\beta(\mathcal{M})$*, then* $f_\beta(\mathcal{M}, \mathcal{X}) = \mathcal{M}$
4. *If* $\mathcal{M} \neq \emptyset$*, then* $f_\beta(\mathcal{M}, \mathcal{X}) \neq \emptyset$.

The concept of mappings allows us to define a family of refined operators for fragments of classical logic that captures the examples given before.

Definition 8. *Let* $\Delta : \mathcal{E}_\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{K}_\mathcal{L}$ *be a merging operator and* $\mathcal{L}' \subseteq \mathcal{L}$ *be a* β*-fragment of classical logic with* $\beta \in \mathcal{B}$*. For a* β*-mapping* f_β *we denote with* $\Delta^{f_\beta} : \mathcal{E}_{\mathcal{L}'} \times \mathcal{L}' \rightarrow \mathcal{K}_{\mathcal{L}'}$ *the operator for* $\mathcal{L}'$ *defined as* $\mathrm{Mod}(\Delta^{f_\beta}_\mu(E)) = f_\beta(\mathrm{Mod}(\Delta_\mu(E)), \mathcal{M}od(E))$*. The class* $[\Delta, \mathcal{L}']$ *contains all operators* Δ^{f_β} *where* f_β *is a* β*-mapping and* $\beta \in \mathcal{B}$ *such that* $\mathcal{L}'$ *is a* β*-fragment.*

The next proposition is central in reflecting that the above class captures all refined operators we had in mind, cf. Definition 5.

Proposition 2. *Let* $\Delta : \mathcal{E}_\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{K}_\mathcal{L}$ *be a basic merging operator and* $\mathcal{L}' \subseteq \mathcal{L}$ *a characterizable fragment of classical logic. Then,* $[\Delta, \mathcal{L}']$ *is the set of all* Δ*-refinements for* $\mathcal{L}'$*.*

Proof. Let $\mathcal{L}'$ be a β-fragment for some $\beta \in \mathcal{B}$. Let $\Delta^\star \in [\Delta, \mathcal{L}']$. We show that $\Delta^\star$ is a Δ-refinement for $\mathcal{L}'$. Let $\mu \in \mathcal{L}'$ and $E \in \mathcal{E}_{\mathcal{L}'}$. Since $\Delta^\star \in [\Delta, \mathcal{L}']$ there exists a β-mapping f_β, such that $\mathrm{Mod}(\Delta^\star_\mu(E)) = f_\beta(\mathrm{Mod}(\Delta_\mu(E)), \mathcal{M}od(E))$. By Property 1 in Definition 7 $\Delta^\star_\mu(E)$ is indeed in $\mathcal{K}_{\mathcal{L}'}$. Consistency: If $\mathrm{Mod}(\Delta_\mu(E)) \neq \emptyset$ then $\mathrm{Mod}(\Delta^\star_\mu(E)) \neq \emptyset$ by Property 4 in Definition 7. Otherwise, by Property 2 in Definition 7, we get $\mathrm{Mod}(\Delta^\star_\mu(E)) \subseteq Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) = Cl_\beta(\emptyset) = \emptyset$. Equivalence for $\Delta^\star$ is clear by definition and since f_β is defined on sets of models. Containment: let $\phi \in T_{\mathcal{L}'}(\Delta_\mu(E))$, i.e., $\phi \in \mathcal{L}'$ and $\mathrm{Mod}(\Delta_\mu(E)) \subseteq \mathrm{Mod}(\phi)$. We have $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq Cl_\beta(\mathrm{Mod}(\phi))$ by monotonicity of Cl_β. By Property 2 of Definition 7, $\mathrm{Mod}(\Delta^\star_\mu(E)) \subseteq Cl_\beta(\mathrm{Mod}(\Delta_\mu(E)))$. Since $\phi \in \mathcal{L}'$ we have $Cl_\beta(\mathrm{Mod}(\phi)) = \mathrm{Mod}(\phi)$. Thus, $\mathrm{Mod}(\Delta^\star_\mu(E)) \subseteq \mathrm{Mod}(\phi)$, i.e., $\phi \in T_{\mathcal{L}'}(\Delta^\star_\mu(E))$. Invariance: In case $\Delta_\mu(E) \in \mathcal{K}_{\langle\mathcal{L}'\rangle}$, we have $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) = \mathrm{Mod}(\Delta_\mu(E))$ since $\mathcal{L}'$ is a β-fragment. By Property 3 in Definition 7, we have $\mathrm{Mod}(\Delta^\star_\mu(E)) = f_\beta(\mathrm{Mod}(\Delta_\mu(E)), \mathcal{M}od(E)) = \mathrm{Mod}(\Delta_\mu(E))$. Thus $T_{\mathcal{L}'}(\Delta^\star_\mu(E)) \subseteq T_{\mathcal{L}'}(\Delta_\mu(E))$ as required.

Let $\Delta^\star$ be a Δ-refinement for $\mathcal{L}'$. We show that $\Delta^\star \in [\Delta, \mathcal{L}']$. Let f be defined as follows for any set $\mathcal{M}$ of interpretations and $\mathcal{X}$ a multi-set of sets of interpretations: $f(\emptyset, \mathcal{X}) = \emptyset$. For $\mathcal{M} \neq \emptyset$, if $Cl_\beta(\mathcal{M}) = \mathcal{M}$ then $f(\mathcal{M}, \mathcal{X}) = \mathcal{M}$, otherwise if there exists a pair $(E, \mu) \in (\mathcal{E}_{\mathcal{L}'}, \mathcal{L}')$ such that $\mathcal{M}od(E) = \mathcal{X}$ and $\mathrm{Mod}(\Delta_\mu(E)) = \mathcal{M}$, then we define $f(\mathcal{M}, \mathcal{X}) = \mathrm{Mod}(\Delta^\star_\mu(E))$. If there is no such (E, μ) then we arbitrarily define $f(\mathcal{M}, \mathcal{X})$ as the set consisting of a single model, say the minimal model of $\mathcal{M}$ in the lexicographic order. Note that since $\Delta^\star$ is a Δ-refinement for $\mathcal{L}'$, it satisfies the property of equivalence, thus the actual choice of the pair (E, μ) is not relevant, and hence f is well-defined. Thus the refined operator $\Delta^\star$ behaves like the operator Δ^f.

We show that such a mapping f is a β-mapping. We show that the four properties in Definition 7 hold for f. Property 1 is ensured since for every pair $(\mathcal{M}, \mathcal{X})$, $f(\mathcal{M}, \mathcal{X})$ is closed under β. Indeed, either $f(\mathcal{M}, \mathcal{X}) = \mathcal{M}$ if $\mathcal{M}$ is closed under β, or $f(\mathcal{M}, \mathcal{X}) = \mathrm{Mod}(\Delta^\star_\mu(E))$ and since $\Delta^\star_\mu(E) \in \mathcal{K}_{\mathcal{L}'}$ its set of models is closed under β, or $f(\mathcal{M}, \mathcal{X})$ consists of a single interpretation, and thus is also closed under β. Let us show Property 2, i.e., $f(\mathcal{M}, \mathcal{X}) \subseteq Cl_\beta(\mathcal{M})$ for any pair $(\mathcal{M}, \mathcal{X})$. It is obvious when $\mathcal{M} = \emptyset$ (then $f(\mathcal{M}, \mathcal{X}) = \emptyset$), as well as when $f(\mathcal{M}, \mathcal{X})$ is a singleton and when $\mathcal{M}$ is closed and thus $f(\mathcal{M}, \mathcal{X}) = \mathcal{M}$. Otherwise $f(\mathcal{M}, \mathcal{X}) = \mathrm{Mod}(\Delta^\star_\mu(E))$ and since $\Delta^\star$ satisfies containment $\mathrm{Mod}(\Delta^\star_\mu(E)) \subseteq Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))$. Therefore in any case we have $f(\mathcal{M}, \mathcal{X}) \subseteq Cl_\beta(\mathcal{M})$. Property 3 follows trivially from the definition of $f(\mathcal{M}, \mathcal{X})$ when $\mathcal{M}$ is closed under β. Property 4 is ensured by consistency of $\Delta^\star$. □

Note that the β-mapping which is used in the characterization of refined merging operators differs from the one used in the context of revision (see (Creignou et al. 2014)). Indeed, our mapping has two arguments (and not only one as in the case of revision). The additional multi-set of sets of models representing the profile is required to capture approaches like the Min/Cl_β-based refined operator, which are profile dependent.

IC Postulates

The aim of this section is to study whether refinements of merging operators preserve the IC postulates. We first show that in case the initial operator satisfies the most basic postulates ((IC0)–(IC3)), then so does any of its refinements. It turns out that this result can not be extended to the remaining postulates. For (IC4) we characterize a subclass of refinements for which this postulate is preserved. For the four remaining postulates we study two representative kinds of distance-based merging operators. We show that postulates (IC5) and (IC7) are violated for all of our proposed examples of refined operators with the exception of the Min-based refinement. For (IC6) and (IC8) the situation is even worse in the sense that no refinement of our proposed examples of merging operators can satisfy them neither for $\mathcal{L}_{Horn}$ nor for $\mathcal{L}_{Krom}$. Table 1 gives an overview of the results of this section. However, note that some of the forthcoming results are more general and hold for arbitrary fragments and/or operators.

Proposition 3. *Let* Δ *be a merging operator satisfying postulates* (IC0)–(IC3)*, and* $\mathcal{L}' \subseteq \mathcal{L}$ *a characterizable fragment. Then each* Δ*-refinement for* $\mathcal{L}'$ *satisfies* (IC0)–(IC3) *in* $\mathcal{L}'$ *as well.*

Proof. Since $\mathcal{L}'$ is characterizable there exists a $\beta \in \mathcal{B}$, such that $\mathcal{L}'$ is a β-fragment. Let $\Delta^\star$ be a Δ-refinement for $\mathcal{L}'$. According to Proposition 2 we can assume that $\Delta^\star \in [\Delta, \mathcal{L}']$ is an operator of form Δ^{f_β} where f_β is a suitable β-mapping. In what follows, note that we can restrict ourselves to $E \in \mathcal{E}_{\mathcal{L}'}$ and to $\mu \in \mathcal{L}'$ since we have to show that Δ^{f_β} satisfies (IC0)–(IC3) in $\mathcal{L}'$.

(IC0): Since Δ satisfies (IC0), $\mathrm{Mod}(\Delta_\mu(E)) \subseteq \mathrm{Mod}(\mu)$. Thus, $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq Cl_\beta(\mathrm{Mod}(\mu))$ by

	$(\Delta^{d_H,\Sigma})^{Cl_\beta}$	$(\Delta^{d_H,\mathrm{GMax}})^{Cl_\beta}$	$(\Delta^{d_D,x})^{Cl_\beta}$	$(\Delta^{d,x})^{\mathrm{Min}}$	$(\Delta^{d,x})^{\mathrm{Min}/Cl_\beta}$
IC4	+	-	+	-	+
IC5, IC7	-	-	-	+	-
IC6, IC8	-	-	-	-	-

Table 1: Overview of results for (IC4)–(IC8) for refinements in the Horn and Krom fragment ($x \in \{\Sigma, \mathrm{GMax}\}$, $d \in \{d_H, d_D\}$).

monotonicity of the closure. Hence, $Cl_\beta(\mathrm{Mod}(\Delta_\mu(E))) \subseteq \mathrm{Mod}(\mu)$, since $\mu \in \mathcal{L}'$ and $\mathcal{L}'$ is a β-fragment. According to Property 2 in Definition 7 we have $f_\beta(\mathrm{Mod}(\Delta_\mu(E)), \mathcal{M}od(E)) \subseteq Cl_\beta(\mathrm{Mod}(\Delta_\mu(E)))$, and therefore by definition of $\Delta^\star_\mu$, $\mathrm{Mod}(\Delta^\star_\mu(E)) \subseteq \mathrm{Mod}(\mu)$, which proves that $\Delta^\star_\mu(E) \models \mu$.

(IC1): Suppose μ satisfiable. Since Δ satisfies (IC1), $\Delta_\mu(E)$ is satisfiable. Since Δ^{f_β} is a Δ-refinement (Proposition 2), $\Delta^{f_\beta}_\mu(E)$ is also satisfiable by the property of consistency (see Definition 5).

(IC2): Suppose $\bigwedge E$ is consistent with μ. Since Δ satisfies (IC2), $\Delta_\mu(E) = \bigwedge E \wedge \mu$. We have $\mathrm{Mod}(\Delta^\star_\mu(E)) = f_\beta(\mathrm{Mod}(\Delta_\mu(E)), \mathcal{M}od(E)) = f_\beta(\mathrm{Mod}(\bigwedge E \wedge \mu), \mathcal{M}od(E))$. Since $\bigwedge E \wedge \mu \in \mathcal{L}'$ (observe that it is here necessary that the profiles are in the fragment) by Property 3 of Definition 7 we have $\mathrm{Mod}(\Delta^\star_\mu(E)) = \bigwedge E \wedge \mu$.

(IC3): Let $E_1, E_2 \in \mathcal{E}_{\mathcal{L}'}$ and $\mu_1, \mu_2 \in \mathcal{L}'$ with $E_1 \equiv E_2$ and $\mu_1 \equiv \mu_2$. Since Δ satisfies (IC3), $\Delta_{\mu_1}(E_1) \equiv \Delta_{\mu_2}(E_2)$. By the property of equivalence in Definition 5 we have $\Delta^\star_{\mu_1}(E_1) \equiv \Delta^\star_{\mu_2}(E_2)$. □

A natural question is whether refined operators for characterizable fragments in their full generality preserve other postulates, and if not whether one can nevertheless find some refined operators that satisfy some of the remaining postulates.

First we show that one can not expect to extend Proposition 3 to (IC4). Indeed, in the two following propositions we exhibit merging operators which satisfy all postulates, whereas some of their refinements violate (IC4) in some fragments.

Proposition 4. *Let Δ be a merging operator with $\Delta \in \{\Delta^{d,\Sigma}, \Delta^{d,\mathrm{GMax}}\}$, where d is an arbitrary counting distance. Then the* Min*-based refined operator Δ^{Min} violates postulate* (IC4) *in $\mathcal{L}_{Horn}$ and $\mathcal{L}_{Krom}$. In case d is a drastic distance, Δ^{Min} violates postulate* (IC4) *in every characterizable fragment $\mathcal{L}' \subset \mathcal{L}$.*

Proof. First consider d is a drastic distance. We show that Δ^{Min} violates postulate (IC4) in every characterizable fragment $\mathcal{L}' \subset \mathcal{L}$. Since $\mathcal{L}'$ is a characterizable fragment there exists $\beta \in \mathcal{B}$ such that $\mathcal{L}'$ is a β-fragment. Consider a set of models $\mathcal{M}$ that is not closed under β and that is cardinality-minimum with this property. Such a set exists since $\mathcal{L}'$ is a proper subset of $\mathcal{L}$. Observe that necessarily $|\mathcal{M}| > 1$. Let $m \in \mathcal{M}$, consider the knowledge bases K_1 and K_2 such that $\mathrm{Mod}(K_1) = \{m\}$ and $\mathrm{Mod}(K_2) = \mathcal{M} \setminus \{m\}$. By the choice of $\mathcal{M}$ both K_1 and K_2 are in $\mathcal{K}_{\mathcal{L}'}$, whereas $K_1 \cup K_2$ is not. Let $\mu = \top$. Since the merging operator uses a drastic distance it is easy to see that $\Delta_\mu(\{K_1, K_2\}) = \mathrm{Mod}(K_1) \cup \mathrm{Mod}(K_2)$. Therefore, $\mathrm{Mod}(\Delta^{\mathrm{Min}}_\mu(\{K_1, K_2\})) = \mathrm{Min}(\mathrm{Mod}(K_1) \cup \mathrm{Mod}(K_2))$, and this single element is either a model of K_1 or a model of K_2 (but not of both since they do not share any model). This shows that Δ^{Min} violates (IC4).

Otherwise, d is defined such that there exists an $x > 0$, such that $g(x) < g(x+1)$. We first show that then Δ^{Min} violates postulate (IC4) in $\mathcal{L}_{Horn}$. Let A be a set of atoms such that $|A| = x - 1$ and $A \cap \{a, b\} = \emptyset$. Moreover, consider $E = \{K_1, K_2\}$ with $\mathrm{Mod}(K_1) = \{\emptyset, \{a\}, \{b\}\}$, $\mathrm{Mod}(K_2) = \{A \cup \{a, b\}\}$, and let μ such that $\mathrm{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, A \cup \{a, b\}\}$. Since $g(x) < g(x+1)$, we have $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) = \{\{a\}, \{b\}, A \cup \{a, b\}\}$, which is not closed under intersection. Hence, $\mathrm{Mod}(\Delta^{\mathrm{Min}}_\mu(E))$ contains exactly one of the three models depending on the ordering. Therefore, $\#(\mathrm{Mod}(\Delta^{\mathrm{Min}}_\mu(E)), E) = 1$, and thus violating postulate (IC4).

For $\mathcal{L}_{Krom}$, let $x > 0$ be the smallest index such that $g(x) < g(x+1)$ in the definition of distance d. Note that for any y with $0 < y < x$, $g(y) = g(x)$ thus holds. Let A, A' be two disjoint set of atoms with cardinality $x - 1$ and $A \cap \{a, b, c, d\} = A' \cap \{a, b, c, d\} = \emptyset$. Let us consider $E = \{K_1, K_2\}$ with $\mathrm{Mod}(K_1) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{c, d\}\}$ (in case $x > 1$) resp. $\mathrm{Mod}(K_1) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}\}$ (in case $x = 1$), $\mathrm{Mod}(K_2) = \{A \cup \{a, b\}, A' \cup \{c, d\}\}$, and μ such that $\mathrm{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{c, d\}, A \cup \{a, b\}, A' \cup \{c, d\}\}$. The following table represents the case $x > 1$.

	K_1	K_2	E
$\emptyset$	0	$g(x+1)$	$(g(x+1), 0)$
$\{a\}$	0	$g(x)$	$(g(x), 0)$
$\{b\}$	0	$g(x)$	$(g(x), 0)$
$\{c\}$	0	$g(x)$	$(g(x), 0)$
$\{d\}$	0	$g(x)$	$(g(x), 0)$
$\{a, b\}$	0	$g(x-1)$	$(g(x-1), 0)$
$\{c, d\}$	0	$g(x-1)$	$(g(x-1), 0)$
$A \cup \{a, b\}$	$g(x-1)$	0	$(g(x-1), 0)$
$A' \cup \{c, d\}$	$g(x-1)$	0	$(g(x-1), 0)$

For the case $x > 1$, observe $g(x-1) = g(x) < g(x+1)$, and we have $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) = \{\{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{c, d\}, A \cup \{a, b\}, A' \cup \{c, d\}\}$. For the case $x = 1$, note that A and A' are empty, thus the two last rows of the table coincide with the two rows before. Recall that K_1 is defined differently for this case. Hence, the distances of $\{a, b\}$ and $\{c, d\}$ to K_1 are $g(x) = g(1)$. Thus, we have $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) =$

$\{\{a\},\{b\},\{c\},\{d\},\{a,b\},\{c,d\}\}$. Neither of the $\mathcal{M}$ is closed under ternary majority. Hence, $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E))$ contains exactly one of the six resp. eight models depending on the ordering. Therefore, $\#(\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E)),E) = 1$, thus violating postulate (IC4). □

Proposition 5. *Let $\Delta = \Delta^{d,\mathrm{GMax}}$ be a merging operator where d is an arbitrary non-drastic counting distance. Then the closure-based refined operator Δ^{Cl_β} violates* (IC4) *in $\mathcal{L}_{Horn}$ and $\mathcal{L}_{Krom}$.*

Proof. Since d is not drastic, there exists an $x > 0$ such that $g(x) < g(x+1)$. In what follows, we select the smallest such x. We start with the case $\mathcal{L}_{Horn}$. Let A be a set of atoms of cardinality $x-1$ not containing a, b. Let us consider $E = \{K_1, K_2\}$ with $\mathrm{Mod}(K_1) = \{\emptyset\}$ and $\mathrm{Mod}(K_2) = \{A \cup \{a,b\}\}$, and μ such that $\mathrm{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, A \cup \{a,b\}\}$.

	K_1	K_2	E
$\emptyset$	0	$g(x+1)$	$(g(x+1), 0)$
$\{a\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$\{b\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$A \cup \{a,b\}$	$g(x+1)$	0	$(g(x+1), 0)$

Since $g(x) < g(x+1)$, we have $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) = \{\{a\},\{b\}\}$, which is not closed either under intersection. Hence, $\mathrm{Mod}(\Delta_\mu^{Cl_\wedge}(E)) = \{\{a\},\{b\},\emptyset\}$. Therefore, $\#(\mathrm{Mod}(\Delta_\mu^{Cl_\wedge}(E)),E) = 1$, thus violating (IC4).

For the case $\mathcal{L}_{Krom}$, let us consider two disjoint sets A, A' of atoms not containing a, b, c, d of cardinality $x-1$, the profile $E = \{K_1, K_2\}$ with $\mathrm{Mod}(K_1) = \{\emptyset\}$ and $\mathrm{Mod}(K_2) = \{A \cup \{a,b\}, A' \cup \{c,d\}\}$, and constraing μ such that $\mathrm{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a,b\}, \{c,d\}, A \cup \{a,b\}, A' \cup \{c,d\}\}$.

	K_1	K_2	E
$\emptyset$	0	$g(x+1)$	$(g(x+1), g(0))$
$\{a\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$\{b\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$\{c\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$\{d\}$	$g(1)$	$g(x)$	$(g(x), g(1))$
$\{a,b\}$	$g(2)$	$g(x-1)$	$(g(x-1), g(2))$
$\{c,d\}$	$g(2)$	$g(x-1)$	$(g(x-1), g(2))$
$A \cup \{a,b\}$	$g(x+1)$	$g(0)$	$(g(x+1), g(0))$
$A' \cup \{c,d\}$	$g(x+1)$	$g(0)$	$(g(x+1), g(0))$

In case $x = 1$ note that A and A' are empty and $g(2) > g(x) > g(x-1) = g(0)$ (thus the last four lines collapse into two lines). We have $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) = \{\{a\},\{b\},\{c\},\{d\}\}$, which is not closed under ternary majority. Hence, $\mathrm{Mod}(\Delta_\mu^{Cl_{\mathrm{maj}_3}}(E)) = \{\{a\},\{b\},\{c\},\{d\},\emptyset\}$. In case $x > 1$, we have $g(x+1) > g(x) = g(x-1) = g(2) = g(1)$. Thus, $\mathcal{M} = \mathrm{Mod}(\Delta_\mu(E)) = \{\{a\},\{b\},\{c\},\{d\},\{a,b\},\{c,d\}\}$, which is not closed under ternary majority either and one has to add $\emptyset$. Therefore, in both cases $\#(\mathrm{Mod}(\Delta_\mu^{Cl_{\mathrm{maj}_3}}(E)),E) = 1$, thus violating (IC4). □

In order to identify a class of refinements which satisfy (IC4), we now introduce the notion of fairness for Δ-refinements.

Definition 9. *Let $\mathcal{L}'$ be a fragment of classical logic. A Δ-refinement for $\mathcal{L}'$, $\Delta^\star$, is* fair *if it satisfies the following property for each $E \in \mathcal{E}_{\mathcal{L}'}$, $\mu \in \mathcal{L}'$: If $\#(\Delta_\mu(E),E) \neq 1$ then $\#(\Delta_\mu^\star(E),E) \neq 1$.*

Proposition 6. *Let $\mathcal{L}'$ be a characterizable fragment. (1) The Cl_β-based refinement of both $\Delta^{d_D,\Sigma}$ and $\Delta^{d_D,\mathrm{GMax}}$ for $\mathcal{L}'$ is fair. (2) The Min/Cl_β-based refinement of any merging operator for $\mathcal{L}'$ is fair.*

Proof. Let $\mathcal{L}'$ be a β-fragment. Let $E \in \mathcal{E}_{\mathcal{L}'}$ such that $E = \{K_1, \ldots K_n\}$, $\mu \in \mathcal{L}'$ and let Δ be $\Delta^{d_D,\Sigma}$ or $\Delta^{d_D,\mathrm{GMax}}$ for case (1), resp. let Δ be an arbitray merging operator in case of (2).

Δ^{Cl_β}: If $\#(\Delta_\mu(E),E) > 1$ then, $\#(Cl_\beta(\Delta_\mu(E)),E) \geq \#(\Delta_\mu(E),E) > 1$. Since the drastic distance is used observe that for any model m of μ we have $d(m,E) = n - |\{i \mid m \in K_i\}|$. Thus, if $\#(\Delta_\mu(E),E) = 0$, then $\mathrm{Mod}(\Delta_\mu(E)) \cap \bigcup_i \mathrm{Mod}(K_i) = \emptyset$, and thus $\mathrm{Mod}(\Delta_\mu(E)) = \mathrm{Mod}(\mu)$. In this case $\mathrm{Mod}(\Delta_\mu^{Cl_\beta}(E)) = \mathrm{Mod}(\Delta_\mu(E))$ and therefore $\#(\Delta_\mu^{Cl_\beta}(E),E) = 0$ as well.

$\Delta^{\mathrm{Min}/Cl_\beta}$: If $\#(\Delta_\mu(E),E) = 0$ then $\mathrm{Mod}(\Delta_\mu(E)) \cap \bigcup_i \mathrm{Mod}(K_i) = \emptyset$. By Definition 6 $\Delta_\mu^{\mathrm{Min}/Cl_\beta}(E) = \Delta_\mu^{\mathrm{Min}}(E)$, therefore $\#(\Delta_\mu^{\mathrm{Min}/Cl_\beta}(E),E) = 0$ as well. If $\#(\Delta_\mu(E),E) > 1$ then by Definition 6, $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}/Cl_\beta}) = \mathrm{Mod}(\Delta_\mu^{Cl_\beta}(E))$, thus $\#(\Delta_\mu^{\mathrm{Min}/Cl_\beta}(E),E) \geq \#(\Delta_\mu(E),E) > 1$. □

Fairness turns out to be a sufficient property to preserve the postulate (IC4) as stated in the following proposition.

Proposition 7. *Let Δ be a merging operator satisfying postulate* (IC4)*, and $\mathcal{L}' \subseteq \mathcal{L}$ a characterizable fragment. Then every fair Δ-refinement for $\mathcal{L}'$ satisfies* (IC4) *as well.*

Proof. Consider Δ a merging operator satisfying postulate (IC4). Let $\Delta^\star$ be a fair Δ-refinement for $\mathcal{L}'$. If $\Delta^\star$ does not satisfy (IC4), then there exist $E = \{K_1, K_2\}$ with $K_1, K_2 \in \mathcal{L}'$ and $\mu \in \mathcal{L}'$, with $K_1 \models \mu$ and $K_2 \models \mu$ such that $\mathrm{Mod}(\Delta_\mu^\star(E)) \cap \mathrm{Mod}(K_1) \neq \emptyset$ and $\mathrm{Mod}(\Delta_\mu^\star(E)) \cap \mathrm{Mod}(K_2) = \emptyset$, i.e., such that $\#(\Delta_\mu^\star(E),E) = 1$. Since Δ satisfies postulate (IC4) we have $\#(\Delta_\mu(E),E) \neq 1$, thus contradicting the fairness property in Definition 9. □

With the above result at hand, we can conclude that the Cl_β-based refinement of both $\Delta^{d_D,\Sigma}$ and $\Delta^{d_D,\mathrm{GMax}}$ for $\mathcal{L}'$ as well as the Min/Cl_β-based refinement of any merging operator satisfies (IC4).

Remark 1. *Observe that the distance which is used in distance-based operators matters with respect to the preservation of* (IC4)*, as well as for fairness. Indeed, while the Cl_β-refinement of $\Delta^{d_D,\mathrm{GMax}}$ is fair, and therefore satisfies* (IC4)*, the Cl_β-refinement of $\Delta^{d,\mathrm{GMax}}$ where d is an arbitrary non-drastic counting distance violates postulate* (IC4) *in $\mathcal{L}_{Horn}$ and $\mathcal{L}_{Krom}$, and therefore is not fair.*

For all refinements considered so far we know whether (IC4) is preserved or not, with one single exception: the Cl_β-refinement of $\Delta^{d,\Sigma}$ where d is an arbitrary non-drastic

counting distance. In this case we get a partial positive result.

Proposition 8. *Let Δ be a merging operator with $\Delta = \Delta^{d,\Sigma}$, where d is an arbitrary counting distance that satisfies the triangular inequality. Then the closure-based refined operator Δ^{Cl_β} satisfies postulate* (IC4) *in any characterizable fragment.*

Proof. Let $\mathcal{L}'$ be a β-fragment. Let $E = \{K_1, K_2\}$ with $K_1, K_2 \in \mathcal{L}'$ and $\mu \in \mathcal{L}'$, with $K_1 \models \mu$ and $K_2 \models \mu$. The merging operator Δ satisfies (IC4) therefore $\Delta_\mu(E) \wedge K_1$ is consistent if and only if $\Delta_\mu(E) \wedge K_2$.

If both $\Delta_\mu(E) \wedge K_1$ and $\Delta_\mu(E) \wedge K_2$ are consistent, then so are *a fortiori* $\Delta_\mu^{Cl_\beta}(E) \wedge K_1$ and $\Delta_\mu^{Cl_\beta}(E) \wedge K_2$. Therefore a violation of (IC4) can only occur when both $\Delta_\mu(E) \wedge K_1$ and $\Delta_\mu(E) \wedge K_2$ are inconsistent. We prove that this never occurs. Suppose that $\Delta_\mu(E) \wedge K_1$ is inconsistent, this means that there exists $m \not\in K_1$ such that $min(\mathrm{Mod}(\mu), \leq_E) = d(m, E)$ and that for all $m_1 \in K_1$, $d(m, E) < d(m_1, E)$, i.e., $d(m, K_1) + d(m, K_2) < d(m_1, K_1) + d(m_1, K_2)$ since Σ is the aggregation function. Choose now $m_1 \in K_1$ such that $d(m, K_1) = d(m, m_1)$ and $m_2 \in K_2$ such that $d(m, K_2) = d(m, m_2)$. We have $d(m, K_1) + d(m, K_2) = d(m, m_1) + d(m, m_2) < d(m_1, K_1) + d(m_1, K_2) = d(m_1, K_2)$ since $m_1 \in K_1$ and hence $d(m_1, K_1) = 0$. Since d satisfies the triangular inequality we have $d(m_1, m_2) \leq d(m_1, m) + d(m, m_2)$. But this contradicts $d(m, m_1) + d(m, m_2) < d(m_1, K_2) \leq d(m_1, m_2)$, thus $\Delta_\mu(E) \wedge K_1$ can not be inconsistent. □

Remark 2. *The above proposition together with Proposition 5 shows that the aggregation function that is used in distance-based operators matters with respect to the preservation of the postulate* (IC4).

Interestingly Proposition 8 (recall that the Hamming distance satisfies the triangular inequality) together with the following proposition show that fairness, which is a sufficient condition for preserving (IC4) is not a necessary one.

Proposition 9. *The Cl_β-refinement of $\Delta^{d_H,\Sigma}$ is not fair in $\mathcal{L}_{Horn}$ and in $\mathcal{L}_{Krom}$.*

Proof. We give the proof for $\mathcal{L}_{Horn}$. One can verify that the same example works for $\mathcal{L}_{Krom}$ as well.

Let us consider $E = \{K_1, K_2\}$ and μ in $\mathcal{L}_{Horn}$ with $\mathrm{Mod}(K_1) = \{\{a\}, \{a, b\}, \{a, d\}, \{a, f\}\}$, $\mathrm{Mod}(K_2) = \{\{a, b, c, d, e, f, g\}\}$ and $\mathrm{Mod}(\mu) = \{\{a\}, \{a, b, c\}, \{a, d, e\}, \{a, f, g\}\}$. We have $\mathrm{Mod}(\Delta_\mu^{d_H,\Sigma}(E)) = \{\{a, b, c\}, \{a, d, e\}, \{a, f, g\}\}$, and $\mathrm{Mod}(\Delta_\mu^{Cl_\wedge}(E)) = \{\{a\}, \{a, b, c\}, \{a, d, e\}, \{a, f, g\}\}$. Therefore, $\#(\mathrm{Mod}(\Delta_\mu^{d_H,\Sigma}(E)), E) = 0$, whereas $\#(\mathrm{Mod}(\Delta_\mu^{Cl_\wedge}(E)), E) = 1$, thus proving that fairness is not satisfied. □

It turns out that our refined operators have a similar behavior with respect to postulates (IC5) & (IC7) as well as (IC6) & (IC8). Therefore we will deal with the remaining postulates in pairs. In fact the Min-based refinement satisfies (IC5) and (IC7), whereas the refined operators Δ^{Cl_β} and $\Delta^{\mathrm{Min}/Cl_\beta}$ violate these two postulates.

Proposition 10. *Let Δ be a merging operator satisfying postulates* (IC5) *and* (IC6) *(resp.* (IC7) *and* (IC8)*), and $\mathcal{L}' \subseteq \mathcal{L}$ a characterizable fragment. Then the refined operator Δ^{Min} for $\mathcal{L}'$ satisfies* (IC5) *(resp.* (IC7)*) in $\mathcal{L}'$ as well.*

Proof. Since $\mathcal{L}'$ is characterizable there exists a $\beta \in \mathcal{B}$, such that $\mathcal{L}'$ is a β-fragment.

(IC5): If $\Delta_\mu^{\mathrm{Min}}(E_1) \wedge \Delta_\mu^{\mathrm{Min}}(E_2)$ is inconsistent, then (IC5) is satisfied. Assume that $\Delta_\mu^{\mathrm{Min}}(E_1) \wedge \Delta_\mu^{\mathrm{Min}}(E_2)$ is consistent. Then, by definition of Δ^{Min} we know that $\Delta_\mu(E_1) \wedge \Delta_\mu(E_2)$ is consistent as well. From (IC5) and (IC6) it follows that $\mathrm{Mod}(\Delta_\mu(E_1)) \cap \mathrm{Mod}(\Delta_\mu(E_2)) = \mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2))$. We distinguish two cases. First assume that both $\mathrm{Mod}(\Delta_\mu(E_1))$ and $\mathrm{Mod}(\Delta_\mu(E_2))$ are closed under β. By Definition 2 we know that $\mathrm{Mod}(\Delta_\mu(E_1)) \cap \mathrm{Mod}(\Delta_\mu(E_2)) = \mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2))$ is closed under β as well. Hence, (IC5) is satisfied. For the second case assume that not both $\mathrm{Mod}(\Delta_\mu(E_1))$ and $\mathrm{Mod}(\Delta_\mu(E_2))$ are closed under β. From the definition of Δ^{Min} it follows that $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_1)) \cap \mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_2))$ consists of a single interpretation, say I with $I \in \mathrm{Mod}(\Delta_\mu(E_1)) \cap \mathrm{Mod}(\Delta_\mu(E_2))$. If $\mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2))$ is closed under β we have $I \in \mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_1 \sqcup E_2))$ and (IC5) is satisfied. If $\mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2))$ is not closed under β, then $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_1 \sqcup E_2))$ consists of a single interpretation, say $J \in \mathrm{Mod}(\Delta_\mu(E_1)) \cap \mathrm{Mod}(\Delta_\mu(E_2))$. From $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_1)) \cap \mathrm{Mod}^{\mathrm{Min}}(\Delta_\mu(E_2)) = \{I\}$ it follows that $\mathrm{Min}(\{I, J\}) = I$ and from $\mathrm{Mod}(\Delta_\mu^{\mathrm{Min}}(E_1 \sqcup E_2)) = \{J\}$ it follows that $\mathrm{Min}(\{I, J\}) = J$. Hence, $I = J$ and (IC5) is satisfied.

(IC7): If $\Delta_{\mu_1}^{\mathrm{Min}}(E) \wedge \mu_2$ is inconsistent, then (IC7) is satisfied. Assume that $\Delta_{\mu_1}^{\mathrm{Min}}(E) \wedge \mu_2$ is consistent. Then, by definition of Δ^{Min} we know that $\Delta_{\mu_1}(E) \wedge \mu_2$ is consistent as well. From (IC7) and (IC8) it follows that $\mathrm{Mod}(\Delta_{\mu_1}(E)) \cap \mathrm{Mod}(\mu_2) = \mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}(E))$. We distinguish two cases. First assume that $\mathrm{Mod}(\Delta_{\mu_1}(E))$ is closed under β. By Definition 2 we know that $\mathrm{Mod}(\Delta_{\mu_1}(E)) \cap \mathrm{Mod}(\mu_2) = \mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}(E))$ is closed under β as well. Hence, (IC7) is satisfied. For the second case assume that $\mathrm{Mod}(\Delta_{\mu_1}(E))$ is not closed under β. From the definition of Δ^{Min} it follows that $\mathrm{Mod}(\Delta_{\mu_1}^{\mathrm{Min}}(E)) \cap \mathrm{Mod}(\mu_2)$ consists of a single interpretation, say I with $I \in \mathrm{Mod}(\Delta_{\mu_1}(E)) \cap \mathrm{Mod}(\mu_2)$. If $\mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}(E))$ is closed under β we have $I \in \mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}^{\mathrm{Min}}(E))$ and (IC7) is satisfied. If $\mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}(E))$ is not closed under β, then $\mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}^{\mathrm{Min}}(E))$ consists of a single interpretation, say $J \in \mathrm{Mod}(\Delta_{\mu_1}(E)) \cap \mathrm{Mod}(\mu_2)$. From $\mathrm{Mod}(\Delta_{\mu_1}^{\mathrm{Min}}(E)) \cap \mathrm{Mod}(\mu_2) = \{I\}$ it follows that $\mathrm{Min}(\{I, J\}) = I$ and from $\mathrm{Mod}(\Delta_{\mu_1 \wedge \mu_2}^{\mathrm{Min}}(E)) = \{J\}$ it follows that $\mathrm{Min}(\{I, J\}) = J$. Hence, $I = J$ and (IC7) is satisfied. □

Proposition 11. *Let Δ be a merging operator with $\Delta \in \{\Delta^{d,\Sigma}, \Delta^{d,\mathrm{GMax}}\}$, where d is an arbitrary counting distance. Then the refined operators Δ^{Cl_β} and $\Delta^{\mathrm{Min}/Cl_\beta}$ violate postulates* (IC5) *and* (IC7) *in $\mathcal{L}_{Horn}$ and in $\mathcal{L}_{Krom}$.*

Proof. We give the proof for Δ^{Cl_β} with $\Delta = \Delta^{d,\Sigma}$ where d is associated with a function g. The given examples also apply to GMax and for the refinement $\Delta^{\text{Min}/Cl_\beta}$.

(IC5): Let $\beta \in \{\wedge, \text{maj}_3\}$. Consider $E_1 = \{K_1, K_2, K_3\}$, $E_2 = \{K_4\}$ and μ with $\text{Mod}(K_1) = \{\{a\}, \{a,b\}, \{a,c\}\}$, $\text{Mod}(K_2) = \{\{b\}, \{a,b\}, \{b,c\}\}$, $\text{Mod}(K_3) = \{\{c\}, \{a,c\}, \{b,c\}\}$, $\text{Mod}(K_4) = \{\emptyset, \{b\}\}$, and $\text{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, \{c\}\}$.

	K_1	K_2	K_3	K_4	E_1	$E_1 \sqcup E_2$
$\emptyset$	$g(1)$	$g(1)$	$g(1)$	0	$3g(1)$	$3g(1)$
$\{a\}$	0	$g(1)$	$g(1)$	$g(1)$	$2g(1)$	$3g(1)$
$\{b\}$	$g(1)$	0	$g(1)$	0	$2g(1)$	$2g(1)$
$\{c\}$	$g(1)$	$g(1)$	0	$g(1)$	$2g(1)$	$3g(1)$

Since $g(1) > 0$ by definition of a counting distance, we have $\text{Mod}(\Delta_\mu^{Cl_\beta}(E_1)) = \{\emptyset, \{a\}, \{b\}, \{c\}\}$, $\text{Mod}(\Delta_\mu^{Cl_\beta}(E_2)) = \{\emptyset, \{b\}\}$, and $\text{Mod}(\Delta_\mu^{Cl_\beta}(E_1 \sqcup E_2)) = \{\{b\}\}$, violating (IC5).

(IC7): For $\mathcal{L}_{Horn}$, consider $E = \{K_1, K_2, K_3\}$ with $\text{Mod}(K_1) = \{\{a\}\}$, $\text{Mod}(K_2) = \{\{b\}\}$, $\text{Mod}(K_3) = \{\{a,b\}\}$, and assume $\text{Mod}(\mu_1) = \{\emptyset, \{a\}, \{b\}\}$ and $\text{Mod}(\mu_2) = \{\emptyset, \{a\}\}$.

	K_1	K_2	K_3	E
$\emptyset$	$g(1)$	$g(1)$	$g(2)$	$2g(1) + g(2)$
$\{a\}$	0	$g(2)$	$g(1)$	$g(1) + g(2)$
$\{b\}$	$g(2)$	0	$g(1)$	$g(1) + g(2)$

We have $\text{Mod}(\Delta_{\mu_1}(E)) = \{\{a\}, \{b\}\}$, thus $\text{Mod}(\Delta_{\mu_1}^{Cl_\wedge}(E)) = \{\emptyset, \{a\}, \{b\}\}$. Therefore, $\text{Mod}(\Delta_{\mu_1}^{Cl_\wedge}(E) \wedge \mu_2) = \{\emptyset, \{a\}\}$, whereas $\text{Mod}(\Delta_{\mu_1 \wedge \mu_2}^{Cl_\wedge}(E)) = \{\{a\}\}$, violating (IC7).

For $\mathcal{L}_{Krom}$ let $E = \{K_1, K_2, K_3, K_4, K_5\}$, μ_1 and μ_2 with $\text{Mod}(K_1) = \{\{a\}\}$, $\text{Mod}(K_2) = \{\{b\}\}$, $\text{Mod}(K_3) = \{\{c\}\}$, $\text{Mod}(K_4) = \{\{a,b\}, \{a,c\}\}$, $\text{Mod}(K_5) = \{\{a,b\}, \{b,c\}\}$, $\text{Mod}(\mu_1) = \{\emptyset, \{a\}, \{b\}, \{c\}\}$, and $\text{Mod}(\mu_2) = \{\emptyset, \{a\}\}$.

	K_1	K_2	K_3	K_4	K_5	E
$\emptyset$	$g(1)$	$g(1)$	$g(1)$	$g(2)$	$g(2)$	$2g(2) + 3g(1)$
$\{a\}$	0	$g(2)$	$g(2)$	$g(1)$	$g(1)$	$2g(2) + 2g(1)$
$\{b\}$	$g(2)$	0	$g(2)$	$g(1)$	$g(1)$	$2g(2) + 2g(1)$
$\{c\}$	$g(2)$	$g(2)$	0	$g(1)$	$g(1)$	$2g(2) + 2g(1)$

We have $\text{Mod}(\Delta_{\mu_1}^{Cl_{\text{maj}_3}}(E)) = \{\emptyset, \{a\}, \{b\}, \{c\}\}$, thus $\text{Mod}(\Delta_{\mu_1}^{Cl_{\text{maj}_3}}(E) \wedge \mu_2) = \{\emptyset, \{a\}\}$, and $\text{Mod}(\Delta_{\mu_1 \wedge \mu_2}^{Cl_{\text{maj}_3}}(E)) = \{\{a\}\}$. This violates postulate (IC7). □

Actually in the Horn fragment the negative results of the above proposition can be extended to any fair refinement.

Proposition 12. *Let Δ be a merging operator with $\Delta \in \{\Delta^{d,\Sigma}, \Delta^{d,\text{GMax}}\}$, where d is an arbitrary counting distance. Then any fair refined operator Δ^* violates postulates* (IC5) *and* (IC7) *in $\mathcal{L}_{Horn}$.*

Proof. The same, or simpler examples as in the proof of the previous proposition will work here. We give the proof in the case of $\Delta^{d,\Sigma}$ where d is a counting distance associated with the function g. It is easy to see that the given examples work as well when using the aggregation function GMax. It can be observed in the following that any involved set of models is closed under intersection and hence it can be represented by a Horn formula.

(IC5): Let us consider $E_1 = \{K_1, K_2\}$, $E_2 = \{K_3\}$ and μ with $\text{Mod}(K_1) = \{\{a\}, \{a,b\}\}$,$\text{Mod}(K_2) = \{\{b\}, \{a,b\}\}$, $\text{Mod}(K_3) = \{\emptyset, \{b\}\}$ and $\text{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}\}$. Since $g(1) > 0$ by definition of a counting distance, we have $\text{Mod}(\Delta_\mu(E_1)) = \{\{a\}, \{b\}\}$, and thus $\text{Mod}(\Delta_\mu^*(E_1)) \subseteq \{\emptyset, \{a\}, \{b\}\}$. We can exclude $\text{Mod}(\Delta_\mu^*(E_1)) = \{\{a\}, \{b\}\}$ since it is not closed under $\wedge$. By Definition 9 we can exclude $\text{Mod}(\Delta_\mu^*(E_1)) = \{\{a\}\}$ and $\text{Mod}(\Delta_\mu^\star(E_1)) = \{\{b\}\}$. Therefore either $\text{Mod}(\Delta_\mu^*(E_1)) = \{\emptyset\}$ or $\text{Mod}(\Delta_\mu^*(E_1)) = \{\emptyset, \{a\}, \{b\}\}$. On the one hand, since $\text{Mod}(\Delta_\mu^*(E_2)) = \{\emptyset, \{b\}\}$, in any case $\text{Mod}(\Delta_\mu^*(E_1) \wedge \Delta_\mu^*(E_2))$ contains $\emptyset$. On the other hand $\text{Mod}(\Delta_\mu^*(E_1 \sqcup E_2)) = \{\{b\}\}$. This violates postulate (IC5).

(IC7): There we have $\text{Mod}(\Delta_{\mu_1 \wedge \mu_2}(E)) = \{\{a\}\}$. By properties 3 and 4 of Definition 5 it holds $\text{Mod}(\Delta_{\mu_1 \wedge \mu_2}^\star(E)) = \{\{a\}\}$. Since $\text{Mod}(\Delta_{\mu_1}(E)) = \{\{a\}, \{b\}\}$, it follows that $\text{Mod}(\Delta_{\mu_1}^\star(E)) \subseteq \{\emptyset, \{a\}, \{b\}\}$. We can exclude $\text{Mod}(\Delta_{\mu_1}^\star(E)) = \{\{a\}, \{b\}\}$ since it is not closed under $\wedge$. By Definition 9 we can exclude $\text{Mod}(\Delta_{\mu_1}^\star(E)) = \{\{a\}\}$ and $\text{Mod}(\Delta_{\mu_1}^\star(E)) = \{\{b\}\}$. Hence, $\emptyset \in \text{Mod}(\Delta_{\mu_1}^\star(E))$. Therefore $\emptyset \in \text{Mod}(\Delta_{\mu_1}^\star(E)) \cap \text{Mod}(\mu_2)$ but $\emptyset \notin \text{Mod}(\Delta_{\mu_1 \wedge \mu_2}^\star(E))$ which violates (IC7). □

We leave it as an open question whether this proposition can be extended to Krom. For the two remaining postulates, (IC6) and (IC8), the situation is even worse, since any refinement of the two kinds of distance-based merging operators we considered violates them in $\mathcal{L}_{Horn}$ and in $\mathcal{L}_{Krom}$.

Proposition 13. *Let Δ be a merging operator with $\Delta \in \{\Delta^{d,\Sigma}, \Delta^{d,\text{GMax}}\}$, where d is an arbitrary counting distance. Then any refined operator $\Delta^\star$ violates postulates* (IC6) *and* (IC8) *in $\mathcal{L}_{Horn}$ and in $\mathcal{L}_{Krom}$.*

Proof. As an example we give the proof for (IC6) in $\mathcal{L}_{Horn}$ for $\Delta^{d,\text{GMax}}$. Since $\mathcal{L}_{Horn}$ is an $\wedge$-fragment, there is an $\wedge$-mapping f such that $\Delta^\star = \Delta^f$ and we have $f(\mathcal{M}, \mathcal{X}) \subseteq Cl_\wedge(\mathcal{M})$ with $Cl_\wedge(f(\mathcal{M}, \mathcal{X})) = f(\mathcal{M}, \mathcal{X})$. Let us consider $E_1 = \{K_1, K_2, K_3\}$ and μ with $\text{Mod}(K_1) = \{\{a\}, \{a,b\}\}$, $\text{Mod}(K_2) = \{\{b\}, \{a,b\}\}$, $\text{Mod}(K_3) = \{\emptyset, \{a\}, \{b\}\}$ and $\text{Mod}(\mu) = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$.

	K_1	K_2	K_3	E_1
$\emptyset$	$g(1)$	$g(1)$	0	$(g(1), g(1), 0)$
$\{a\}$	0	$g(1)$	0	$(g(1), 0, 0)$
$\{b\}$	$g(1)$	0	0	$(g(1), 0, 0)$
$\{a,b\}$	0	0	$g(1)$	$(g(1), 0, 0)$

We have $\mathcal{M} = \text{Mod}(\Delta_\mu(E_1)) = \{\{a\}, \{b\}, \{a,b\}\}$. Let us consider the possibilities for $\text{Mod}(\Delta_\mu^\star(E_1)) = f(\mathcal{M}, \mathcal{M}od(E_1))$. If $\emptyset \in f(\mathcal{M}, \mathcal{M}od(E_1))$, then let $E_2 = \{K_4\}$ with K_4 in $\mathcal{L}_{Horn}$ be such that $\text{Mod}(K_4) = \{\emptyset\}$. Thus, $\text{Mod}(\Delta_\mu^\star(E_2)) = \{\emptyset\}$ and $\text{Mod}(\Delta_\mu^\star(E_1) \wedge \Delta_\mu^\star(E_2)) = \{\emptyset\}$. Moreover, $\text{Mod}(\Delta_\mu(E_1 \sqcup E_2)) = \{\emptyset, \{a\}, \{b\}\}$ or $\{\emptyset, \{a\}, \{b\}, \{a,b\}\}$ depending on whether $g(1) < g(2)$ or $g(1) = g(2)$. Since both sets are closed under intersection, we have $\text{Mod}(\Delta_\mu^\star(E_1 \sqcup E_2)) =$

$\mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2))$. Thus $\mathrm{Mod}(\Delta^\star_\mu(E_1 \sqcup E_2)) \not\subseteq \{\emptyset\}$ and (IC6) does not hold.

Otherwise, $f(\mathcal{M}, \mathcal{M}od(E_1)) \subseteq \{\{a\}, \{b\}, \{a, b\}\}$. By symmetry assume w.l.o.g. that $f(\mathcal{M}, \mathcal{M}od(E_1)) \subseteq \{\{a, b\}, \{a\}\}$ (note that $\{\{a\}, \{b\}\} \subseteq f(\mathcal{M}, \mathcal{M}od(E_1))$ would imply $\emptyset \in f(\mathcal{M}, \mathcal{M}od(E_1))$). If $f(\mathcal{M}, \mathcal{M}od(E_1)) = \{\{a\}\}$ or $\{\{a, b\}\}$, then let $E_2 = \{K_1\}$. Then, $\mathrm{Mod}(\Delta_\mu(E_2)) = \{\{a\}, \{a, b\}\} = \mathrm{Mod}(\Delta^\star_\mu(E_2))$, and $\mathrm{Mod}(\Delta^\star_\mu(E_1) \wedge \Delta^\star_\mu(E_2)) = \{\{a\}\}$ or $\{\{a, b\}\}$. Furthermore, $\mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2)) = \{\{a\}, \{a, b\}\} = \mathrm{Mod}(\Delta^\star_\mu(E_1 \sqcup E_2))$, thus violating (IC6). If $f(\mathcal{M}, \mathcal{M}od(E_1)) = \{\{a, b\}, \{a\}\}$, then let $E_2 = \{K_2\}$. Then, $\mathrm{Mod}(\Delta_\mu(E_2)) = \{\{b\}, \{a, b\}\} = \mathrm{Mod}(\Delta^\star_\mu(E_2))$, and $\mathrm{Mod}(\Delta^\star_\mu(E_1) \wedge \Delta^\star_\mu(E_2)) = \{\{a, b\}\}$. Furthermore, $\mathrm{Mod}(\Delta_\mu(E_1 \sqcup E_2)) = \{\{b\}, \{a, b\}\} = \mathrm{Mod}(\Delta^\star_\mu(E_1 \sqcup E_2))$, and thus (IC6) does not hold. □

Conclusion

We have investigated to which extent known merging operators can be refined to work within propositional fragments. Compared to revision, this task is more involved since merging operators have many parameters that have to be taken into account, and the field of investigation is very broad.

We have first defined desired properties any refined merging operator should satisfy and provided a characterization of all refined merging operators. We have shown that the refined merging operators preserve the basic merging postulates, namely (IC0)–(IC3). The situation is more complex for the other postulates. For the postulate (IC4) we have provided a sufficient condition for its preservation by a refinement (fairness). We have shown that this condition is not necessary and it would be interesting to study how to weaken it in order to get a necessary and sufficient condition. For the other postulates, we have focused on two representative families of distance-based merging operators that satisfy the postulates (IC0)–(IC8). For these two families the preservation of the postulates (IC5) and (IC7) depends on the used refinement and it would be interesting to obtain a necessary and sufficient condition for this. In contrast, there is no hope for such a condition for (IC6) and (IC8), since we have shown that any refinement of merging operators belonging to these families violates these postulates.

As future work we are interested in solving the open question of whether Proposition 12 can be extended to the Krom fragment or whether there exists a fair refinement for Krom which satisfies (IC5) or (IC7). We also plan a thorough investigation of the complexity of refined merging operators.

Acknowledgments

This work has been supported by PHC Amadeus project No 29144UC (OeAD FR 12/2013), by the Austrian Science Fund (FWF): P25521, and by the Agence Nationale de la Recherche, ASPIQ project ANR-12-BS02-0003.

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.* 50(2):510–530.

Baral, C.; Kraus, S.; and Minker, J. 1991. Combining multiple knowledge bases. *IEEE Trans. Knowl. Data Eng.* 3(2):208–220.

Bloch, I., and (Eds), A. H. 2001. Fusion: General concepts and characteristics. *Int. J. Intell. Syst.* 16(10):1107–1134.

Booth, R.; Meyer, T.; Varzinczak, I.; and Wassermann, R. 2011. On the link between partial meet, kernel, and infra contraction and its application to Horn logic. *J. Artif. Intell. Res.* 42:31–53.

Chacón, J., and Pino Pérez, R. 2012. Exploring the rationality of some syntactic merging operators. In *Proc. IBERAMIA*, volume 7637 of *Lecture Notes in Computer Science*, 21–30. Springer.

Cholvy, L. 1998. Reasoning about merging information. *Handbook of DRUMS* 3:233–263.

Creignou, N.; Papini, O.; Pichler, R.; and Woltran, S. 2014. Belief revision within fragments of propositional logic. *J. Comput. Syst. Sci.* 80(2):427–449. (Preliminary version in Proc. KR, 2012).

Delgrande, J., and Peppas, P. 2011. Revising Horn theories. In *Proc. IJCAI*, 839–844.

Delgrande, J., and Wassermann, R. 2013. Horn clause contraction functions. *J. Artif. Intell. Res.* 48:475–511.

Konieczny, S., and Pino Pérez, R. 2002. Merging information under constraints: A logical framework. *J. Log. Comput.* 12(5):773–808.

Konieczny, S., and Pino Pérez, R. 2011. Logic based merging. *J. Philosophical Logic* 40(2):239–270.

Konieczny, S.; Lang, J.; and Marquis, P. 2004. DA^2 merging operators. *Artif. Intell.* 157(1-2):49–79.

Lin, J., and Mendelzon, A. 1998. Merging databases under constraints. *Int. J. Cooperative Inf. Syst.* 7(1):55–76.

Lin, J. 1996. Integration of weighted knowledge bases. *Artif. Intell.* 83(2):363–378.

Putte, F. V. D. 2013. Prime implicates and relevant belief revision. *J. Log. Comput.* 23(1):109–119.

Revesz, P. 1993. On the semantics of theory change: Arbitration between old and new information. In *Proc. PODS*, 71–82.

Revesz, P. 1997. On the semantics of arbitration. *IJAC* 7(2):133–160.

Schaefer, T. 1978. The complexity of satisfiability problems. In *Proc. STOC*, 216–226.

Zhuang, Z., and Pagnucco, M. 2012. Model based Horn contraction. In *Proc. KR*, 169–178.

Zhuang, Z.; Pagnucco, M.; and Zhang, Y. 2013. Definability of Horn revision from Horn contraction. In *Proc. IJCAI*.

Belief Revision and Trust

Aaron Hunter
British Columbia Institute of Technology
Burnaby, Canada
aaron_hunter@bcit.ca

Abstract

Belief revision is the process in which an agent incorporates a new piece of information together with a pre-existing set of beliefs. When the new information comes in the form of a report from another agent, then it is clear that we must first determine whether or not that agent should be trusted. In this paper, we provide a formal approach to modeling trust as a pre-processing step before belief revision. We emphasize that trust is not simply a relation between agents; the trust that one agent has in another is often restricted to a particular domain of expertise. We demonstrate that this form of trust can be captured by associating a state-partition with each agent, then relativizing all reports to this state partition before performing belief revision. In this manner, we incorporate only the part of a report that falls under the perceived domain of expertise of the reporting agent. Unfortunately, state partitions based on expertise do not allow us to compare the relative strength of trust held with respect to different agents. To address this problem, we introduce pseudometrics over states to represent differing degrees of trust. This allows us to incorporate simultaneous reports from multiple agents in a way that ensures the most trusted reports will be believed.

Introduction

The notion of trust must be addressed in many agent communication systems. In this paper, we consider one isoloated aspect of trust: the manner in which trust impacts the process of belief revision. Some of the most influential approaches to belief revision have used the simplifying assumption that all new information must be incorporated; however, this is clearly untrue in cases where information comes from an untrusted source. In this paper, we are concerned with the manner in which an agent uses an external notion of trust in order to determine how new information should be integrated with some pre-existing set of beliefs.

Our basic approach is the following. We introduce a simple model of trust that allows an agent to determine if a source can be trusted to distinguish between different pairs of states. We use this notion of trust as a precursor to belief revision. Hence, before revising by a new formula, an agent first determines to what extent the source of the information can be trusted. In many cases, the agent will only incorporate "part" of the formula into their beliefs. We then extend our model of trust to a more general setting, by introducing quantitative measures of trust that allow us to compare the degree to which different agents are trusted. Fundamental properties are introduced and established, and applications are considered.

Preliminaries

Intuition

It is important to note that an agent typically does not trust another agent universally. As such, we will not apply the label "trusted" to another agent; instead, we will say that an agent is trusted with respect to a certain domain of knowledge. This is further complicated by the fact that there are different reasons that an agent may not be trusted. For example, an agent might not be trusted due to their perceived knowledge of a domain. In other cases, an agent might not be trusted due to their perceived dishonesty, or bias. In this paper, our primary focus is on trust as a function of the perceived expertise of other agents. Towards the end, we briefly address the different formal mechanisms that would be required to deal with deceit.

Motivating Example

We introduce a motivating example in commonsense reasoning where an agent must rely on an informal notion of trust in order to inform rational belief change; we will return to this example periodically as we introduce our formal model.

Consider an agent that visits a doctor, having difficulty breathing. Incidentally, the agent is wearing a necklace that prominently features a jewel on a pendant. During the examination, the doctor checks the patient's throat for swelling or obstruction; at the same time, the doctor happens to look at the necklace. Following the examination, the doctor tells the patient "you have a viral infection in your throat - and by the way, you should know that the jewel in your necklace is not a diamond."

The important part about this example is the fact that the doctor provides information about two distinct domains: human health and jewelry. In practice, a patient is very likely to trust the doctor's diagnosis about the viral infection. On the other hand, the patient really has very little reason to trust the doctor's evaluation of the necklace. We suggest that a rational agent should actually incorporate the doctor's statement about the infection into their own beliefs, while essentially

ignoring the comment on the necklace. This approach is dictated by the kind of trust that the patient has in the doctor. Our aim in this paper is to formalize this kind of "localized" domain-specific trust, and then demonstrate how this form of trust is used in practice to inform belief revision.

Trust

Trust consists of two related components. First, we can think of trust in terms of how likely an agent is to believe what another agent says. Alternatively, we can think of trust in terms of the degree to which an agent is likely to allow another to perform actions on their behalf. In this paper, we will be concerned only with the former.

A great deal of existing work on trust focuses on the manner in which an agent develops a *reputation* based on past behaviour. A brief survey of reputation systems is given in (Huynh, Jennings, and Shadbolt 2006). Reputation systems can be used to inform the allocation of tasks (Ramchurn et al. 2009), or to avoid deception (Salehi-Abari and White 2009). The model of trust presented in this paper is not intended to be an alternative to existing reputation systems; we are not concerned with the manner in which an agent learns to trust another. Instead, our focus is simply on developing a suitable model of trust that is expressive enough to inform the process of *belief revision*. The manner in which this model of trust is developed over time is beyond the scope of this paper.

Belief Revision

Belief revision refers to the process in which an agent must integrate new information with some pre-existing beliefs about the state of the world. One of the most influential approaches to belief revision is the AGM approach, in which an agent incorporates the new information while keeping as much of the intial belief state as consistently possible (Alchourrón, Gärdenfors, and Makinson 1985).

This approach was originally defined with respect to a finite set P of propositional variables representing properties of the world. A *state* is a propositional interpretation over P, representing a possible state of the world. A *belief set* is a deductively closed set of formulas, representing the beliefs of an agent. Since P is finite, it follows that every belief set defines a corresponding *belief state*, which is the set of states that an agent considers to be possible. A revision operator is a function that takes a belief set and a formula as input, and returns a new belief set. An AGM revision operator is a revision operator that satisfies the AGM postulates, as specified in (Alchourrón, Gärdenfors, and Makinson 1985).

It turns out that every AGM revision operator is characterized by a total pre-order over possible worlds. To be more precise, a *faithful assignment* is a function that maps each belief set to a total pre-order over states in which the models of the belief set are the minimal states. When an agent is presented with a new formula ϕ for revision, the revised belief state is the set of all minimal models of ϕ in the total pre-order given by the faithful assignment. We refer the reader to (Katsuno and Mendelzon 1992) for a proof of this result, as well as a complete description of the implications. For our purposes, we simply need to know that each AGM revision operator necessarily defines a faithful assignment.

A Model of Trust

Domain-Specific Trust

Assume we have a fixed propositional signature $\mathbf{F}$ as well as a set of agents $\mathbf{A}$. For each $A \in \mathbf{A}$, let Bel_A denote a deductively closed set of formulas over $\mathbf{F}$ called the *belief set* of A. For each A, let $*_A$ denote an AGM revision operator that intuitively captures the way that the agent A revises their beliefs when presented with new information. This revision operator represents sort of an "ideal" revision situation, in which A has complete trust in the new information. We want to modify the way this operator is used, by adding a representation of the extent to which A trusts each other agent $B \in \mathbf{A}$ over $\mathbf{F}$.

We assume that all new information is reported by an agent, so each formula for revision can be labelled with the name of the reporting agent.[1] At this point, we are not concerned with degrees of trust or with resolving conflicts between different sources of information. Instead, we start with a binary notion of trust, where A either trusts B or does not trust B with respect to a particular domain of expertise.

We encode trust by allowing each agent A to associate a partition Π_A^B over possible states with each agent B.

Definition 1 *A* state partition Π *is a collection of subsets of* $2^{\mathbf{F}}$ *that is collectively exhaustive and mutually exclusive. For any state* $s \in 2^{\mathbf{F}}$*, let* $\Pi(s)$ *denote the element of* Π *that contains* s*.*

If $\Pi = \{2^{\mathbf{F}}\}$ then we call Π the *trivial partition* with respect to $\mathbf{F}$. If $\Pi = \{\{s\} \mid s \in 2^{\mathbf{F}}\}$, then we call Π the *unit partition*.

Definition 2 *For each* $A \in \mathbf{A}$ *the* trust function T_A *is a function that maps each* $B \in \mathbf{A}$ *to a state partition* Π_A^B*.*

The partition Π_A^B represents the trust that A has in B over different aspects of knowledge. Informally, the partition encodes states that A will trust B to distinguish. If $\Pi_A^B(s_1) \neq \Pi_A^B(s_2)$, then A will trust that B can distinguish between states s_1 and s_2. Conversely, if $\Pi_A^B(s_1) = \Pi_A^B(s_2)$, then A does not see B as an authority capable of distinguishing between s_1 and s_2. We clarify by returning to our motivating example.

Example Let $\mathbf{A} = \{A, D, J\}$ and let $\mathbf{F} = \{sick, diam\}$. Informally, the fluent $sick$ is true if A has an illness and the fluent $diam$ is true if a certain piece of jewelry that A is wearing contains a real diamond. If we imagine that D represents a doctor and J represents a jeweler, then we can use state partitions to represent the trust that A has in D and J with respect to different domains. Following standard shorthand notation, we represent a state s by the set of fluent symbols that are *true* in s. In order to make the descriptions of a partition more readable, we use a | symbol to visually

[1] This is not a significant restriction. In domains involving sensing or other forms of discovery, we could simply allow an agent A to self-report information with complete trust.

separate different cells. The following partitions are then intuitively plausible in this example:

$$\Pi_A^D := \{sick, diam\}, \{sick\}|\{diam\}, \emptyset$$
$$\Pi_A^J := \{sick, diam\}, \{diamond\}|\{sick\}, \emptyset$$

Hence, A trusts the doctor D to distinguish between states where A is sick as opposed to states where A is not sick. However, A does not trust D to distinguish between worlds that are differentiated by the authenticity of a diamond. The formula $sick \wedge \neg diamond$ encodes the doctor's statement that the agent is sick, and the necklace they are wearing has a fake diamond.

Although the preceding example is simple, it illustrates how a partition can be used to encode the perceived expertise of agents. In the doctor-jeweler example, we could equivalently have defined trust with respect to the set of fluents. In other words, we could have simply said that D is trusted over the fluent $sick$. However, there are many practical cases where this is not sufficient; we do not want to rely on the fluent vocabulary to determine what is a valid feature with respect to trust. For example, a doctor may have specific expertise over lung infections for those working in factories, but not for lung infections for those working in a space shuttle. By using state partitions to encode trust, we are able to capture a very flexible class of distinct areas of trust.

Incorporating Trust in Belief Revision

As indicated previously, we assume each agent A has an AGM belief revision operator $*_A$ for incorporating new information. In this section, we describe how the revision operator $*_A$ is combined with the trust function T_A to define a new, trust-incorporating revision operator $*_A^B$. In many cases, the operator $*_A^B$ will not be an AGM operator because it will fail to satisfy the AGM postulates. In particular, A will not necessarily believe a new formula when it is reported by an untrusted source. This is a desirable feature.

Our approach is to define revision as a two-step process. First, the agent considers the source and the relevant state partition to determine how much of the new information to incorporate. Second, the agent performs standard AGM revision using the faithful assignment corresponding to the belief revision operator.

Definition 3 *Let ϕ be a formula and let $T_A(B) = \Pi_A^B$. Define:*

$$\Pi_A^B[\phi] = \bigcup \{\Pi_A^B(s) \mid s \models \phi\}.$$

Hence $\Pi_A^B[\phi]$ is the union of all cells that contain a model of ϕ.

If A does not trust B to distinguish between states s and t, then any report from B that provides evidence that s is the actual state is also evidence that t is the actual state. When A performs belief revision, it should be with respect to the distinctions that B can be trusted to make. It follows that A need not believe ϕ after revision; instead A should interpret ϕ to be evidence of any state s that is B-indistinguishable from a model of ϕ. Formally, this means that the formula ϕ is construed to be evidence for each state in $\Pi_A^B[\phi]$.

Definition 4 *Let $A, B \in \mathbf{A}$ with $T_A(B) = \Pi_A^B$, and let $*_A$ be an AGM revision operator for A. For any belief set K with corresponding ordering $\prec_K$ given by the underlying faithful assignment, the trust-sensitive revision $K *_A^B \phi$ is the set of formulas true in*

$$\min_{\prec_K}(\{s \mid s \in \Pi_A^B[\phi]\}).$$

So rather than taking the minimal models of ϕ, we take all minimal states that B can not be trusted to distinguish from the minimal models of ϕ.

It is worth remarking that this notion can be formulated synactically as well. Since $\mathbf{F}$ is finite, each state s is defined by a unique, maximal conjunction over literals in $\mathbf{F}$; we simply take the conjunction of all the atomic formulas that are true in s together with the negation of all the atomic formulas that are false in s.

Definition 5 *For any state s, let $prop(s)$ denote the unique, maximal conjunction of literals true in s.*

This definition can be extended for a cell in a state partition.

Definition 6 *Let Π be a state partition. For any state s,*

$$prop(\Pi(s)) = \bigvee \{prop(s') \mid s' \in \Pi(s)\}.$$

Note that $prop(\Pi(s))$ is a well-defined formula in disjunctive normal form, due to the finiteness of $\mathbf{F}$. Intuitively, $prop(\Pi(s))$ is the formula that defines the partition $\Pi(s)$. In the case of a trust partition Π_A^B, we can use this idea to define the *trust expansion* of a formula.

Definition 7 *Let $A, B \in \mathbf{A}$ with the corresponding state partition Π_A^B, and let ϕ be a formula. The* trust expansion *of ϕ for A with respect to B is the formula*

$$\phi_A^B := \bigvee \{prop(\Pi_A^B(s)) \mid s \models \phi\}.$$

Note that this is a finite disjunction of disjunctions, which is again a well defined formula. We refer to ϕ_A^B as the trust expansion of ϕ because it is true in all states that are consistent with ϕ with respect to distinctions that A trusts B to be able to make. It is an expansion because the set of models of ϕ_A^B is normally larger than the set of models of ϕ. The trust sensitive revision operator could equivalently be defined as the normal revision, following translation of ϕ to the corresponding trust expansion.

Example Returning to our example, we consider a few different formulas for revision:

1. $\phi_1 = sick$
2. $\phi_2 = \neg diam$
3. $\phi_3 = sick \wedge \neg diam.$

Suppose that the agent initially believes that they are not sick, and that the diamond they have is real, so $K = \neg sick \wedge diam$. For simplicity, we will assume that the underlying pre-order $\prec_K$ has only two levels: those states where K is true are minimal, and those where K is false are not. We have the following results for revision

1. $K *_A^D \phi_1 = sick \wedge diam$

2. $K *_A^D \phi_2 = \neg sick \wedge diam$

3. $K *_A^D \phi_3 = sick \wedge diam.$

The first result indicates that A believes the doctor when the doctor reports that they are sick. The second result indicates that A essentially ignores a report from the doctor on the subject of jewelry. The third result is perhaps the most interesting. It demonstrates that our approach allows an agent to just incorporate a part of a formula. Hence, even though ϕ_3 is given as a single piece of information, the agent A only incorporates the part of the formula over which the doctor is trusted.

Formal Properties

Basic Results

We first consider extreme cases for trust-sensitive revision operators. Intuitively, if $T_A(B)$ is the trivial partition, then A does not trust B to be able to distinguish between any states. Therefore, A should not incorporate any new information obtained from B. The following proposition makes this observation explicit.

Proposition 1 *If $T_A(B)$ is the trivial partition, then $K *_A^B \phi = K$ for all K and ϕ.*

The other extreme situation occurs when $T_A(B)$ is the unit partition, which consists of all singleton sets. In this case, A trusts B to be able to distinguish between every possible pair of states. It follows from this result that trust sensitive revision operators are not AGM revision operators.

Proposition 2 *If $T_A(B)$ is the unit partition, then $*_A^B = *_A$.*

Hence, if B is universally trusted, then the corresponding trust sensitive revision operator is just the a priori revision operator for A.

Refinements

There is a partial ordering on partitions based on the notion of *refinement*. We say that Π_1 is a refinement of Π_2 just in case, for each $S_1 \in \Pi_1$, there exists $S_2 \in \Pi_2$ such that $S_1 \subseteq S_2$. We also say that Π_1 is *finer* than Π_2. In terms of trust-partitions, refinement has a natural interpretation in terms of "breadth of trust." If the partition corresponding to B is finer than that corresponding to C, it means that B is trusted more broadly than C. To be more precise, it means that B is trusted to distinguish between all of the states that C can distinguish, and possibly more. If B is trusted more broadly that C, it follows that a report from B should give give A more information. This idea is formalized in the following proposition.

Proposition 3 *For any formula ϕ, if Π_A^B is a refinement of Π_A^C, then $|K *_A^B \phi| \subseteq |K *_A^C \phi|$.*

This is a desirable property; if B is trusted over a greater range of states, then fewer states are possible after a report from B.

Multiple Reports

One natural question that arises is how to deal with multiple reports of information from different agents, with different trust partitions. In our example, for instance, we might get a conflicting report from a jeweler with respect to the status of the necklace. In order to facilitate the discussion, we introduce a precise notion of a *report*.

Definition 8 *A* report *is a pair (B, ϕ), where $B \in \mathbf{A}$ and ϕ is a formula.*

We can now extend the definition of trust senstive revision to reports in the obvious manner. In fact, if the revising agent A is clear from the context, we can use the short hand notation:

$$K * (\phi, B) = K *_A^B \phi.$$

The following definition extends the notion of revision to incorporate multiple reports.

Definition 9 *Let $\{A\} \cup \mathbf{B} \subseteq \mathbf{A}$, and let $\Phi = \{(\phi_i, B_i) \mid i < n\}$ be a finite set of reports. Given K, $*$ and $\prec_K$, the trust-sensitive revision $K *_A \Phi$ is the set of formulas true in*

$$\min_{\prec_K}(\{s \mid s \in \Pi_A^{Bi}[\phi_i]\}).$$

So the trust sensitive revision for a finite set of reports from different agents is essentially the normal, single-shot revision by the conjunction of formulas. The only difference is that we expand each formula with respect to the trust partition for a particular reporting agent.

Example In the doctor and jeweler domain, we can consider how how an agent might incorporate a set of reports from D and J. We start with the same initial belief set as before: $K = \neg sick \wedge diam$. Consider the following reports:

1. $\Phi_1 = \{(sick, D), (\neg diam, D)\}$
2. $\Phi_2 = \{(sick, J), (\neg diam, J)\}$
3. $\Phi_3 = \{(sick, D), (\neg diam, J)\}$
4. $\Phi_4 = \{(sick, J), (\neg diam, D)\}$

We have the following results following revision:

1. $K *_A \Phi_1 = sick \wedge diam$
2. $K *_A \Phi_2 = \neg sick \wedge \neg diam$
3. $K *_A \Phi_3 = sick \wedge \neg diam$
4. $K *_A \Phi_4 = \neg sick \wedge diam.$

These results demonstrate how the agent A essentially incorporates information from D and J in domains where they are trusted, and ignores information when they are not trusted. Note that, in this case, D and J are trusted over disjoint sets of states. As a result, it is not possible to have contradictory reports that are equally trusted.

The problem with Definition 9 is that the set of states in the minimization may be empty. This occurs when multiple agents give conflicting reports, and we trust each agent on the domain. In order to resolve this kind of conflict, we need a more expressive form of trust that allows some agents to be trusted more than others. We introduce such a representation in the next section.

Trust Pseudometrics

Measuring Trust

In the previous section, we were concerned with a binary notion of trust that did not include any measure of the strength of trust held in a particular agent or domain. Such an approach is appropriate in cases where we only receive new information from a single source, or from a set of sources that are equally reliable. However, it is not sufficient if we consider cases where several different sources may provide conflicting information. In such cases, we need to determine which information source is the most trust worthy with respect to the domain currently under consideration.

In the binary approach, we associated a partition of the state space with each agent. In order to capture different levels of trust, we would like to introduce a measure of the distance between two states from the perspective of a particular agent. In other words, an agent A would like to associate a distance function d_B over states with each other agent B. If $d_B(s,t) = 0$, then B can not be trusted to distinguish between the states s and t. On the other hand, if $d_B(s,t)$ is very large, then A has a high level of trust in B's ability to distinguish between s and t. The notion of distance that we introduce will be a *psuedometric* on the state space. A pseudometric is a function d that satisfies the following properties for all x, y, z in the domain X:

1. $d(x,x) = 0$
2. $d(x,y) = d(y,x)$
3. $d(x,z) \leq d(x,y) + d(y,z)$

The difference between a *metric* and a pseudometric is that we do not require that $d(x,y) = 0$ implies $x = y$ (the so-called law of indiscernables). This would be undesirable in our setting, because we want to use the distance 0 to represent states that are indistinguishable rather than identical. The first two properties are clearly desirable for a measure of our trust in another agent's ability to discern states. The third property is the triangle inequality, and it is required to guarantee that our trust in other agents is transitive across different domains.

Definition 10 *For each $A \in \mathbf{A}$, a* pseudometric trust function $\mathcal{T}_A$ *is a function that maps each $B \in \mathbf{A}$ to a pseudometric d_B over $2^{\mathbf{F}}$.*

The pair $(2^{\mathbf{F}}, \mathcal{T}_A)$ is called a pseudometric trust space. We would like to model the situation where a sequence of formulas $\Phi = \phi_1, \ldots, \phi_n$ is received from the agents $\mathbf{B} = B_1, \ldots, B_n$, respectively. Note that the order does not matter, we think of the formulas as arriving at the same instant with no preference between them other than the preference induced by the pseudometric trust space.

We associate a sequence of state partitions with each pseudometric trust space.

Proposition 4 *Let $(2^{\mathbf{F}}, \mathcal{T}_A)$ be a pseudometric trust space, let $B \in \mathbf{A} - A$, and let i be a natural number. For each state s, define the set $\Phi^A_B(i)(s)$ as follows:*

$$\Pi^A_B(i)(s) = \{t \mid d_B(s,t) \leq i\}.$$

The collection of sets $\{\Pi^A_B(i)(s) \mid s \in 2^{\mathbf{F}}\}$ is a state partition.

We let $\Pi^A_B(i)$ denote the state partition obtained from this proposition. The cells of the partition $\Pi^A_B(i)$ consist of all states are separated by a distance of no more than i. The following proposition is immediate.

Proposition 5 *$\Pi^A_B(i)$ is a refinement of $\Pi^A_B(j)$, for any $i < j$.*

Hence, a pseudometric trust space defines a sequence of partitions for each agent. This sequence of partitions gets coarser as we increase the index; increasing the index corresponds to requiring a higher level of trust that an agent can distinguish between states. Since we can use Definition 4 to define a trust sensitive revision operator from a state partition, we can now define a trust sensitive revision operator for any fixed distance i between states. Informally, as i increases, we require B to have a greater degree of certainty in order to trust them to distinguish between states. However, it is not clear in advance exactly which i is the right threshold. Our approach will be to find the lowest possible threshold that yields a consistent result.

Note that $\Pi^A_B(i)$ will be a trivial partition for any i that is less than the minimum distance assigned by the underlying pseudometric trust function.

Definition 11 *Let $(2^{\mathbf{F}}, \mathcal{T}_A)$ be a pseudometric trust space, and let m be the least natural number such that $\Pi^A_B(m)$ is non-trival. The* trust sensitive revision operator *for A with respect to B is the trust sensitive revision operator given by $\Pi^A_B(m)$.*

This is a simple extension of our approach based on state partitions. In the next section, we take advantage of the added expressive power of pseudometrics.

Example We modify the doctor example. In order to consider different levels of trust, it is more interesting to consider a domain involving two doctors: a general practitioner D and a specialist S. We also assume that the vocabulary includes two fluents: ear and $skin$. Informally, ear is understood to be true if the patient has an ear infection, whereas $skin$ is true if the patient has skin cancer. The important point is that an ear infection is something that can easily be diagnosed by any doctor, whereas skin cancer is typically diagnosed by a specialist. In order to capture these facts, we define two pseudometrics d_D and d_S. For simplicity, we label the possible states as follows:

$$\begin{aligned} s_1 &= \{ear, skin\} \\ s_2 &= \{ear\} \\ s_3 &= \{skin\} \\ s_4 &= \emptyset \end{aligned}$$

We define the pseudometrics as follows: With these pseudo-

	s_1, s_2	s_1, s_3	s_1, s_4	s_2, s_3	s_2, s_4	s_3, s_4
d_D	1	2	2	2	2	1
d_S	2	2	2	2	2	2

metrics, it is easy to see that both D and S can distinguish all

of the states. However, S is more trusted to distinguish between states related to a skin cancer diagnosis. In our framework, we would like to ensure that this implies S will be trusted in the case of conflicting reports from D and S with respect to skin cancer.

Multiple Reports

We view the distances in a pseudometric trust space as absolute measurements. As such, if $d_B(s,t) > d_C(s,t)$, then we have greater trust in B as opposed to C as far as the ability to discern the states s and t is concerned. We would like to use this intuition to resolve conflicting reports between agents.

Proposition 6 *Let $\{A\} \cup \mathbf{B} \subseteq \mathbf{A}$, and let $\Phi = \{(\phi_i, B_i) \mid i < n\}$ be a finite set of reports. There exists a natural number m such that*

$$\bigcap_{i<n} (\Pi_A^{Bi}[\phi_i](m)) \neq \emptyset.$$

Hence, for any set of reports, we can get a non-intersecting intersection if we take a sufficiently coarse state partition. In many cases this partition will be non-trival. Using this proposition, we define multiple report revision as follows.

Definition 12 *Let $(2^{\mathbf{F}}, \mathcal{T}_A)$ be a pseudometric trust space, let $\Phi = \{(\phi_i, B_i) \mid i < n\}$ be a finite set of reports, and let m be the least natural number such that $\bigcap_{i<n}(\Pi_A^{Bi}[\phi_i](m)) \neq \emptyset$. Given K, $*$ and $\prec_K$, the trust-sensitive revision $K *_A^{\mathbf{B}} \Phi$ is the set of formulas true in*

$$\min_{\prec_K}(\{s \mid s \in \Pi_A^{Bi}[\phi_i](m)\}).$$

Hence, trust-sensitive revision in this context involves finding the finest possible partition that provides a meaningful combination of the reports, and then revising with the corresponding state partition.

Trust and Deceit

To this point, we have only been concerned with modeling the trust that one agent holds in another due to perceived knowledge or expertise. Of course, the issue of trust also arises in cases where one agent suspects that another may be dishonest. However, the manner in which trust must be handled differs greatly in this context. If A does not trust B, then there is little reason for A to believe any part of a message sent directly from B.

Discussion

Related Work

We are not aware of any other work on trust that explicitly deals with the interaction between trust and formal belief revision operators. There is, however, a great deal of work on frameworks for modelling trust. As noted previously, the focus of such work is often on building reputations. One notable approach to this problem with an emphasis on knowledge representation is (Wang and Singh 2007), in which trust is built based on evidence. This kind of approach could be used as a precursor step to build a trust metric, although one would need to account for domain expertise.

Different levels of trust are treated in (Krukow and Nielsen 2007), where a lattice structure is used to represent various levels of trust strength. This is similar to our notion of a trust pseudometric, but it permits incomparable elements. There are certainly situations where this is a reasonable advantage. However, the emphasis is still on the representation of trust in *an agent* as opposed to trust in an agent with respect to a domain.

One notable approach that is similar to ours is the semantics of trust presented in (Krukow and Nielsen 2007), which is a domain-based approach to differential trust in an agent. The emphasis there is on *trust management*, however. That is, the authors are concerned with how agents maintain some record of trust in the other agents; they are not concerned with a differential approach to belief revision.

Conclusion

In this paper, we have developed an approach to trust sensitive belief revision in which an agent is trusted only with respect to a particular domain. This has been formally accomplished first by using state partitions to indicate which states an agent can be trusted to distinguish, and then by using distance functions to quantify the strength of trust. In both cases, the model of trust is used as sort of a precursor to belief revision. Each agent is able to perform belief revision based on a pre-order over states, but the actual formula for revision is parametrized and expanded based on the level of trust held in the reporting agent.

There are many directions for future work, in terms of both theory and applications. As noted previously, one of the subtle distinctions that must be addressed is the difference between trusted *expertise* and trusted *honesty*. The present framework does not explicitly deal with the problem of deception or *belief manipulation* (Hunter 2013); it would be useful to explore how models of trust must differ in this context. In terms of applications, our approach could be used in any domain where agents must make decisions based on beliefs formulated from multiple reports. This is the case, for example, in many networked communication systems.

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.

Hunter, A. 2013. Belief manipulation: A formal model of deceit in message passing systems. In *Proceedings of the Pacific Asia Workshop on Security Informatics*, 1–8.

Huynh, T. D.; Jennings, N. R.; and Shadbolt, N. R. 2006. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 13(2):119–154.

Katsuno, H., and Mendelzon, A. 1992. Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52(2):263–294.

Krukow, K., and Nielsen, M. 2007. Trust structures. *International Journal of Information Security* 6(2-3):153–181.

Ramchurn, S.; Mezzetti, C.; Giovannucci, A.; Rodriguez-Aguilar, J.; Dash, J.; and Jennings, N. 2009. Trust-based mechanisms for robust and efficient task allocation in the presence of execution uncertainty. *JAIR* 35:119–159.

Salehi-Abari, A., and White, T. 2009. Towards con-resistant trust models for distributed agent systems. In *IJCAI*, 272–277.

Wang, Y., and Singh, M. P. 2007. Formal trust model for multiagent systems. In *IJCAI*, 1551–1556.

On the Non-Monotonic Description Logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$

Oliver Fernández Gil*
University of Leipzig
Department of Computer Science
fernandez@informatik.uni-leipzig.de

Abstract

In the last 20 years many proposals have been made to incorporate non-monotonic reasoning into description logics, ranging from approaches based on default logic and circumscription to those based on preferential semantics. In particular, the non-monotonic description logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ uses a combination of the preferential semantics with minimization of a certain kind of concepts, which represent atypical instances of a class of elements. One of its drawbacks is that it suffers from the problem known as the *property blocking inheritance*, which can be seen as a weakness from an inferential point of view. In this paper we propose an extension of $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$, namely $\mathcal{ALC}$+$\mathbf{T}^{+}_{\mathsf{min}}$, with the purpose to solve the mentioned problem. In addition, we show the close connection that exists between $\mathcal{ALC}$+$\mathbf{T}^{+}_{\mathsf{min}}$ and concept-circumscribed knowledge bases. Finally, we study the complexity of deciding the classical reasoning tasks in $\mathcal{ALC}$+$\mathbf{T}^{+}_{\mathsf{min}}$.

Introduction

Description Logics (DLs) (Baader *et al.* 2003) are a well-investigated family of logic-based knowledge representation formalisms. They can be used to represent knowledge of a problem domain in a structured and formal way. To describe this kind of knowledge each DL provides constructors that allow to build concept descriptions. A knowledge base consists of a TBox that states general assertions about the problem domain and an ABox that asserts properties about explicit individuals.

Nevertheless, classical description logics do not provide any means to reason about exceptions. In the past 20 years research has been directed with the purpose to incorporate non-monotonic reasoning formalisms into DLs. In (Baader & Hollunder 1995a), an integration of Reiter's default logic (Reiter 1980) within the terminological language $\mathcal{ALCF}$ is proposed and later extended in (Baader & Hollunder 1995b) to allow the use of priorities between default rules. Taking a different approach, (Bonatti, Lutz, & Wolter 2009) introduces circumscribed DLs and analyses in detail the complexity of reasoning in circumscribed extensions of expressive description logics. In addition, recent works (Casini & Straccia 2010; Britz, Meyer, & Varzinczak 2011; Giordano *et al.* 2013a) attempt to introduce defeasible reasoning by extending DLs with preferential and rational semantics based on the KLM approach to propositional non-monotonic reasoning (Lehmann & Magidor 1992).

In particular, the logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ introduced in (Giordano *et al.* 2013b) combines the use of a preferential semantics and the minimization of a certain kind of concepts. This logic is built on top of the description logic $\mathcal{ALC}$ (Schmidt-Schauß & Smolka 1991) and is based on a typicality operator $\mathbf{T}$ whose intended meaning is to single out the *typical* instances of a class C of elements. The underlying semantics of $\mathbf{T}$ is based on a preference relation over the domain. More precisley, classical $\mathcal{ALC}$ interpretations are equipped with a partial order over the domain elements setting a preference relation among them. Based on such an order, for instance, the set of *typical birds* or $\mathbf{T}(\mathsf{Bird})$, comprises those individuals from the domain that are birds and minimal in the class of all birds with respect to the preference order. Using this operator, the subsumption statement $\mathbf{T}(\mathsf{Bird}) \sqsubseteq \mathsf{Fly}$ expresses that *typical birds fly*. In addition, the use of a minimal model semantics considers models that minimize the atypical instances of Bird. Then, when no information is given about whether a bird is able to fly or not, it is possible to *assume* that it flies in view of the assertion $\mathbf{T}(\mathsf{Bird}) \sqsubseteq \mathsf{Fly}$.

As already pointed out by the authors, the preferential order over the domain limits the logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ in the sense that if a class P is an exceptional case of a superclass B, then no default properties from B can be inherited by P during the reasoning process, including those that are unrelated with the exceptionality of P with respect to B. For example:

$$\mathsf{Penguin} \sqsubseteq \mathsf{Bird}$$
$$\mathbf{T}(\mathsf{Bird}) \sqsubseteq \mathsf{Fly} \sqcap \mathsf{Winged}$$
$$\mathbf{T}(\mathsf{Penguin}) \sqsubseteq \neg\mathsf{Fly}$$

It is not possible to infer that typical penguins have wings, even when the only reason for them to be exceptional with respect to birds is that they normally do not fly.

In the present paper we extend the non-monotonic logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ from (Giordano *et al.* 2013b) with the introduction of several preference relations. We show how this extension can handle the inheritance of defeasible properties, resembling the use of abnormality predicates from circumscription (McCarthy 1986). In addition, we show the close relationship between the extended non-monotonic logic

*Supported by DFG Graduiertenkolleg 1763 (QuantLA).

$\mathcal{ALC}$**+T**$^{+}_{\mathsf{min}}$ and *concept-circumscribed* knowledge bases (Bonatti, Lutz, & Wolter 2009). Based on such a relation, we provide a complexity analysis of the different reasoning tasks showing $\mathsf{NExp}^{\mathsf{NP}}$- completeness for concept satisfiability and co-$\mathsf{NExp}^{\mathsf{NP}}$-completeness for subsumption and instance checking.

Missing proofs can be found in the long version of the paper at http://www.informatik.uni-leipzig.de/~fernandez/NMR14long.pdf.

The logic $\mathcal{ALC}$**+T**$_{\mathsf{min}}$

We recall the logic $\mathcal{ALC}$**+T** proposed in (Giordano *et al.* 2013b) and its non-monotonic extension $\mathcal{ALC}$**+T**$_{\mathsf{min}}$. Let $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ be three countable sets of *concept names, role names and individual names*, respectively. The language defined by $\mathcal{ALC}$**+T** distinguishes between normal concept descriptions and *extended concept* descriptions which are formed according to the following syntax rules:

$$C ::= A \mid \neg C \mid C \sqcap D \mid \exists r.C,$$
$$C_e ::= C \mid \mathbf{T}(A) \mid \neg C_e \mid C_e \sqcap D_e$$

where $A \in \mathsf{N_C}, r \in \mathsf{N_R}$, C and D are classical $\mathcal{ALC}$ concept descriptions, C_e and D_e are extended concept descriptions, and $\mathbf{T}$ is the newly introduced operator. We use the usual abbreviations $C \sqcup D$ for $\neg(\neg C \sqcap \neg D)$, $\forall r.C$ for $\neg\exists r.\neg C$, $\top$ for $A \sqcup \neg A$ and $\bot$ for $\neg\top$.

A knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. The TBox $\mathcal{T}$ contains subsumption statements $C \sqsubseteq D$ where C is a classical $\mathcal{ALC}$ concept or an extended concept of the form $\mathbf{T}(A)$, and D is a classical $\mathcal{ALC}$ concept. The Abox $\mathcal{A}$ contains assertions of the form $C_e(a)$ and $r(a, b)$ where C_e is an extended concept, $r \in \mathsf{N_R}$ and $a, b \in \mathsf{N_I}$. The assumption that the operator $\mathbf{T}$ is applied to concept names is without loss of generality. For a complex $\mathcal{ALC}$ concept C, one can always introduce a fresh concept name A_C which can be made equivalent to C by adding the subsumption statements $A_C \sqsubseteq C$ and $C \sqsubseteq A_C$ to the background TBox. Then, $\mathbf{T}(C)$ can be equivalently expressed as $\mathbf{T}(A_C)$.

In order to provide a semantics for the operator $\mathbf{T}$, usual $\mathcal{ALC}$ interpretations are equipped with a preference relation $<$ over the domain elements:

Definition 1 (Interpretation in $\mathcal{ALC}$**+T**)**.** An $\mathcal{ALC}$**+T** interpretation $\mathcal{I}$ is a tuple $(\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <)$ where:

- $\Delta^{\mathcal{I}}$ is the domain,
- $.^{\mathcal{I}}$ is an interpretation function that maps concept names to subsets of $\Delta^{\mathcal{I}}$ and role names to binary relations over $\Delta^{\mathcal{I}}$,
- $<$ is an irreflexive and transitive relation over $\Delta^{\mathcal{I}}$ that satisfies the following condition (**Smoothness Condition**): for all $S \subseteq \Delta^{\mathcal{I}}$ and for all $x \in S$, either $x \in \mathsf{Min}_<(S)$ or $\exists y \in \mathsf{Min}_<(S)$ such that $y < x$, with $\mathsf{Min}_<(S) = \{x \in S \mid \nexists y \in S \text{ s.t. } y < x\}$.

The operator $\mathbf{T}$ is interpreted as follows: $[\mathbf{T}(A)]^{\mathcal{I}} = \mathsf{Min}_<(A^{\mathcal{I}})$. For arbitrary concept descriptions, $.^{\mathcal{I}}$ is inductively extended in the same way as for $\mathcal{ALC}$ taking into account the introduced semantics for $\mathbf{T}$.

As mentioned in (Giordano *et al.* 2013b; 2009), $\mathcal{ALC}$**+T** is still monotonic and has several limitations. In the following we present the logic $\mathcal{ALC}$**+T**$_{\mathsf{min}}$, proposed in (Giordano *et al.* 2013b) as a non-monotonic extension of $\mathcal{ALC}$**+T**, where a preference relation is defined between $\mathcal{ALC}$**+T** interpretations and only minimal models are considered.

First, we introduce the modality $\Box$ as in (Giordano *et al.* 2013b).

Definition 2. Let $\mathcal{I}$ be an $\mathcal{ALC}$**+T** interpretation and C a concept description. Then, $\Box C$ is interpreted under $\mathcal{I}$ in the following way:

$$(\Box C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \text{ for all } y \in \Delta^{\mathcal{I}} \text{ if } y < x \text{ then } y \in C^{\mathcal{I}}\}$$

We remark that $\Box C$ does not extend the syntax of $\mathcal{ALC}$**+T**. The purpose of using it is to characterize elements of the domain with respect to whether all their predecessors in $<$ are instances of C or not. For example, $\Box\neg\mathsf{Bird}$ defines a concept such that $d \in (\Box\neg\mathsf{Bird})^{\mathcal{I}}$ if all the predecessors of d, with respect to $<$ under the interpretation $\mathcal{I}$, are not instances of Bird. Hence, it is not difficult to see that:

$$[\mathbf{T}(\mathsf{Bird})]^{\mathcal{I}} = (\mathsf{Bird} \sqcap \Box\neg\mathsf{Bird})^{\mathcal{I}}$$

Then, the idea is to prefer models that minimize the instances of $\neg\Box\neg\mathsf{Bird}$ in order to minimize the number of *atypical* birds.

Now, let $\mathcal{L}_{\mathbf{T}}$ be a finite set of concept names occurring in the knowledge base. These are the concepts whose atypical instances are meant to be minimized. For each interpretation $\mathcal{I}$, the set $\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}}$ represents all the instance of concepts of the form $\neg\Box\neg A$ for all $A \in \mathcal{L}_{\mathbf{T}}$. Formally,

$$\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} = \{(x, \neg\Box\neg A) | x \in (\neg\Box\neg A)^{\mathcal{I}}, \text{ with } x \in \Delta^{\mathcal{I}}, A \in \mathcal{L}_{\mathbf{T}}\}.$$

Based on this, the notion of minimal models is defined in the following way.

Definition 3 (Minimal models)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <_{\mathcal{I}})$, $\mathcal{J} = (\Delta^{\mathcal{J}}, .^{\mathcal{J}}, <_{\mathcal{J}})$ be two interpretations. We say that $\mathcal{I}$ is preferred to $\mathcal{J}$ with respect to the set $\mathcal{L}_{\mathbf{T}}$ (denoted as $\mathcal{I} <_{\mathcal{L}_{\mathbf{T}}} \mathcal{J}$), iff:

- $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$,
- $a^{\mathcal{I}} = a^{\mathcal{J}}$ for all $a \in \mathsf{N_I}$,
- $\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} \subset \mathcal{J}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}}$.

An interpretation $\mathcal{I}$ is a minimal model of $\mathcal{K}$ with respect to $\mathcal{L}_{\mathbf{T}}$ (denoted as $\mathcal{I} \models^{\mathcal{L}_{\mathbf{T}}}_{\mathsf{min}} \mathcal{K}$) iff $\mathcal{I} \models \mathcal{K}$ and there is no model $\mathcal{J}$ of $\mathcal{K}$ such that $\mathcal{J} <_{\mathcal{L}_{\mathbf{T}}} \mathcal{I}$.

Based on the notion of minimal models, the standard reasoning tasks are defined for $\mathcal{ALC}$**+T**$_{\mathsf{min}}$.

- *Knowledge base consistency (or satisfiability):* A knowledge base $\mathcal{K}$ is consistent w.r.t. $\mathcal{L}_{\mathbf{T}}$, if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models^{\mathcal{L}_{\mathbf{T}}}_{\mathsf{min}} \mathcal{K}$.
- *Concept satisfiability:* An extended concept C_e is satisfiable with respect to $\mathcal{K}$ if there exists a minimal model $\mathcal{I}$ of $\mathcal{K}$ w.r.t. $\mathcal{L}_{\mathbf{T}}$ such that $C_e^{\mathcal{I}} \neq \emptyset$.

- *Subsumption:* Let C_e and D_e be two extended concepts. C_e is subsumed by D_e w.r.t. $\mathcal{K}$ and $\mathcal{L}_{\mathbf{T}}$, denoted as $\mathcal{K} \models^{\mathcal{L}_{\mathbf{T}}}_{\mathsf{min}} C_e \sqsubseteq D_e$, if $C_e^{\mathcal{I}} \subseteq D_e^{\mathcal{I}}$ for all minimal models $\mathcal{I}$ of $\mathcal{K}$.
- *Instance checking:* An individual name a is an instance of an extended concept C_e w.r.t. $\mathcal{K}$, denoted as $\mathcal{K} \models^{\mathcal{L}_{\mathbf{T}}}_{\mathsf{min}} C_e(a)$, if $a^{\mathcal{I}} \in C_e^{\mathcal{I}}$ in all the minimal models $\mathcal{I}$ of $\mathcal{K}$.

Regarding the computational complexity, the case of *knowledge base consistency* is not interesting in itself since the logic $\mathcal{ALC}$**+T** enjoys the finite model property (Giordano *et al.* 2013b). Note that if there exists a finite model $\mathcal{I}$ of $\mathcal{K}$, then the sets that are being minimized are finite. Therefore, every descending chain starting from $\mathcal{I}$ with respect to $<_{\mathcal{L}_{\mathbf{T}}}$ must be finite and a minimal model of $\mathcal{K}$ always exists. Thus, the decision problem only requires to decide knowledge base consistency of the underlying monotonic logic $\mathcal{ALC}$**+T** which has been shown to be EXPTIME-complete (Giordano *et al.* 2009). For the other reasoning tasks, a $\mathsf{NExp}^{\mathsf{NP}}$ upper bound is provided for *concept satisfiability* and a co-$\mathsf{NExp}^{\mathsf{NP}}$ upper bound for subsumption and instance checking (Giordano *et al.* 2013b).

Extending $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ with more typicality operators

As already mentioned in (Giordano *et al.* 2013b; 2009), the use of a global relation to represent that one individual is more typical than another one, limits the expressive power of the logic. It is not possible to express that an individual x is more typical than an individual y with respect to some aspect As_1 and at the same time y is more typical than x (or not comparable to x) with respect to a different aspect As_2. This, for example, implies that a subclass cannot inherit any property from a superclass, if the subclass is already exceptional with respect to one property of the superclass. This effect is also known as *property inheritance blocking* (Pearl 1990; Geffner & Pearl 1992), and is a known problem in preferential extensions of DLs based on the KLM approach.

We revisit the example from the introduction to illustrate this problem.

Example 4. Consider the following knowledge base:

$$\begin{aligned} \mathsf{Penguin} &\sqsubseteq \mathsf{Bird} \\ \mathbf{T}(\mathsf{Bird}) &\sqsubseteq \mathsf{Fly} \sqcap \mathsf{Winged} \\ \mathbf{T}(\mathsf{Penguin}) &\sqsubseteq \neg\mathsf{Fly} \end{aligned}$$

Here, *penguins* represent an exceptional subclass of *birds* in the sense that they *usually are unable to fly*. However, it might be intuitive to conclude that they *normally have wings* ($\mathbf{T}(\mathsf{Penguin}) \sqsubseteq \mathsf{Winged}$) since although birds fly because they have wings, having wings does not imply the ability to fly. In fact, as said before, it is not possible to sanction this kind of conclusion in $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$. The problem is that due to the global character of the order $<$ among individuals of the domain, once an element d is assumed to be a *typical* penguin, then automatically a more preferred individual e must exist that is a *typical* bird. This rules out the possibility to apply the non-monotonic assumption represented by the second assertion to d.

In relation with circumscription, this situation can be modelled using abnormality predicates to represent exceptionality with respect to different aspects (McCarthy 1980; 1986). The following example shows a knowledge base which is defined using abnormality concepts similar as the examples in (Bonatti, Lutz, & Wolter 2009).

Example 5.

$$\begin{aligned} \mathsf{Penguin} &\sqsubseteq \mathsf{Bird} \\ \mathsf{Bird} &\sqsubseteq \mathsf{Fly} \sqcup Ab_1 \\ \mathsf{Bird} &\sqsubseteq \mathsf{Winged} \sqcup Ab_2 \\ \mathsf{Penguin} &\sqsubseteq \neg\mathsf{Fly} \sqcup Ab_{\mathsf{penguin}} \end{aligned}$$

The semantics of circumscription allows to consider only models that minimize the instances of the abnormality concepts. In this example, concepts Ab_1 and Ab_2 are used to represent birds that are atypical with respect to two independent aspects (i.e.: Fly and Winged). If the minimization forces an individual d to be a *not abnormal* penguin (i.e.: d is not an instance of Ab_{penguin}), then it must be an instance of Ab_1, but at the same time nothing forces it to be an instance of Ab_2. Therefore, it is possible to assume that d has wings because of the minimization of Ab_2.

In this paper, we follow a suggestion given in (Giordano *et al.* 2013b) that asks for the extension of the logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ with more preferential relations in order to express typicality of a class with respect to different aspects. We define the logic $\mathcal{ALC}$+$\mathbf{T}^+$ and its extension $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ in a similar way as for $\mathcal{ALC}$**+T** and $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$, but taking into account the possibility to use more than one typicality operator.

We start by fixing a finite number of typicality operators $\mathbf{T}_1, \ldots, \mathbf{T}_k$. Classical concept descriptions and extended concept descriptions are defined by the following syntax:

$$\begin{aligned} C &::= A \mid \neg C \mid C \sqcap D \mid \exists r.C, \\ C_e &::= C \mid \mathbf{T}_i(A) \mid \neg C_e \mid C_e \sqcap D_e, \end{aligned}$$

where all the symbols have the same meaning as in $\mathcal{ALC}$**+T** and $\mathbf{T}_i$ ranges over the set of typicality operators. The semantics is defined as an extension of the semantics for $\mathcal{ALC}$**+T** that takes into account the use of more than one **T** operator.

Definition 6 (Interpretations in $\mathcal{ALC}$+$\mathbf{T}^+$)**.** An interpretation $\mathcal{I}$ in $\mathcal{ALC}$+$\mathbf{T}^+$ is a tuple $(\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <_1, \ldots, <_k)$ where:

- $\Delta^{\mathcal{I}}$ is the domain,
- $<_i$ $(1 \leq i \leq k)$ is an irreflexive and transitive relation over $\Delta^{\mathcal{I}}$ satisfying the Smoothness Condition.

Typicality operators are interpreted in the expected way with respect to the different preference relations over the domain: $[\mathbf{T}_i(A)]^{\mathcal{I}} = \mathsf{Min}_{<_i}(A^{\mathcal{I}})$.

Similar as for $\mathcal{ALC}$**+T**, we introduce for each preference relation $<_i$ an indexed box modality $\Box_i$ such that:

$$(\Box_i C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : \text{ if } y <_i x \text{ then } y \in C^{\mathcal{I}}\}$$

Then, the set of typical instances of a concept A with respect to the i^{th} typical operator can be expressed in terms of the indexed $\Box$ modalities:

$$[\mathbf{T}_i(A)]^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid x \in (A \sqcap \Box_i \neg A)^{\mathcal{I}}\}$$

Now, we define the extension of $\mathcal{ALC}$+$\mathbf{T}^+$ that results in the non-monotonic logic $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$. Let $\mathcal{L}_{\mathbf{T}_1}, \ldots, \mathcal{L}_{\mathbf{T}_k}$ be k finite sets of concept names. Given an $\mathcal{ALC}$+$\mathbf{T}^+$ interpretation $\mathcal{I}$, the sets $\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_i}}$ are defined as:

$$\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_i}} = \{(x, \neg\Box_i\neg A) \mid x \in (\neg\Box_i\neg A)^{\mathcal{I}} \wedge A \in \mathcal{L}_{\mathbf{T}_i}\}$$

Based on these sets, we define the preference relation $<^+_{\mathcal{L}_{\mathbf{T}}}$ on $\mathcal{ALC}$+$\mathbf{T}^+$ interpretations that characterizes the non-monotonic semantics of $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$.

Definition 7 (Preference relation)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <_{i_1}, \ldots, <_{i_k})$, $\mathcal{J} = (\Delta^{\mathcal{J}}, .^{\mathcal{J}}, <_{j_1}, \ldots, <_{j_k})$ be two interpretations. We say that $\mathcal{I}$ is preferred to $\mathcal{J}$ (denoted as $<^+_{\mathcal{L}_{\mathbf{T}}}$) with respect to the sets $\mathcal{L}_{\mathbf{T}_i}$, iff:

- $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$,
- $a^{\mathcal{I}} = a^{\mathcal{J}}$ for all $a \in \mathsf{N_I}$,
- $\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_i}} \subseteq \mathcal{J}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_i}}$ for all $1 \leq i \leq k$,
- $\exists \ell$ s.t. $\mathcal{I}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_\ell}} \subset \mathcal{J}^{\Box^-}_{\mathcal{L}_{\mathbf{T}_\ell}}$.

An $\mathcal{ALC}$+$\mathbf{T}^+$ interpretation $\mathcal{I}$ is a minimal model of $\mathcal{K}$ (denoted as $\mathcal{I} \models^{\mathcal{L}_{\mathbf{T}^+}}_{\mathsf{min}} \mathcal{K}$) iff $\mathcal{I} \models \mathcal{K}$ and there exists no interpretation $\mathcal{J}$ such that: $\mathcal{J} \models \mathcal{K}$ and $\mathcal{J} <^+_{\mathcal{L}_{\mathbf{T}}} \mathcal{I}$. The different reasoning tasks are defined in the usual way, but with respect to the new entailment relation $\models^{\mathcal{L}_{\mathbf{T}^+}}_{\mathsf{min}}$.

We revise Example 4 to show how to distinguish between a bird being typical with respect *to being able to fly* or *to having wings*, in $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$. The example shows the use of two typicality operators $\mathbf{T}_1$ and $\mathbf{T}_2$, where $<_1$ and $<_2$ are the underlying preference relations.

Example 8.

$$\begin{aligned} \mathsf{Penguin} &\sqsubseteq \mathsf{Bird} \\ \mathbf{T}_1(\mathsf{Bird}) &\sqsubseteq \mathsf{Fly} \\ \mathbf{T}_2(\mathsf{Bird}) &\sqsubseteq \mathsf{Winged} \\ \mathbf{T}_1(\mathsf{Penguin}) &\sqsubseteq \neg\mathsf{Fly} \end{aligned}$$

In the example, we use two preference relations to express typicality of birds with respect to two different aspects independently. The use of a second preference relation permits that typical penguins can also be typical birds with respect to $<_2$. Therefore, it is possible to infer that typical penguins do have wings. Looking from the side of individual elements: having the assertion $\mathsf{Penguin}(e)$, the minimal model semantics allows to assume that e is a typical penguin and also a typical bird with respect to $<_2$, even when a bird d must exist such that d is preferred to e with respect to $<_1$.

It is interesting to observe that the defeasible property *not being able to fly*, for penguins, is stated with respect to $\mathbf{T}_1$. If instead, we use $\mathbf{T}_2(\mathsf{Penguin}) \sqsubseteq \neg\mathsf{Fly}$, there will be minimal models where e is an instance of $\mathbf{T}_1(\mathsf{Bird})$ and others where it is an instance of $\mathbf{T}_2(\mathsf{Penguin})$. This implies that it will not be possible to infer for e, the defeasible properties corresponding to the most specific concept it belongs to.

The same problem is realized, with respect to circumscription in Example 5, where some minimal models prefer e to be a *normal* bird ($e \in \neg Ab_1$), while others consider e as a *normal* penguin ($e \in \neg Ab_{\mathsf{penguin}}$). To address this problematic about specificity, one needs to use priorities between the minimized concepts (or abnormality predicates) (McCarthy 1986; Bonatti, Lutz, & Wolter 2009).

In contrast, for the formulation in the example, the semantics induced by the preferential order $<_1$ does not allow to have interpretations where $e \in \mathsf{Penguin}$, $e \in \mathbf{T}_1(\mathsf{Bird})$ and $e \notin \mathbf{T}_1(\mathsf{Penguin})$, i.e., the treatment of specificity comes for free in the semantics of the logic.

Complexity of reasoning in $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$

In the following, we show that reasoning in $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ is $\mathsf{NExp}^{\mathsf{NP}}$-complete for *concept satisfiability* and co-$\mathsf{NExp}^{\mathsf{NP}}$-complete for *subsumption* and *instance checking*. As a main tool we use the close correspondence that exists between *concept-circumscribed* knowledge bases in the DL $\mathcal{ALC}$ (Bonatti, Lutz, & Wolter 2009) and $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ knowledge bases. In fact, this relation has been pointed out in (Giordano *et al.* 2013b) with respect to the logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$. However, on the one hand, the provided mapping from $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ into *concept-circumscribed* knowledge bases is not polynomial, and instead a tableaux calculus is used to show the upper bounds for the main reasoning tasks in $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$. On the other hand, the relation in the opposite direction is only given with respect to the logic $\mathcal{ALCO}$+$\mathbf{T}_{\mathsf{min}}$, which extends $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ by allowing the use of nominals.

First, we improve the mapping proposed in (Giordano *et al.* 2013b) by giving a simpler polynomial reduction, that translates $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ knowledge bases into *concept-circumscribed* knowledge bases while preserving the entailment relation under the translation. Second, we show that using more than one typicality operator, it is possible to reduce the problem of concept satisfiability for *concept-circumscribed* knowledge bases in $\mathcal{ALC}$, into the concept satisfiability problem for $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$.

We start by introducing circumscribed knowledge bases in the DL $\mathcal{ALC}$, as defined in (Bonatti, Lutz, & Wolter 2009). We obviate the use of priorities between minimized predicates.

Definition 9. A circumscribed knowledge base is an expression of the form $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$ where $\mathsf{CP} = (M, F, V)$ is a circumscription pattern such that M, F, V partition the predicates (i.e.: concept and role names) used in $\mathcal{T}$ and $\mathcal{A}$. The set M identifies those concept names whose extension is minimized, F those whose extension must remain fixed and V those that are free to vary. A circumscribed knowledge base where $M \cup F \subseteq \mathsf{N_C}$ is called a *concept-circumscribed* knowledge base.

To formalize a semantics for circumscribed knowledge bases, a preference relation $<_{\mathsf{CP}}$ is defined on interpretations by setting $\mathcal{I} <_{\mathsf{CP}} \mathcal{J}$ iff:

- $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$,
- $a^{\mathcal{I}} = a^{\mathcal{J}}$ for all $a \in \mathsf{N_I}$,
- $A^{\mathcal{I}} = A^{\mathcal{J}}$ for all $A \in F$,
- $A^{\mathcal{I}} \subseteq A^{\mathcal{J}}$ for all $A \in M$ and there exists an $A' \in M$ such that $A'^{\mathcal{I}} \subset A'^{\mathcal{J}}$.

An interpretation $\mathcal{I}$ is a model of $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T},\mathcal{A})$ if $\mathcal{I}$ is a model of $(\mathcal{T},\mathcal{A})$ and there is no model $\mathcal{I}'$ of $(\mathcal{T},\mathcal{A})$ with $\mathcal{I}' <_{\mathsf{CP}} \mathcal{I}$. The different reasoning tasks can be defined in the same way as above.

Similar as for circumscribed knowledge bases in (Bonatti, Lutz, & Wolter 2009), one can show that concept satisfiability, subsumption and instance checking can be polynomially reduced to one another in $\mathcal{ALC}$**+T**$^{+}_{\mathsf{min}}$. However, to reduce instance checking into concept satisfiability slightly different technical details have to be considered.

Lemma 10. *Let $\mathcal{K} = (\mathcal{T},\mathcal{A})$ be an $\mathcal{ALC}$**+T**$^{+}$ knowledge base, C_e an extended concept, $\mathcal{L}_{\mathbf{T}_1},\ldots,\mathcal{L}_{\mathbf{T}_k}$ be finite sets of concept names and A a fresh concept name not occurring in $\mathcal{K}$ and C_e. Then, $\mathcal{K} \models^{\mathcal{L}_{\mathbf{T}^+}}_{\mathsf{min}} C_e(a)$ iff $\neg \boldsymbol{T}_{k+1}(A) \sqcap \neg C_e$ is unsatisfiable w.r.t. $\mathcal{K}' = (\mathcal{T} \cup \{\top \sqsubseteq A\}, \mathcal{A} \cup \{(\neg \boldsymbol{T}_{k+1}(A))(a)\})$, where $\mathcal{L}_{\boldsymbol{T}_{k+1}} = \{A\}$.*

Note that this reduction requires the introduction of an additional typicality operator $\mathbf{T}_{k+1}$. Nevertheless, this does not represent a problem in terms of complexity since, as it will be shown in the following, the complexity does not depend on the number of typicality operators k whenever $k \geq 2$.

Upper Bound

Before going into the details of the reduction we need to define the notion of a signature.

Definition 11. Let $\mathsf{N_T}$ be the set of all the concepts of the form $\mathbf{T}_i(A)$ where $A \in \mathsf{N_C}$. A signature Σ for $\mathcal{ALC}$**+T**$^{+}$ is a finite subset of $\mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_T}$. We denote by $\Sigma|_{\mathcal{ALC}}$ the set $\Sigma \setminus \mathsf{N_T}$.

The signature $\mathsf{sig}(C_e)$ of an extended concept C_e is the set of all concept names, role names and concepts from $\mathsf{N_T}$ that occur in C_e. Similarly, the signature $\mathsf{sig}(\mathcal{K})$ of an $\mathcal{ALC}$**+T**$^{+}$ knowledge base $\mathcal{K}$ is the union of the signatures of all concept descriptions occurring in $\mathcal{K}$. Finally, we denote by $\mathsf{sig}(E_1,\ldots,E_m)$ the set $\mathsf{sig}(E_1) \cup \ldots \cup \mathsf{sig}(E_m)$, where each E_i is either an extended concept or a knowledge base.

Let $\mathcal{K} = (\mathcal{T},\mathcal{A})$ be an $\mathcal{ALC}$**+T**$^{+}$ knowledge base, $\mathcal{L}_{\mathbf{T}_1},\ldots,\mathcal{L}_{\mathbf{T}_k}$ finite sets of concept names and Σ be any signature with $\mathsf{sig}(\mathcal{K}) \subseteq \Sigma$. A corresponding circumscribed knowledge base $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}',\mathcal{A}')$, with $\mathcal{K}' = (\mathcal{T}',\mathcal{A}')$, is built in the following way:

- For every concept A such that it belongs to some set $\mathcal{L}_{\mathbf{T}_i}$ or $\mathbf{T}_i(A) \in \Sigma$, a fresh concept name A^*_i is introduced. These concepts are meant to represent the atypical elements with respect to A and $<_i$ in $\mathcal{K}$, i.e., $\neg\Box_i\neg A$.
- Every concept description C defined over Σ is transformed into a concept $\bar{C}$ by replacing every occurrence of $\mathbf{T}_i(A)$ by $(A \sqcap \neg A^*_i)$.
- The TBox $\mathcal{T}'$ is built as follows:
  - $\bar{C} \sqsubseteq \bar{D} \in \mathcal{T}'$ for all $C \sqsubseteq D \in \mathcal{T}$,
  - For each new concept A^*_i the following assertions are included in $\mathcal{T}'$:

$$A^*_i \equiv \exists r_i.(A \sqcap \neg A^*_i) \quad (1)$$

$$\exists r_i.A^*_i \sqsubseteq A^*_i \quad (2)$$

    where r_i is a fresh role symbol, not occurring in Σ, introduced to represent the relation $<_i$.
- $\mathcal{A}'$ results from replacing every assertion of the form $C(a)$ in $\mathcal{T}$ by the assertion $\bar{C}(a)$.
- Let $\mathcal{L}_{\mathbf{T}}$ be the set:

$$\bigcup_{j=1}^{k} \bigcup_{A \in \mathcal{L}_{\mathbf{T}_j}} A^*_j$$

  then, the concept circumscription pattern CP is defined as $\mathsf{CP} = (M,F,V) = (\mathcal{L}_{\mathbf{T}}, \emptyset, \Sigma|_{\mathcal{ALC}} \cup \{A^*_i \mid A^*_i \notin \mathcal{L}_{\mathbf{T}}\} \cup \{r_i \mid 1 \leq i \leq k\})$.

One can easily see that the provided encoding is polynomial in the size of $\mathcal{K}$. The use of the signature Σ is just a technical detail and since it is chosen arbitrarily, one can also select it properly for the encoding of the different reasoning tasks.

The idea of the translation is to simulate each order $<_i$ with a relation r_i and at the same time fulfill the semantics underlying the $\mathbf{T}_i$ operators. The first assertion, $A^*_i \equiv \exists r_i.(A \sqcap \neg A^*_i)$, intends to express that the atypical elements with respect to A and $<_i$ are those, and only those, that have an r_i-successor e that is an instance of A and at the same time a not atypical A, i.e. , $e \in \mathbf{T}_i(A)$. Indeed, this is a consequence from the logic $\mathcal{ALC}$**+T**$^{+}_{\mathsf{min}}$ because the order $<_i$ is transitive. However, since it is not possible to enforce transitivity of r_i when translated into $\mathcal{ALC}$, we need to use the second assertion $\exists r_i.A^*_i \sqsubseteq A^*_i$. This prevents to have the following situation:

$$d \in A^*_1 \quad d \in B \sqcap \neg B^*_1 \quad (d,e) \in r_1 \quad e \in A \sqcap \neg A^*_1 \quad e \in B^*_1$$

In the absence of assertion (2), this would be consistent with respect to $\mathcal{T}'$, but it would not satisfy the aim of the translation since the typical B-element d would have a predecessor (r_i-successor) e which is atypical with respect to B. In fact, the translation provided in (Giordano *et al.* 2013b) also deals with this situation, but all the possible cases are asserted explicitly yielding an exponential encoding.

The following auxiliary lemma shows that a model of $(\mathcal{T}',\mathcal{A}')$ can always be transformed into a model, that only differs in the interpretation of r_i, and $(r_i)^{-1}$ is irreflexive, transitive and *well-founded.*

Lemma 12. *Let $\mathcal{I}$ be an $\mathcal{ALC}$ interpretation such that $\mathcal{I} \models (\mathcal{T}',\mathcal{A}')$. Then, there exists $\mathcal{J}$ such that $\mathcal{J} \models (\mathcal{T}',\mathcal{A}')$, $X^{\mathcal{I}} = X^{\mathcal{J}}$ for all $X \in \Sigma|_{\mathcal{ALC}} \cup \bigcup A^*_i$, and for each r_i we have: $\left(r_i{}^{\mathcal{J}}\right)^{-1}$ is irreflexive, transitive and well-founded.*

Since *well-foundedness* implies the Smoothness Condition, the previous lemma allows us to assume (without loss of generality) that $\left(r_i{}^{\mathcal{I}}\right)^{-1}$ is irreflexive, transitive and satisfies the Smoothness Condition for every model $\mathcal{I}$ of $\mathcal{K}'$.

Now, we denote by $\mathcal{M}_{\mathcal{K}}$ the set of models of $\mathcal{K}$ and by $\mathcal{M}_{\mathcal{K}'}$ the set of models of $\mathcal{K}'$. With the help of the previous lemma, we show that there exists a *one-to-one* correspondence between $\mathcal{M}_{\mathcal{K}}$ and $\mathcal{M}_{\mathcal{K}'}$. We start by defining a mapping φ that transforms $\mathcal{ALC}$**+T**$^{+}$ interpretations into $\mathcal{ALC}$ interpretations.

Definition 13. We define a mapping φ from $\mathcal{ALC}$**+T**$^+$ interpretations into $\mathcal{ALC}$ interpretations such that $\varphi(\mathcal{I}) = \mathcal{J}$ iff:

- $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$,
- $X^{\mathcal{J}} = X^{\mathcal{I}}$ for each $X \in \Sigma|_{\mathcal{ALC}}$,
- $(A_i^*)^{\mathcal{J}} = (\neg\Box_i\neg A)^{\mathcal{I}}$ for each fresh concept name A_i^*,
- $(r_i)^{\mathcal{J}} = (<_i)^{-1}$ for all $i, 1 \leq i \leq k$,
- $a^{\mathcal{J}} = a^{\mathcal{I}}$, for all $a \in \mathsf{N_I}$.

Remark. We stress that interpretations are considered only with respect to concept and role names occurring in Σ for $\mathcal{ALC}$**+T**$^+$, and $\Sigma|_{\mathcal{ALC}} \cup \{A_i^*\} \cup \{r_i\}$ for $\mathcal{ALC}$. All the other concept and role names from $\mathsf{N_C}$ and $\mathsf{N_R}$ are not relevant to distinguish one interpretation from another one. This is, if $\mathcal{I}$ and $\mathcal{J}$ are two $\mathcal{ALC}$**+T**$^+$ interpretations, then $\mathcal{I} \equiv \mathcal{J}$ iff $X^{\mathcal{I}} = X^{\mathcal{J}}$ for all $X \in \Sigma \cap (\mathsf{N_C} \cup \mathsf{N_R})$ and $(<_i)^{\mathcal{I}} = (<_i)^{\mathcal{J}}$ for all $i, 1 \leq i \leq k$. The same applies for $\mathcal{ALC}$ interpretations, but with respect to $\Sigma|_{\mathcal{ALC}} \cup \{A_i^*\} \cup \{r_i\}$.

Next, we show that φ is indeed a bijection from $\mathcal{M}_{\mathcal{K}}$ to $\mathcal{M}_{\mathcal{K}'}$.

Lemma 14. *The mapping φ is a bijection from $\mathcal{M}_{\mathcal{K}}$ to $\mathcal{M}_{\mathcal{K}'}$, such that for every $\mathcal{I} \in \mathcal{M}_{\mathcal{K}}$ and each extended concepts C_e defined over Σ: $C_e^{\mathcal{I}} = (\bar{C}_e)^{\varphi(\mathcal{I})}$.*

Proof. First, we show that for each $\mathcal{I} \in \mathcal{M}_{\mathcal{K}}$ it holds that: $\varphi(\mathcal{I}) \in \mathcal{M}_{\mathcal{K}'}$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <_1, \ldots, <_k)$ be a model of $\mathcal{K}$ and assume that $\varphi(\mathcal{I}) = \mathcal{J}$. We observe that since $[\mathbf{T}_i(A)]^{\mathcal{I}} = (A \sqcap \Box_i\neg A)^{\mathcal{I}}$, then by definition of φ it follows that:

$$[\mathbf{T}_i(A)]^{\mathcal{I}} = (A \sqcap \neg A_i^*)^{\mathcal{J}} \tag{3}$$

Consequently, one can also see that for every extended concept C_e defined over Σ and every element $d \in \Delta^{\mathcal{I}}$:

$$d \in C_e^{\mathcal{I}} \text{ iff } d \in (\bar{C}_e)^{\mathcal{J}} \tag{4}$$

This can be shown by a straightforward induction on the structure of C_e where the base cases are A and $\mathbf{T}_i(A)$. Hence, it follows that $C_e^{\mathcal{I}} = (\bar{C}_e)^{\mathcal{J}}$ for every extended concept C_e defined over Σ.

Now, we show that $\mathcal{J} \models (\mathcal{T}', \mathcal{A}')$. From (4), it is clear that $\mathcal{J} \models \bar{C} \sqsubseteq \bar{D}$ for all $\bar{C} \sqsubseteq \bar{D} \in \mathcal{T}'$. In addition, since $a^{\mathcal{J}} = a^{\mathcal{I}}$ for all $a \in \mathsf{N_I}$, $\mathcal{J}$ satisfies each assertion in $\mathcal{A}'$. It is left to show that each GCI in $\mathcal{T}'$ containing an occurrence of a fresh role r_i is also satisfied by $\mathcal{J}$. For each $d \in \Delta^{\mathcal{I}}$ and concept name A_i^*, it holds:

$$\begin{aligned} d \in (A_i^*)^{\mathcal{J}} &\text{ iff } d \in (\neg\Box_i\neg A)^{\mathcal{I}} \\ &\text{iff } \exists e \in \Delta^{\mathcal{I}} \text{ s.t. } e <_i d \text{ and } e \in [\mathbf{T}_i(A)]^{\mathcal{I}} \\ &\text{iff } (d,e) \in (r_i)^{\mathcal{J}} \text{ and } e \in (A \sqcap \neg A_i^*)^{\mathcal{J}} \quad \text{by (3)} \\ &\text{iff } d \in (\exists r_i.(A \sqcap \neg A_i^*))^{\mathcal{J}} \end{aligned}$$

The case for the second GCI ($\exists r_i.A_i^* \sqsubseteq A_i^*$) can be shown in a very similar way. Thus, $\mathcal{J} \models (\mathcal{T}', \mathcal{A}')$ and consequently φ is a function from $\mathcal{M}_{\mathcal{K}}$ into $\mathcal{M}_{\mathcal{K}'}$.

Second, we show that for any model $\mathcal{J}$ of $\mathcal{K}'$ (i.e. $\mathcal{J} \in \mathcal{M}_{\mathcal{K}'}$), there exists $\mathcal{I} \in \mathcal{M}_{\mathcal{K}}$ with $\varphi(\mathcal{I}) = \mathcal{J}$. Let $\mathcal{J}$ be an arbitrary model of $\mathcal{K}'$, we build an $\mathcal{ALC}$**+T**$^+$ interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}}, <_1, \ldots, <_k)$ in the following way:

- $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$,
- $X^{\mathcal{I}} = X^{\mathcal{J}}$ for each $X \in \Sigma|_{\mathcal{ALC}}$,
- $<_i = \left(r_i^{\mathcal{J}}\right)^{-1}$ for all $i, 1 \leq i \leq k$,
- $a^{\mathcal{I}} = a^{\mathcal{J}}$, for all $a \in \mathsf{N_I}$.

Next, we show that $(\neg\Box_i\neg A)^{\mathcal{I}} = (A_i^*)^{\mathcal{J}}$. Assume that $d \in (\neg\Box_i\neg A)^{\mathcal{I}}$ for some $d \in \Delta^{\mathcal{I}}$, then there exists $e <_i d$ such that $e \in A^{\mathcal{I}}$ and $e \in [\mathbf{T}_i(A)]^{\mathcal{I}}$. This means that for all $f <_i e$(or $(e,f) \in r_i^{\mathcal{J}}$): $f \notin A^{\mathcal{I}}$. Hence, $e \in A^{\mathcal{J}}$ and $e \notin (A_i^*)^{\mathcal{J}}$. All in all, we have $(d,e) \in r_i^{\mathcal{J}}$ and $e \in (A \sqcap \neg A_i^*)^{\mathcal{J}}$, therefore $d \in (A_i^*)^{\mathcal{J}}$. Conversely, assume that $d \in (A_i^*)^{\mathcal{J}}$. Assertion (1) in $\mathcal{T}'$ implies that there exists e such that $(d,e) \in (r_i)^{\mathcal{J}}$ and $e \in A^{\mathcal{J}}$. By construction of $\mathcal{I}$ we have $e <_i d$ and $e \in A^{\mathcal{I}}$. Thus, $d \in (\neg\Box_i\neg A)^{\mathcal{I}}$ and we can conclude that $(\neg\Box_i\neg A)^{\mathcal{I}} = (A_i^*)^{\mathcal{J}}$. Having this, it follows that $\varphi(\mathcal{I}) = \mathcal{J}$. In addition, similar as for equation (3), we have:

$$[\mathbf{T}_i(A)]^{\mathcal{I}} = (A \sqcap \neg A_i^*)^{\mathcal{J}} \tag{5}$$

A similar reasoning, as above yields that $\mathcal{I} \models \mathcal{K}$. This implies that φ is surjective. It is not difficult to see, from the definition of φ, that it is also injective. Thus, φ is a bijection from $\mathcal{M}_{\mathcal{K}}$ to $\mathcal{M}_{\mathcal{K}'}$. □

The previous lemma establishes a *one to one* correspondence between $\mathcal{M}_{\mathcal{K}}$ and $\mathcal{M}_{\mathcal{K}'}$. Then, since $\mathcal{K}$ is an arbitrary $\mathcal{ALC}$**+T**$^+$ knowledge base, Lemma 14 also implies that knowledge base consistency in $\mathcal{ALC}$**+T**$^+$ can be polynomially reduced to knowledge base consistency in $\mathcal{ALC}$, which is EXPTIME-complete (Baader *et al.* 2003).

Theorem 15. *In $\mathcal{ALC}$**+T**$^+$, deciding knowledge base consistency is* EXPTIME*-complete.*

In addition, since $\mathcal{ALC}$ enjoys the finite model property, this is also the case for $\mathcal{ALC}$**+T**$^+$. Using the same argument given before for $\mathcal{ALC}$**+T** and $\mathcal{ALC}$**+T**$_{\mathsf{min}}$, deciding knowledge base consistency in $\mathcal{ALC}$**+T**$^+_{\mathsf{min}}$ reduces to the same problem with respect to the underlying monotonic logic $\mathcal{ALC}$**+T**$^+$. Therefore, we obtain the following theorem.

Theorem 16. *In $\mathcal{ALC}$**+T**$^+_{\mathsf{min}}$, deciding knowledge base consistency is* EXPTIME*-complete.*

Now, we show that φ is not only a bijection from $\mathcal{M}_{\mathcal{K}}$ to $\mathcal{M}_{\mathcal{K}'}$, but it is also *order-preserving* with respect to $<^+_{\mathcal{L}_{\mathbf{T}}}$ and $<_{\mathsf{CP}}$.

Lemma 17. *Let $\mathcal{I}$ and $\mathcal{J}$ be two models of $\mathcal{K}$. Then, $\mathcal{I} <^+_{\mathcal{L}_{\mathbf{T}}} \mathcal{J}$ iff $\varphi(\mathcal{I}) <_{\mathsf{CP}} \varphi(\mathcal{J})$.*

Proof. Assume that $\mathcal{I} <^+_{\mathcal{L}_{\mathbf{T}}} \mathcal{J}$. Then, for all $A \in \mathcal{L}_{T_i}$ we have that $(\neg\Box_i\neg A)^{\mathcal{I}} \subseteq (\neg\Box_i\neg A)^{\mathcal{J}}$ and in particular, for some j and $A' \in \mathcal{L}_{T_j}$ we have $(\neg\Box_j\neg A')^{\mathcal{I}} \subset (\neg\Box_j\neg A')^{\mathcal{J}}$. By definition of φ, we know that $(\neg\Box_i\neg A)^{\mathcal{I}} = (A_i^*)^{\varphi(\mathcal{I})}$. Hence, for all $A_i^* \in M$ we have that $(A_i^*)^{\varphi(\mathcal{I})} \subseteq (A_i^*)^{\varphi(\mathcal{J})}$ and $(A'^*_j)^{\varphi(\mathcal{I})} \subset (A'^*_j)^{\varphi(\mathcal{J})}$. Thus, $\varphi(\mathcal{I}) <_{\mathsf{CP}} \varphi(\mathcal{J})$. The other direction can be shown in the same way. □

The following lemma is an easy consequence from the previous one and the fact that φ is bijection (which implies that φ is invertible).

Lemma 18. *Let $\mathcal{I}$ and $\mathcal{J}$ be $\mathcal{ALC}$+$\boldsymbol{T}^+$ and $\mathcal{ALC}$ interpretations, respectively. Then,*

$$\mathcal{I} \models^{\mathcal{L}_{T^+}}_{\mathsf{min}} \mathcal{K} \;\; \textit{iff} \;\; \varphi(\mathcal{I}) \models \mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}') \quad \text{(a)}$$

$$\mathcal{J} \models \mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}') \;\; \textit{iff} \;\; \varphi^{-1}(\mathcal{J}) \models^{\mathcal{L}_{T^+}}_{\mathsf{min}} \mathcal{K} \quad \text{(b)}$$

Thus, we have a correspondence between minimal models of $\mathcal{K}$ and models of $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}')$. Based on this, it is easy to reduce each reasoning task from $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ into the equivalent task with respect to *concept-circumscribed* knowledge bases. The following lemma states the existence of such a reduction for concept satisfiability, the cases for subsumption and instance checking can be proved in a very similar way.

Lemma 19. *An extended concept C_0 is satisfiable w.r.t. to $\mathcal{K}$ and $\mathcal{L}_{\boldsymbol{T}_1}, \ldots, \mathcal{L}_{\boldsymbol{T}_k}$ iff $\bar{C}_0$ is satisfiable in* $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}')$.

Proof. Let us define Σ as $\mathsf{sig}(\mathcal{K}, C_0)$.

($\Rightarrow$) Assume that $\mathcal{I}$ is a minimal model of $\mathcal{K}$ with $C_0^{\mathcal{I}} \neq \emptyset$. The application of Lemma 18 tells us that $\varphi(\mathcal{I}) \models \mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}')$. In addition, from Lemma 14 we have that $C_0^{\mathcal{I}} = (\bar{C}_0)^{\varphi(\mathcal{I})}$. Thus, $\bar{C}_0$ is satisfiable in $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}', \mathcal{A}')$.
($\Leftarrow$) The argument is similar, but using φ^{-1}. □

Finally, from the complexity results proved in (Bonatti, Lutz, & Wolter 2009) for the different reasoning tasks with respect to *concept-circumscribed* knowledge bases in $\mathcal{ALC}$, we obtain the following upper bounds.

Theorem 20. *In $\mathcal{ALC}$+$\boldsymbol{T}^+_{\mathsf{min}}$, it is in* $\mathsf{NExp}^{\mathsf{NP}}$ *to decide concept satisfiability and in co-*$\mathsf{NExp}^{\mathsf{NP}}$ *to decide subsumption and instance checking.*

Lower Bound

To show the lower bound, we reduce the problem of concept satisfiability with respect to *concept-circumscribed* knowledge bases in $\mathcal{ALC}$, into the concept satisfiability problem in $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$. It is enough to consider *concept-circumscribed* knowledge bases of the form $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$ with $\mathsf{CP} = (M, F, V)$ where $\mathcal{A} = \emptyset$ and $F = \emptyset$. The problem of deciding concept satisfiability for this class of circumscribed knowledge bases has been shown to be $\mathsf{NExp}^{\mathsf{NP}}$-hard for $\mathcal{ALC}$ (Bonatti, Lutz, & Wolter 2009). In order to do that, we modify the reduction provided in (Giordano *et al.* 2013b) which shows $\mathsf{NExp}^{\mathsf{NP}}$-hardness for concept satisfiability in $\mathcal{ALCO}$+$\mathbf{T}_{\mathsf{min}}$.

Before going into the details, we assume without loss of generality that each minimized concept occurs in the knowledge base:

Remark. Let $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$ be a circumscribed knowledge base. If $A \in M$ and A does not occur in $(\mathcal{T}, \mathcal{A})$, then for each model $\mathcal{I}$ of $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$: $A^{\mathcal{I}} = \emptyset$.

Given a circumscribed knowledge base $\mathcal{K} = \mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$ (where CP is of the previous form) and a concept description C_0, we define a corresponding $\mathcal{ALC}$+$\mathbf{T}^+$ knowledge base $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ using **two** typicality operators in the following way.

Let M be the set $\{M_1, \ldots, M_q\}$. Similarly as in (Giordano *et al.* 2013b), individual names c and c_{m_i} (one for each $M_i \in M$) and a fresh concept name D are introduced. Each $\mathcal{ALC}$ concept description C is transformed into C^* inductively by introducing D into concept descriptions of the form $\exists r.C_1$, i.e.: $(\exists r.C_1)^* = \exists r.(D \sqcap C_1^*)$ (see (Giordano *et al.* 2013b) for precise details).

Similar as in (Giordano *et al.* 2013b), we start by adding the following GCIs to the TBox $\mathcal{T}'$:

$$D \sqcap C_1^* \sqsubseteq C_2^* \text{ if } C_1 \sqsubseteq C_2 \in \mathcal{T} \quad (6)$$

$$D \sqcap M_i \sqsubseteq \neg \mathbf{T}_1(M_i) \text{ for all } M_i \in M \quad (7)$$

The purpose of using these subsumption statements is to establish a correspondence between the minimized concept names M_i, from the circumscription side, with the underlying concepts $\neg\Box_1\neg M_i$ on the $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ side, such that the minimization of the M_i concepts can be simulated by the minimization of $\neg\Box_1\neg M_i$. The individual names c_{m_i} are introduced to guarantee the existence of typical M_i's in view of assertion (7). The concept D plays the role to distinguish the elements of the domain that are not mapped to those individual names by an interpretation.

Note that if under an interpretation $\mathcal{I}$ an element d is an instance of D and M_i at the same time, then it has to be an instance of $\neg\mathbf{T}_1(M_i)$ and therefore an instance of $\neg\Box_1\neg M_i$ as well. Hence, it is important that whenever d becomes an instance of $\Box_1\neg M_i$ in a preferred interpretation to $\mathcal{I}$, it happens because d becomes an instance of $\neg M_i$ while it is still an instance of D. In order to force this effect during the minimization, the interpretation of the concept D should remain fixed in some way. As pointed out in (Giordano *et al.* 2013b), this seems not to be possible in $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ and that is why the reduction is realized for $\mathcal{ALCO}$+$\mathbf{T}_{\mathsf{min}}$ where nominals are used with that purpose.

In contrast, for $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ this effect on D can be simulated by introducing a second typicality operator $\mathbf{T}_2$, setting $\mathcal{L}_{\mathbf{T}_1} = M, \mathcal{L}_{\mathbf{T}_2} = \{A\}$ and adding the following two assertions to $\mathcal{T}'$:

$$\top \sqsubseteq A \quad (8)$$

$$\neg D \sqsubseteq \neg\mathbf{T}_2(A) \quad (9)$$

where A is a fresh concept name. Note that if an element d becomes a $(\neg D)$-element, it automatically becomes a $(\neg\Box_2\neg A)$-element.

The ABox $\mathcal{A}'$ contains the following assertions:

- $D(c)$,
- for each $M_i \in M$:
  - $(\neg D)(c_{m_i})$,
  - $(\mathbf{T}_1(M_i))(c_{m_i})$,
  - $(\neg M_j)(c_{m_i})$ for all $j \neq i$.

Finally, a concept description C_0' is defined as $D \sqcap C_0^*$.

Lemma 21. *C_0 is satisfiable in* $\mathsf{Circ}_{\mathsf{CP}}(\mathcal{T}, \mathcal{A})$ *iff C_0' is satisfiable w.r.t. $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ in $\mathcal{ALC}$+$\boldsymbol{T}^+_{\mathsf{min}}$.*

Proof. Details of the proof are deferred to the long version of the paper. □

Since the size of $\mathcal{K}'$ is polynomial with respect to the size of $\mathcal{K}$, the application of the previous lemma yields the following result.

Theorem 22. *In $\mathcal{ALC}$+$\boldsymbol{T}^+_{\mathsf{min}}$, concept satisfiability is* $\mathsf{NExp}^{\mathsf{NP}}$*-hard.*

Since concept satisfiability, subsumption and instance checking are polynomially interreducible (see Lemma 10), Theorem 22 yields co-$\mathsf{NExp}^{\mathsf{NP}}$ lower bounds for the subsumption and the instance checking problem.

Corollary 23. *In $\mathcal{ALC}$+$\boldsymbol{T}^+_{\mathsf{min}}$, it is* $\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide concept satisfiability and co-*$\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide subsumption and instance checking.*

Finally, we remark that the translations provided between $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ and *concept-circumscribed* knowledge bases do not depend on the classical constructors of the description logic $\mathcal{ALC}$. Therefore, the same translations can be used for the more expressive description logics $\mathcal{ALCIO}$ and $\mathcal{ALCQO}$. From the complexity results obtained in (Bonatti, Lutz, & Wolter 2009) for circumscription in $\mathcal{ALCIO}$ and $\mathcal{ALCQO}$, we also obtain the following corollary.

Corollary 24. *In $\mathcal{ALCIO}$+$\boldsymbol{T}^+_{\mathsf{min}}$ and $\mathcal{ALCQO}$+$\boldsymbol{T}^+_{\mathsf{min}}$, it is* $\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide concept satisfiability and co-*$\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide subsumption and instance checking.*

Moreover, from the lower bound obtained in (Giordano *et al.* 2013b) for $\mathcal{ALCO}$+$\mathbf{T}_{\mathsf{min}}$, the results also apply for the logics $\mathcal{ALCIO}$+$\mathbf{T}_{\mathsf{min}}$ and $\mathcal{ALCQO}$+$\mathbf{T}_{\mathsf{min}}$.

Corollary 25. *In $\mathcal{ALCIO}$+$\boldsymbol{T}_{\mathsf{min}}$ and $\mathcal{ALCQO}$+$\boldsymbol{T}_{\mathsf{min}}$, it is* $\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide concept satisfiability and co-*$\mathsf{NExp}^{\mathsf{NP}}$*-complete to decide subsumption and instance checking.*

Conclusions

In this paper, we have provided an extension of the non-monotonic description logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$, by adding the possibility to use more than one preference relation over the domain elements. This extension, called $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$, allows to express typicality of a class of elements with respect to different aspects in an "independent" way. Based on this, a class of elements P that is exceptional with respect to a superclass B regarding a specific aspect, could still be not exceptional with respect to different unrelated aspects. The latter permits that defeasible properties from B not conflicting with the exceptionality of P, can be inherited by elements in P. As already observed in the paper, this is not possible in the logic $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$.

In addition, we have introduced translations that show the close relationship between $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ and *concept-circumscribed* knowledge bases in $\mathcal{ALC}$. First, the provided translation from $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ into *concept-circumscribed* knowledge bases is polynomial, in contrast with the exponential translation given in (Giordano *et al.* 2013b) for $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$. Second, the translation presented for the opposite direction shows how to encode circumscribed knowledge base, by using two typicality operators and no nominals.

Using these translations, we were able to determine the complexity of deciding the different reasoning tasks in $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$. We have shown that it is $\mathsf{NExp}^{\mathsf{NP}}$-complete to decide concept satisfiability and co-$\mathsf{NExp}^{\mathsf{NP}}$-complete to decide subsumption and instance checking. Moreover, the same translations can be used for the corresponding extensions of $\mathcal{ALC}$+$\mathbf{T}^+_{\mathsf{min}}$ into more expressive description logics like $\mathcal{ALCIO}$ and $\mathcal{ALCQO}$. The results also apply for extensions of $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ with respect to the underlying description logics, in view of the hardness result shown for $\mathcal{ALCO}$+$\mathbf{T}_{\mathsf{min}}$ in (Giordano *et al.* 2013b).

As possible future work, the exact complexity for reasoning in $\mathcal{ALC}$+$\mathbf{T}_{\mathsf{min}}$ still remains open. It would be interesting to see if it is actually possible to improve the $\mathsf{NExp}^{\mathsf{NP}}$(co-$\mathsf{NExp}^{\mathsf{NP}}$) upper bounds. If that were the case, there is a possibility to identify a corresponding fragment from *concept-circumscribed* knowledge bases with a better complexity than $\mathsf{NExp}^{\mathsf{NP}}$(co-$\mathsf{NExp}^{\mathsf{NP}}$).

As a different aspect, it can be seen that the logic $\mathcal{ALC}$+$\mathbf{T}$ and our proposed extension $\mathcal{ALC}$+$\mathbf{T}^+$ impose syntactic restrictions on the use of the typicality operator. First, it is not possible to use a typicality operator under a role operator. Second, only subsumption statements of the form $\mathbf{T}(A) \sqsubseteq C$ are allowed in the TBox. The latter, seems to come from the fact that $\mathcal{ALC}$+$\mathbf{T}$ is based on the approach to propositional non-monotonic reasoning proposed in (Lehmann & Magidor 1992), where a conditional assertion of the form $A|\!\sim\!C$ is used to express that A's *normally* have property C.

As an example, by lifting these syntactic restrictions, one will be able to express things like:

$$\mathbf{T}(\mathsf{Senior\_Teacher}) \sqsubseteq \mathsf{Excellent\_Teacher}$$
$$\mathbf{T}(\mathsf{Student}) \sqsubseteq \forall \mathsf{attend}.(\mathsf{Class} \sqcap \exists \mathsf{imparted}.\mathbf{T}(\mathsf{Senior\_Teacher}))$$

This allows to relate the typical instances from different classes in a way which is not possible with the current syntax. From a complexity point of view, it is not difficult to observe that the given translations in the paper will also be applicable in this case, without increasing the overall complexity. The reason is that after lifting the mentioned syntactic restrictions, the occurrences of $\mathbf{T}_i(A)$ in an extended concept can still be seen as basic concepts.

Therefore, it would be interesting to study what are the effects of removing these restrictions, with respect to the kind of conclusions that would be obtained from a knowledge base expressed in the resulting non-monotonic logic.

Acknowledgements

I thank my supervisors Gerhard Brewka and Franz Baader for helpful discussions.

References

Baader, F., and Hollunder, B. 1995a. Embedding defaults into terminological knowledge representation formalisms. *J. Autom. Reasoning* 14(1):149–180.

Baader, F., and Hollunder, B. 1995b. Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *J. Autom. Reasoning* 15(1):41–68.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Bonatti, P. A.; Lutz, C.; and Wolter, F. 2009. The complexity of circumscription in dls. *J. Artif. Intell. Res. (JAIR)* 35:717–773.

Britz, K.; Meyer, T.; and Varzinczak, I. J. 2011. Semantic foundation for preferential description logics. In Wang, D., and Reynolds, M., eds., *Australasian Conference on Artificial Intelligence*, volume 7106 of *Lecture Notes in Computer Science*, 491–500. Springer.

Casini, G., and Straccia, U. 2010. Rational closure for defeasible description logics. In Janhunen, T., and Niemelä, I., eds., *JELIA*, volume 6341 of *Lecture Notes in Computer Science*, 77–90. Springer.

Geffner, H., and Pearl, J. 1992. Conditional entailment: Bridging two approaches to default reasoning. *Artif. Intell.* 53(2-3):209–244.

Giordano, L.; Olivetti, N.; Gliozzi, V.; and Pozzato, G. L. 2009. Alc + t: A preferential extension of description logics. *Fundam. Inform.* 96(3):341–372.

Giordano, L.; Gliozzi, V.; Olivetti, N.; and Pozzato, G. L. 2013a. Minimal model semantics and rational closure in description logics. In Eiter, T.; Glimm, B.; Kazakov, Y.; and Krötzsch, M., eds., *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, 168–180. CEUR-WS.org.

Giordano, L.; Gliozzi, V.; Olivetti, N.; and Pozzato, G. L. 2013b. A non-monotonic description logic for reasoning about typicality. *Artif. Intell.* 195:165–202.

Lehmann, D. J., and Magidor, M. 1992. What does a conditional knowledge base entail? *Artif. Intell.* 55(1):1–60.

McCarthy, J. 1980. Circumscription - a form of non-monotonic reasoning. *Artif. Intell.* 13(1-2):27–39.

McCarthy, J. 1986. Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.* 28(1):89–116.

Pearl, J. 1990. System z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In Parikh, R., ed., *TARK*, 121–135. Morgan Kaufmann.

Reiter, R. 1980. A logic for default reasoning. *Artif. Intell.* 13(1-2):81–132.

Schmidt-Schauß, M., and Smolka, G. 1991. Attributive concept descriptions with complements. *Artif. Intell.* 48(1):1–26.

An Argumentation System for Reasoning with Conflict-minimal Paraconsistent $\mathcal{ALC}$

Wenzhao Qiao and **Nico Roos**
Department of Knowledge Engineering, Maastricht University
Bouillonstraat 8-10, 6211 LH Maastricht, The Netherlands
{wenzhao.qiao,roos}@maastrichtuniversity.nl

Abstract

The semantic web is an open and distributed environment in which it is hard to guarantee consistency of knowledge and information. Under the standard two-valued semantics everything is entailed if knowledge and information is inconsistent. The semantics of the paraconsistent logic LP offers a solution. However, if the available knowledge and information is consistent, the set of conclusions entailed under the three-valued semantics of the paraconsistent logic LP is smaller than the set of conclusions entailed under the two-valued semantics. Preferring conflict-minimal three-valued interpretations eliminates this difference.

Preferring conflict-minimal interpretations introduces non-monotonicity. To handle the non-monotonicity, this paper proposes an assumption-based argumentation system. Assumptions needed to close branches of a semantic tableaux form the arguments. Stable extensions of the set of derived arguments correspond to conflict minimal interpretations and conclusions entailed by all conflict-minimal interpretations are supported by arguments in all stable extensions.

Introduction

In the semantic web, the description logics $\mathcal{SHOIN}(D)$ and $\mathcal{SROIQ}(D)$ are the standard for describing ontologies using the TBox, and information using the Abox. Since the semantic web is an open and distributed environment, knowledge and information originating from different sources need not be consistent. In case of inconsistencies, no useful conclusion can be derived when using a standard two-valued semantics. Everything is entailed because the set of two-valued interpretations is empty. Resolving the inconsistencies is often not an option in an open and distributed environment. Therefore, methods that allow us to derive useful conclusions in the presence of inconsistencies are preferred.

One possibility to draw useful conclusions from inconsistent knowledge and information is by focussing on conclusions supported by all maximally consistent subsets. This approach was first proposed by Rescher (1964) and was subsequence worked out further by others (Brewka 1989; Roos 1988; 1992). A simple implementation of this approach focusses on conclusions entailed by the intersection of all maximally consistent subsets. Instead of focussing on the intersection of all maximally consistent subsets, one may also consider a single consistent subset for each conclusion (Poole 1988; Huang, van Harmelen, & ten Teije 2005). For conclusions entailed by all (preferred) maximally consistent subsets of the knowledge and information, a more sophisticated approach is needed. An argumentation system for this more general case has been described by Roos (1992). Since these approaches need to identify consistent subsets of knowledge and information, they are *non-monotonic*.

A second possibility for handling inconsistent knowledge and information is by replacing the standard two-valued semantics by a three-valued semantics such as the semantics of the paraconsistent logic LP (Priest 1989). An important advantage of this paraconsistent logic over the maximally consistent subset approach is that the entailment relation is *monotonic*. A disadvantage is that consistent knowledge and information entail less conclusions when using the three-valued semantics than when using the two-valued semantics. Conflict-minimal interpretations reduce the gap between the sets of conclusions entailed by the two semantics (Priest 1989; 1991). Priest (1991) calls resulting logic: LPm. The conflict-minimal interpretations also makes LPm *non-monotonic* (Priest 1991).

In this paper we present an argumentation system for conclusions entailed by conflict-minimal interpretations of the description logic $\mathcal{ALC}$ (Schmidt-Schauß & Smolka 1991) when using the semantics of the paraconsistent logic LP. We focus on $\mathcal{ALC}$ instead of the more expressive logics $\mathcal{SHOIN}(D)$ and $\mathcal{SROIQ}(D)$ to keep the explanation simple. The described approach can also be applied to more expressive description logics.

The proposed approach starts from a semantic tableaux method for the paraconsistent logic LP described by Bloesch (1993), which has been adapted to $\mathcal{ALC}$. The semantic tableaux is used for deriving the entailed conclusions when using the LP-semantics. If a tableaux cannot be closed, the desired conclusion may still hold in all conflict-minimal interpretations. The open tableaux enables us to identify assumptions about conflict-minimality. These assumptions are used to construct an *assumption-based argumentation system*, which supports conclusions entailed by all conflict minimal interpretations.

The remainder of the paper is organized as follows. First, we describe $\mathcal{ALC}$, a three-valued semantics for $\mathcal{ALC}$ based on the semantics of the paraconsistent logic LP, and a corre-

sponding semantic tableaux method. Second, we describe how a semantic tableaux can be used to determine arguments for conclusions supported by conflict-minimal interpretations. Subsequently, we present the correctness and completeness proof of the described approach. Next we describe some related work. The last section summarizes the results and points out directions of future work.

Paraconsistent $\mathcal{ALC}$

The language of $\mathcal{ALC}$ We first give the standard definitions of the language of $\mathcal{ALC}$. We start with defining the set of concepts $\mathcal{C}$ given the atomic concepts $\mathbf{C}$, the role relations $\mathbf{R}$, the operators for constructing new concepts $\neg$, $\sqcap$ and $\sqcup$, and the quantifiers $\exists$ and $\forall$. Moreover, we introduce to special concepts, $\top$ and $\bot$, which denote *everything* and *nothing*, respectively.

Definition 1 *Let $\mathbf{C}$ be a set of atomic concepts and let $\mathbf{R}$ be a set of atomic roles.*

The set of concepts $\mathcal{C}$ is recursively defined as follows:

- $\mathbf{C} \subseteq \mathcal{C}$*; i.e. atomic concepts are concepts.*
- $\top \in \mathcal{C}$ *and* $\bot \in \mathcal{C}$*.*
- *If $C \in \mathcal{C}$ and $D \in \mathcal{C}$, then $\neg C \in \mathcal{C}$, $C \sqcap D \in \mathcal{C}$ and $C \sqcup D \in \mathcal{C}$.*
- *If $C \in \mathcal{C}$ and $R \in \mathbf{R}$, then $\exists R.C \in \mathcal{C}$ and $\forall R.C \in \mathcal{C}$.*
- *Nothing else belongs to $\mathcal{C}$.*

In the description logic $\mathcal{ALC}$, we have two operators: $\sqsubseteq$ and $=$, for describing a relation between two concepts:

Definition 2 *If $\{C, D\} \subseteq \mathcal{C}$, then we can formulate the following relations (terminological definitions):*

- $C \sqsubseteq D$*; i.e., C is subsumed by D,*
- $C = D$*; i.e., C is equal to D.*

A finite set $\mathcal{T}$ of terminological definitions is called a *TBox*.

In the description logic $\mathcal{ALC}$, we also have an operator ":", for describing that an individual from the set of individual names $\mathbf{N}$ is an instance of a concept, and that a pair of individuals is an instance of a role.

Definition 3 *Let $\{a, b\} \subseteq \mathbf{N}$ be two individuals, let $C \in \mathcal{C}$ be a concept and let $R \in \mathbf{R}$ be a role. Then assertions are defined as:*

- $a : C$
- $(a, b) : R$

A finite set $\mathcal{A}$ of assertions is called an *ABox*.

A *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a tuple consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. In this paper we will denote elements of the TBox and ABox $\mathcal{T} \cup \mathcal{A}$ as propositions.

We define a three-valued semantics for $\mathcal{ALC}$ which is based on the semantics of the paraconsistent logic LP. We do not use the notation $I = (\Delta, \cdot^I)$ that is often used for the semantics of description logics. Instead we will use a notation that is often used for predicate logic because it is more convenient to describe projections and truth-values.

Definition 4 *A three-valued interpretation $I = \langle O, \pi \rangle$ is a couple where O is a non-empty set of objects and π is an interpretation function such that:*

- *for each atomic concept $C \in \mathbf{C}$, $\pi(C) = \langle P, N \rangle$ where $P, N \subseteq O$ are the positive and negative instances of the concept C, respectively, and where $P \cup N = O$,*
- *for each individual $i \in \mathbf{N}$ it holds that $\pi(i) \in O$, and*
- *for each atomic role $R \in \mathbf{R}$ it holds that $\pi(R) \subseteq O \times O$.*

We will use the projections $\pi(C)^+ = P$ and $\pi(C)^- = N$ to denote the positive and negative instances of a concept C, respectively.

We do not consider inconsistencies in roles since we cannot formulate inconsistent roles in $\mathcal{ALC}$. In a more expressive logic, such as $\mathcal{SROIQ}$, roles may become inconsistent, for instance because we can specify disjoint roles.

Using the three-valued interpretations $I = \langle O, \pi \rangle$, we define the interpretations of concepts in $\mathcal{C}$.

Definition 5 *The interpretation of a concept $C \in \mathcal{C}$ is defined by the extended interpretation function π^*.*

- $\pi^*(C) = \pi(C)$ *iff* $C \in \mathbf{C}$
- $\pi^*(\top) = \langle O, X \rangle$*, where* $X \subseteq O$
- $\pi^*(\bot) = \langle X, O \rangle$*, where* $X \subseteq O$
- $\pi^*(\neg C) = \langle \pi^*(C)^-, \pi^*(C)^+ \rangle$
- $\pi^*(C \sqcap D) = \langle \pi^*(C)^+ \cap \pi^*(D)^+, \pi^*(C)^- \cup \pi^*(D)^- \rangle$
- $\pi^*(C \sqcup D) = \langle \pi^*(C)^+ \cup \pi^*(D)^+, \pi^*(C)^- \cap \pi^*(D)^- \rangle$
- $\pi^*(\exists R.C) =$
  $\langle \{x \in O \mid \exists y \in O, (x, y) \in \pi(R)$ *and* $y \in \pi(C)^+\},$
  $\{x \in O \mid \forall y \in O, (x, y) \in \pi(R)$ *implies* $y \in \pi(C)^-\} \rangle$
- $\pi^*(\forall R.C) =$
  $\langle \{x \in O \mid \forall y \in O, (x, y) \in \pi(R)$ *implies* $y \in \pi(C)^+\},$
  $\{x \in O \mid \exists y \in O, (x, y) \in \pi(R)$ *and* $y \in \pi(C)^-\} \rangle$

Note that we allow inconsistencies in the concepts $\top$ and $\bot$. There may not exist a tree-valued interpretation for a knowledge-base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if we require that $X = \varnothing$. Consider for instance: $a : C$, $a : D$ and $C \sqcap D \sqsubseteq \bot$.

We also use the extended interpretation function π^* to define the truth values of the propositions: $C \sqsubseteq D$, $a : C$ and $(a, b) : R$. The truth values of the three-valued semantics are defined using sets of the "classical" truth values: t and f. We use three sets in the LP-semantics: $\{t\}$, $\{f\}$ and $\{t, f\}$, which correspond to TRUE, FALSE and CONFLICT.

Definition 6 *Let $\{a, b\} \subseteq \mathbf{N}$ be two individuals, let $C \in \mathcal{C}$ be a concept and let $R \in \mathbf{R}$ be a role. Then an interpretation $I = \langle O, \pi \rangle$ of propositions is defined as:*

- $t \in \pi^*(a : C)$ *iff* $\pi^*(a) \in \pi^*(C)^+$
- $f \in \pi^*(a : C)$ *iff* $\pi^*(a) \in \pi^*(C)^-$
- $t \in \pi^*(C \sqsubseteq D)$ *iff* $\pi^*(C)^+ \subseteq \pi^*(D)^+$*, and* $\pi^*(D)^- \subseteq \pi^*(C)^-$
- $f \in \pi^*(C \sqsubseteq D)$ *iff* $t \notin \pi^*(C \sqsubseteq D)$
- $t \in \pi^*(C = D)$ *iff* $\pi^*(C)^+ = \pi^*(D)^+$ *and* $\pi^*(D)^- = \pi^*(C)^-$
- $f \in \pi^*(C = D)$ *iff* $t \notin \pi^*(C = D)$
- $t \in \pi^*((a, b) : R)$ *iff* $(\pi^*(a), \pi^*(b)) \in \pi(R)$
- $f \in \pi^*((a, b) : R)$ *iff* $(\pi^*(a), \pi^*(b)) \notin \pi(R)$

The interpretation of the subsumption relation given above was proposed by Patel-Schneider (1989) for their four-valued semantics. Patel-Schneider's interpretation of the subsumption relation does not correspond to the material implication $\forall x[C(x) \rightarrow D(x)]$ in first-order logic. The latter is equivalent to $\forall x[\neg C(x) \vee D(x)]$ under the two-valued semantics, which corresponds to: "for every $o \in O$, $o \in \pi^*(C)^-$ or $o \in \pi^*(D)^+$" under the three-valued semantics. No conclusion can be drawn from $a : C$ and $C \sqsubseteq D$ under the three-valued semantics since there always exists an interpretation such that $\pi(a : C) = \{t, f\}$.

The entailment relation can be defined using the interpretations of propositions.

Definition 7 *Let $I = \langle O, \pi \rangle$ be an interpretation, let φ be a proposition, and let Σ be a set of propositions. The the entailment relation is defined as:*

- *$I \models \varphi$ iff $t \in \pi^*(\varphi)$*
- *$I \models \Sigma$ iff $t \in \pi^*(\sigma)$ for every $\sigma \in \Sigma$.*
- *$\Sigma \models \varphi$ iff $I \models \Sigma$ implies $I \models \varphi$ for each interpretation I*

Semantic tableaux We use a semantic tableaux method that is based on the semantic tableaux method for LP described by Bloesch (1993). This tableaux method will enable us to identify the assumptions underlying relevant conflict minimal interpretations.

Bloesch proposes to label every proposition in the tableaux with either the labels $\mathbb{T}$ (at least true), $\mathbb{F}$ (at least false), or their complements $\overline{\mathbb{T}}$ and $\overline{\mathbb{F}}$, respectively. So, $\mathbb{T}\varphi$ corresponds to $t \in \pi(\varphi)$, $\overline{\mathbb{T}}\varphi$ corresponds to $t \notin \pi(\varphi)$, $\mathbb{F}\varphi$ corresponds to $f \in \pi(\varphi)$, and $\overline{\mathbb{F}}\varphi$ corresponds to $f \notin \pi(\varphi)$.

Although we do not need it in the semantic tableaux, we also make use of $\mathbb{C}\varphi$ and $\overline{\mathbb{C}}\varphi$, which corresponds semantically with $\{t, f\} = \pi(\varphi)$ and $\{t, f\} \neq \pi(\varphi)$, respectively. So, $\mathbb{C}\varphi$ is equivalent to: '$\mathbb{T}\varphi$ *and* $\mathbb{F}\varphi$', and $\overline{\mathbb{C}}\varphi$ is equivalent to: '$\overline{\mathbb{T}}\varphi$ *or* $\overline{\mathbb{F}}\varphi$'.

To prove that $\Sigma \models \varphi$ using Bloesch's tableaux method (Bloesch 1993), we have to show that a tableaux with root $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \overline{\mathbb{T}}\varphi$ closes. The tableaux closes if every branch has a node in which for some proposition α the node contains: "$\mathbb{T}\alpha$ *and* $\overline{\mathbb{T}}\alpha$", or "$\mathbb{F}\alpha$ *and* $\overline{\mathbb{F}}\alpha$", or "$\overline{\mathbb{T}}\alpha$ *and* $\overline{\mathbb{F}}\alpha$".

Based on Bloesch's semantic tableaux method for LP, the following tableaux rules have been formulated. The soundness and completeness of the set of rules are easy to prove.

$$\frac{\mathbb{T}\, a : \neg C}{\mathbb{F}\, a : C} \qquad \frac{\overline{\mathbb{T}}\, a : \neg C}{\overline{\mathbb{F}}\, a : C} \qquad \frac{\mathbb{F}\, a : \neg C}{\mathbb{T}\, a : C} \qquad \frac{\overline{\mathbb{F}}\, a : \neg C}{\overline{\mathbb{T}}\, a : C}$$

$$\frac{\mathbb{T}\, a : C \sqcap D}{\mathbb{T}\, a : C, \mathbb{T}\, a : D} \qquad \frac{\overline{\mathbb{T}}\, a : C \sqcap D}{\overline{\mathbb{T}}\, a : C \mid \overline{\mathbb{T}}\, a : D}$$

$$\frac{\mathbb{F}\, a : C \sqcap D}{\mathbb{F}\, a : C \mid \mathbb{F}\, a : D} \qquad \frac{\overline{\mathbb{F}}\, a : C \sqcap D}{\overline{\mathbb{F}}\, a : C, \overline{\mathbb{F}}\, a : D}$$

$$\frac{\mathbb{T}\, a : C \sqcup D}{\mathbb{T}\, a : C \mid \mathbb{T}\, a : D} \qquad \frac{\overline{\mathbb{T}}\, a : C \sqcup D}{\overline{\mathbb{T}}\, a : C, \overline{\mathbb{T}}\, a : D}$$

$$\frac{\mathbb{F}\, a : C \sqcup D}{\mathbb{F}\, a : C, \mathbb{F}\, a : D} \qquad \frac{\overline{\mathbb{F}}\, a : C \sqcup D}{\overline{\mathbb{F}}\, a : C \mid \overline{\mathbb{F}}\, a : D}$$

$$\frac{\mathbb{T}\, a : \exists r.C}{\mathbb{T}\, (a, x) : r, \mathbb{T}\, x : C} \qquad \frac{\overline{\mathbb{T}}\, a : \exists r.C, \mathbb{T}\, (a, b) : r}{\overline{\mathbb{T}}\, b : C}$$

$$\frac{\mathbb{F}\, a : \exists r.C, \mathbb{T}\, (a, b) : r}{\mathbb{F}\, b : C} \qquad \frac{\overline{\mathbb{F}}\, a : \exists r.C}{\mathbb{T}\, (a, x) : r, \overline{\mathbb{F}}\, x : C}$$

$$\frac{\mathbb{T}\, a : \forall r.C, \mathbb{T}\, (a, b) : r}{\mathbb{T}\, b : C} \qquad \frac{\overline{\mathbb{T}}\, a : \forall r.C}{\mathbb{T}\, (a, x) : r, \overline{\mathbb{T}}\, x : C}$$

$$\frac{\mathbb{F}\, a : \forall r.C}{\mathbb{T}\, (a, b) : r, \mathbb{F}\, b : C} \qquad \frac{\overline{\mathbb{F}}\, a : \forall r.C, \mathbb{T}\, (a, b) : r}{\overline{\mathbb{F}}\, b : C}$$

The individual a in the following tableaux rules for the subsumption relation must be an existing individual name, while the individual x must be a new individual name.

$$\frac{\mathbb{T}\, C \sqsubseteq D}{\overline{\mathbb{T}}\, a : C \mid \mathbb{T}\, a : D} \qquad \frac{\mathbb{T}\, C \sqsubseteq D}{\overline{\mathbb{F}}\, a : D \mid \mathbb{F}\, a : C}$$

$$\frac{\overline{\mathbb{T}}\, C \sqsubseteq D}{\mathbb{T}\, x : C, \overline{\mathbb{T}}\, x : D \mid \mathbb{F}\, x : D, \overline{\mathbb{F}}\, x : C}$$

$$\frac{\mathbb{T}\, C = D}{\mathbb{T}\, C \sqsubseteq D, \mathbb{T}\, D \sqsubseteq C} \qquad \frac{\overline{\mathbb{T}}\, C = D}{\overline{\mathbb{T}}\, C \sqsubseteq D \mid \overline{\mathbb{T}}\, D \sqsubseteq C}$$

An important issue is guaranteeing that the constructed semantic tableaux is always finite. The *blocking* method described by (Buchheit, Donini, & Schaerf 1993; Baader, Buchheit, & Hollander 1996) is used to guarantee the construction of a finite tableaux. A rule that is *blocked*, may not be not be used in the construction of the tableaux.

Definition 8 *Let Γ be a node of the tableau, and let x and y be two individual names. Moreover, let $\Gamma(x) = \{\mathbb{L}x : C \mid \mathbb{L}x : C \in \Gamma\}$.*

- *$x <_r y$ if $(x, y) : R \in \Gamma$ for some $R \in \mathbf{R}$.*
- *y is blocked if there is an individual name x such that: $x <_r^+ y$ and $\Gamma(y) \subseteq \Gamma(x)$, or $x <_r y$ and x is blocked.*

Conflict Minimal Interpretations A price that we pay for changing to the three-valued LP-semantics in order to handle inconsistencies is a reduction in the set of entailed conclusions, even if the knowledge and information is consistent.

Example 1 *The set of propositions $\Sigma = \{a : \neg C, a : C \sqcup D\}$ does not entail $a : D$ because there exists an interpretation $I = \langle O, \pi \rangle$ for Σ such that $\pi(a : C) = \{t, f\}$ and $\pi(a : D) = \{f\}$.*

Priest (1989; 1991) points out that more useful conclusions can be derived from the paraconsistent logic LP if we would prefer conflict-minimal interpretations. The resulting logic is LPm. Here we follow the same approach. First, we define a conflict ordering on interpretations.

Definition 9 *Let $\mathbf{C}$ be a set of atomic concepts, let $\mathbf{N}$ be a set of individual names, and let I_1 and I_2 be two three-valued interpretations.*

The interpretation I_1 contains less conflicts than the interpretation I_2, denoted by $I_1 <_c I_2$, iff:

$$\{a : C \mid a \in \mathbf{N}, C \in \mathbf{C}, \pi_1(a : C) = \{t, f\}\} \subset \{a : C \mid a \in \mathbf{N}, C \in \mathbf{C}, \pi_2(a : C) = \{t, f\}\}$$

The following example gives an illustration of a conflict ordering for the set of propositions of Example 1.

Example 2 *Let $\Sigma = \{a : \neg C, a : C \sqcup D\}$ be a set of propositions and let I_1, I_2, I_3, I_4 and I_5 be five interpretations such that:*

- $\pi_1^*(a:C) = \{f\}$, $\pi_1^*(a:D) = \{t\}$,
- $\pi_2^*(a:C) = \{f\}$, $\pi_2^*(a:D) = \{t, f\}$.
- $\pi_3^*(a:C) = \{t, f\}$, $\pi_3^*(a:D) = \{t\}$,
- $\pi_4^*(a:C) = \{t, f\}$, $\pi_4^*(a:D) = \{f\}$,
- $\pi_5^*(a:C) = \{t, f\}$, $\pi_5^*(a:D) = \{t, f\}$.

Then $I_1 <_c I_2$, $I_1 <_c I_3$, $I_1 <_c I_4$, $I_1 <_c I_5$, $I_2 <_c I_5$, $I_3 <_c I_5$ *and* $I_4 <_c I_5$.

Using the conflict ordering, we define the conflict minimal interpretations.

Definition 10 *Let I_1 be a three-valued interpretation and let Σ be a set of propositions.*

I_1 is a conflict minimal interpretation of Σ, denoted by $I_1 \models_{<_c} \Sigma$, iff $I_1 \models \Sigma$ and for no interpretation I_2 such that $I_2 <_c I_1$, $I_2 \models \Sigma$ holds.

In Example 2, I_1 is the only conflict-minimal interpretation.

The conflict-minimal entailment of a proposition by a set of propositions can now be defined.

Definition 11 *Let $\Sigma = (\mathcal{T} \cup \mathcal{A})$ be a set of propositions and let φ be a proposition.*

Σ entails conflict-minimally the proposition φ, denoted by $\Sigma \models_{<_c} \varphi$, iff for every interpretation I, if $I \models_{<_c} \Sigma$, then $I \models \varphi$.

The conflict-minimal interpretations in Example 2 entail the conclusion $a : D$.

The subsumption relation The conflict-minimal interpretations enables us to use an interpretation of the subsumption relation based on the material implication.

- For every $o \in O$, $o \in \pi^*(C)^-$ or $o \in \pi^*(D)^+$

This semantics of the subsumption relation resolves a problem with the semantics of Patel-Schneider (1989). Under Patel-Schneider's semantics, $\{a : C, a : \neg C, C \sqsubseteq D\}$ entails $a : D$. This entailment is undesirable if information about $a : C$ is contradictory.

The tableaux rules of the new interpretation are:

$$\frac{\mathbb{T}\, C \sqsubseteq D}{\mathbb{F}\, a : C \mid \mathbb{T}\, a : D} \qquad \frac{\overline{\mathbb{T}}\, C \sqsubseteq D}{\mathbb{T}\, a : C, \mathbb{F}\, a : D}$$

Arguments for conclusions supported by conflict minimal interpretations

The conflict-minimal interpretations of a knowledge base entail more useful conclusions. Unfortunately, focusing on conclusions supported by conflict-minimal interpretations makes the reasoning process *non-monotonic*. Adding the assertion $a : \neg D$ to the set of propositions in Example 2 eliminates interpretations I_1 and I_3, which includes the only conflict-minimal interpretation I_1. The interpretations I_2 and I_4 are the new conflict-minimal interpretations. Unlike the original conflict-minimal interpretation I_1, the new conflict-minimal interpretations I_2 and I_4 do not entail $a : D$.

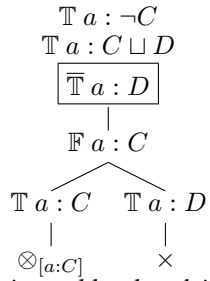
Deriving conclusions supported by the conflict-minimal interpretations is problematic because of the non-monotonicity. The modern way to deal with non-monotonicity is by giving an argument supporting a conclusion, and subsequently verifying whether there are no counter-arguments (Dung 1995). Here we will follow this argumentation-based approach.

We propose an approach for deriving arguments that uses the semantic tableaux method for our paraconsistent logic as a starting point. The approach is based on the observation that an interpretation satisfying the root of a semantic tableaux will also satisfy one of the leafs. Now suppose that the only leafs of a tableaux that are not closed; i.e., leaf in which we do not have "$\mathbb{T}\alpha$ *and* $\overline{\mathbb{T}}\alpha$" or "$\mathbb{F}\alpha$ *and* $\overline{\mathbb{F}}\alpha$" or "$\overline{\mathbb{T}}\alpha$ *and* $\overline{\mathbb{F}}\alpha$", are leafs in which "$\mathbb{T}\alpha$ *and* $\mathbb{F}\alpha$" holds for some proposition α. So, in every open branch of the tableaux, $\mathbb{C}\alpha$ holds for some proposition α. If we can assume that there are no conflicts w.r.t. each proposition α in the conflict-minimal interpretations, then we can also close the open branches. The set of assumptions $\overline{\mathbb{C}}\alpha$, equivalent to "$\overline{\mathbb{T}}\alpha$ *or* $\overline{\mathbb{F}}\alpha$", that we need to close the open branches, will be used as the argument for the conclusion supported by the semantic tableaux.

An advantage of the proposed approach is that there is no need to consider arguments if a conclusion already holds without considering conflict-minimal interpretations.

A branch that can be closed *assuming* that the conflict-minimal interpretations contain **no** conflicts with respect to the proposition α; i.e., assuming $\overline{\mathbb{C}}\alpha$, will be called a *weakly closed* branch. We will call a tableaux *weakly closed* if some branches are weakly closed and all other branches are closed. If we can (weakly) close a tableaux for $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in (\mathcal{T} \cup \mathcal{A})\} \cup \overline{\mathbb{T}}\varphi$, we consider the set of assumptions $\overline{\mathbb{C}}\alpha$ needed to weakly close the tableaux, to be the argument supporting $\Sigma \models_{\leq_c} \varphi$. Example 3 gives an illustration.

Example 3 *Let $\Sigma = \{a : \neg C, a : C \sqcup D\}$ be a set of propositions. To verify whether $a : D$ holds, we may construct the following tableaux:*



Only the left branch is weakly closed *in this tableau. We assume that the assertion $a : C$ will* not *be assigned* CONFLICT *in any conflict-minimal interpretation. That is, we assume that $\overline{\mathbb{C}}\, a : C$ holds.*

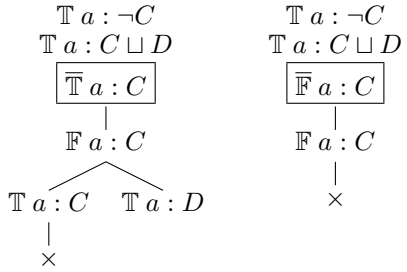
In the following definition of an argument, we consider arguments for $\mathbb{T}\varphi$ and $\mathbb{F}\varphi$.

Definition 12 *Let Σ be set of propositions and let φ a proposition. Moreover, let $\mathscr{T}$ be a (weakly) closed semantic tableaux with root $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \overline{\mathbb{L}}\varphi$ and $\mathbb{L} \in \{\mathbb{T}, \mathbb{F}\}$. Finally, let $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ be the set of assumptions on which the closures of weakly closed branches are based.*

Then $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{L}\varphi)$ is an argument.

The next step is to verify whether the assumptions: $\overline{\mathbb{C}}\alpha_i$ are valid. If one of the assumptions does not hold, we have a *counter-argument* for our argument supporting $\Sigma \models_{\leq_c} \varphi$. To verify the correctness of an assumption, we add the assumption to Σ. Since an assumption $\overline{\mathbb{C}}\alpha$ is equivalent to: "$\overline{\mathbb{T}}\alpha$ *or* $\overline{\mathbb{F}}\alpha$", we can consider $\overline{\mathbb{T}}\alpha$ and $\overline{\mathbb{F}}\alpha$ separately. Example 4 gives an illustration for the assumption $\overline{\mathbb{C}}\ a : C$ used in Example 3.

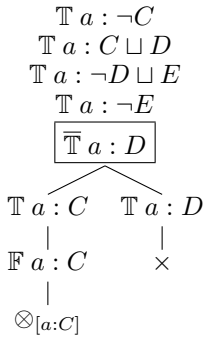
Example 4 *Let $\Sigma = \{a : \neg C, a : C \sqcup D\}$ be a set of propositions. To verify whether the assumption $\overline{\mathbb{C}}\ a : C$ holds in every conflict minimal interpretation, we may construct a tableaux assuming $\overline{\mathbb{T}}\ a : C$ and a tableaux assuming $\overline{\mathbb{F}}\ a : C$:*



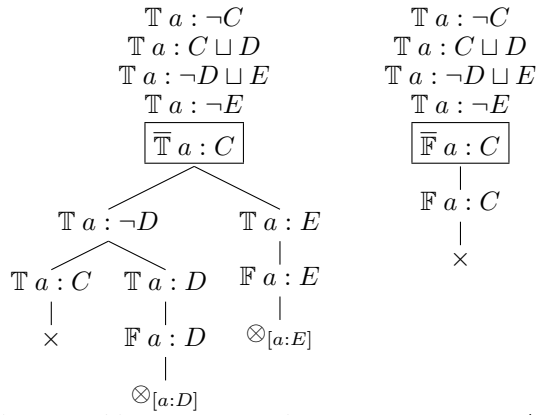
The right branch of the first tableaux cannot be closed. Therefore, the assumption $\overline{\mathbb{T}}\ a : C$ is valid, implying that the assumption $\overline{\mathbb{C}}\ a : C$ is also valid. Hence, there exists no *counter-argument.*

Since the validity of assumptions must be verified with respect to conflict-minimal interpretations, assumptions may also be used in the counter-arguments. This implies that we may have to verify whether there exists a counter-argument for a counter-argument. Example 5 gives an illustration.

Example 5 *Let $\Sigma = \{a : \neg C, a : C \sqcup D, a : \neg D \sqcup E, a : \neg E\}$ be a set of propositions. To verify whether $a : D$ holds, we may construct the following tableaux:*



This weakly closed tableaux implies the argument $A_0 = (\{\overline{\mathbb{C}}\ a : C\}, \mathbb{T}\ a : D)$. Next, we have to verify whether there exists a counter-argument for A_0. To verify the existence of a counter-argument, we construct two tableaux, one for $\overline{\mathbb{T}}\ a : C$ and one for $\overline{\mathbb{F}}\ a : C$. As we can see below, both tableaux are (weakly)-closed, and therefore form the counter-argument $A_1 = (\{\overline{\mathbb{C}}\ a : D, \overline{\mathbb{C}}\ a : E\}, \mathbb{C}\ a : C)$. We say that the argument A_1 attacks *the argument A_0 because the former is a counter-argument of the latter.*



The two tableaux forming the counter-argument A_1 are closed under the assumptions: $\overline{\mathbb{C}}\ a : D$ and $\overline{\mathbb{C}}\ a : E$. So, A_1 is a valid argument if there exists no valid counter-argument for $\mathbb{C}\ a : D$, and no counter-argument for $\mathbb{C}\ a : E$.

Argument A_1 is equivalent to two other arguments, namely: $A_2 = (\{\overline{\mathbb{C}}\ a : C, \overline{\mathbb{C}}\ a : E\}, \mathbb{C}\ a : D)$ and $A_3 = (\{\overline{\mathbb{C}}\ a : C, \overline{\mathbb{C}}\ a : D\}, \mathbb{C}\ a : E)$. A proof of the equivalence will be given in the next section, Proposition 1.

The arguments A_2 and A_3 implied by A_1 are both counter-arguments of A_1. Moreover, A_1 is a counter-argument of A_2 and A_3, and A_2 and A_3 are counter-arguments of each other. No other counter-arguments can be identified in this example. Figure 1 show all the arguments and the attack relation, denoted by the arrows, between the arguments.

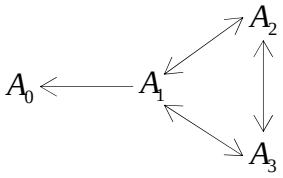


Figure 1: The attack relations between the arguments of Example 5.

We will now formally define the arguments and the attack relations that we can derive from the constructed semantic tableaux.

Definition 13 *Let Σ be set of propositions and let $\overline{\mathbb{C}}\alpha =$"$\overline{\mathbb{T}}\alpha$ or $\overline{\mathbb{F}}\alpha$" be an assumption in the argument A. Moreover, let $\mathscr{T}_1$ be a (weakly) closed semantic tableaux with root $\Gamma_1 = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \overline{\mathbb{T}}\alpha$ and let $\mathscr{T}_2$ be a (weakly) closed semantic tableaux with root $\Gamma_2 = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \overline{\mathbb{F}}\alpha$. Finally, let $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ be the set of assumptions on which the weakly closed branches in the tableaux $\mathscr{T}_1$ or the tableaux $\mathscr{T}_2$ are based.*

Then $A' = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\alpha)$ is a counter-argument of the argument A. We say that the argument A' attacks *the argument A, denoted by: $A' \longrightarrow A$.*

The form of argumentation that we have here is called assumption-based argumentation (ABA), which has been developed since the end of the 1980's (Bondarenko *et al.*

1997; Bondarenko, Toni, & Kowalski 1993; Dung, Kowalski, & Toni 2009; Gaertner & Toni 2007; Roos 1988; 1992).

Example 5 shows that an argument can be counter-argument of an argument and vice versa; e.g., arguments A_2 and A_3. This raises the question which arguments are valid. Argumentation theory and especially the argumentation framework (AF) introduced by Dung (1995) provides an answer.

Arguments are viewed in an argumentation framework as atoms over which an attack relation is defined. Figure 1 shows the arguments and the attack relations between the arguments forming the argumentation framework of Example 5. The formal specification of an argumentation framework is given by the next definition.

Definition 14 *An argumentation framework is a couple* $AF = (\mathscr{A}, \longrightarrow)$ *where* $\mathscr{A}$ *is a finite set of arguments and* $\longrightarrow \subseteq \mathscr{A} \times \mathscr{A}$ *is an attack relation over the arguments.*

For convenience, we extend the attack relation $\longrightarrow$ to sets of arguments.

Definition 15 *Let* $A \in \mathscr{A}$ *be an argument and let* $\mathscr{S}, \mathscr{P} \subseteq \mathscr{A}$ *be two sets of arguments. We define:*

- $\mathscr{S} \longrightarrow A$ *iff for some* $B \in \mathscr{S}$, $B \longrightarrow A$.
- $A \longrightarrow \mathscr{S}$ *iff for some* $B \in \mathscr{S}$, $A \longrightarrow B$.
- $\mathscr{S} \longrightarrow \mathscr{P}$ *iff for some* $B \in \mathscr{S}$ *and* $C \in \mathscr{P}$, $B \longrightarrow C$.

Dung (1995) describes different argumentation semantics for an argumentation framework in terms of sets of acceptable arguments. These semantics are based on the idea of selecting a coherent subset $\mathscr{E}$ of the set of arguments $\mathscr{A}$ of the argumentation framework $AF = (\mathscr{A}, \longrightarrow)$. Such a set of arguments $\mathscr{E}$ is called an *argument extension*. The arguments of an argument extension support propositions that give a coherent description of what might hold in the world. Clearly, a basic requirement of an argument extension is being *conflict-free*; i.e., no argument in an argument extension attacks another argument in the argument extension. Besides being conflict-free, an argument extension should defend itself against attacking arguments by attacking the attacker.

Definition 16 *Let* $AF = (\mathscr{A}, \longrightarrow)$ *be an argumentation framework and let* $\mathscr{S} \subseteq \mathscr{A}$ *be a set of arguments.*

- $\mathscr{S}$ *is* conflict-free *iff* $\mathscr{S} \not\longrightarrow \mathscr{S}$.
- $\mathscr{S}$ *defends an argument* $A \in \mathscr{A}$ *iff for every argument* $B \in \mathscr{A}$ *such that* $B \longrightarrow A$, $\mathscr{S} \longrightarrow B$.

Not every conflict-free set of arguments that defends itself, is considered to be an argument extension. Several additional requirements have been formulated by Dung (1995), resulting in three different semantics: the *stable*, the *preferred* and the *grounded* semantics.

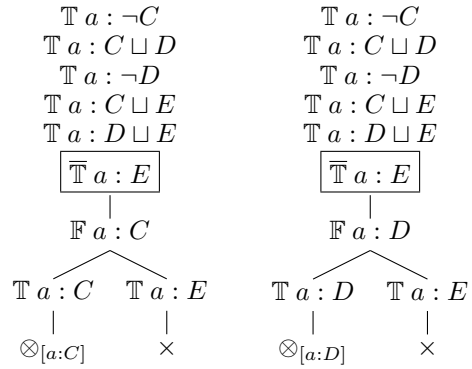
Definition 17 *Let* $AF = (\mathscr{A}, \longrightarrow)$ *be an argumentation framework and let* $\mathscr{E} \subseteq \mathscr{A}$.

- $\mathscr{E}$ *is a* stable extension *iff* $\mathscr{E}$ *is conflict-free,and for every argument* $A \in (\mathscr{A} \setminus \mathscr{E})$, $\mathscr{E} \longrightarrow A$; *i.e.,* $\mathscr{E}$ *defends itself against every possible attack by arguments in* $\mathscr{A} \setminus \mathscr{E}$.
- $\mathscr{E}$ *is a* preferred extension *iff* $\mathscr{E}$ *is maximal (w.r.t.* $\subseteq$*) set of arguments that (1) is conflict-free,and (2)* $\mathscr{E}$ *defends every argument* $A \in \mathscr{E}$.
- $\mathscr{E}$ *is a* grounded extension *iff* $\mathscr{E}$ *is the minimal (w.r.t.* $\subseteq$*) set of arguments that (1) is conflict-free, (2) defends every argument* $A \in \mathscr{E}$*, and (3) contains all arguments in* $\mathscr{A}$ *it defends.*

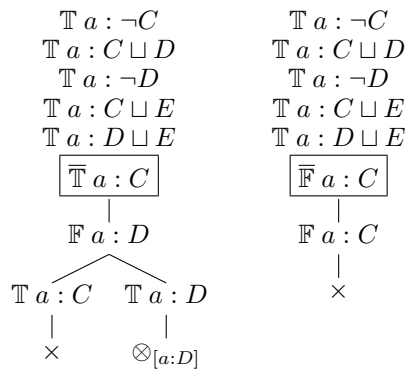
We are interested in stable semantics. We will show in the next section that stable extensions correspond to conflict-minimal interpretations. More specifically, we will prove that a conclusion supported by an argument in every stable extension, is entailed by every conflict-minimal interpretation, and vice versa.

Is it possible that a conclusion is supported by a different argument in every stable extension? The answer is Yes, as is illustrated by Example 6. In this example we have two arguments supporting the conclusion $a : E$, namely A_0 and A_1. As can be seen in Figure 2, there are two stable extensions of the argumentation framework. One extension contains the argument A_0 and the other contains the argument A_1. So, in every extension there is an argument supporting the conclusion $a : E$. Hence, $\Sigma \models_{\leq_c} a : E$.

Example 6 *Let* $\Sigma = \{a : \neg C, a : C \sqcup D, a : \neg D, a : C \sqcup E, a : D \sqcup E\}$ *be a set of propositions. The following two tableaux imply the two arguments* $A_0 = (\{\overline{\mathbb{C}}\ a : C\}, \mathbb{T}\ a : E)$ *and* $A_1 = (\{\overline{\mathbb{C}}\ a : D\}, \mathbb{T}\ a : E)$*, both supporting the conclusion* $a : E$*:*



The assumption $\overline{\mathbb{C}}\ a : C$ *in argument* A_0 *makes it possible to determine a counter-argument* $A_2 = (\{\overline{\mathbb{C}}\ a : D\}, \mathbb{C}\ a : C)$ *using of the following two tableaux:*



According to Proposition 1, A_2 *implies the counter-argument* $A_3 = (\{\overline{\mathbb{C}}\ a : C\}, \mathbb{C}\ a : D)$ *of* A_1 *and* A_2. A_2

is also a counter-argument of A_3. Figure 2 shows the attack relations between the arguments A_0, A_1, A_2 and A_3.

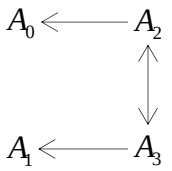


Figure 2: The attack relations between the arguments of Example 6.

Example 7 gives an illustration of the semantic interpretations of Example 6. The example shows two conflict-minimal interpretations. These conflict-minimal interpretations correspond with the two *stable extensions*. Interpretation I_1 entails $a : E$ because I_1 must entail $a : C \sqcup E$ and I_1 does not entail $a : C$, and interpretation I_2 entails $a : E$ because I_2 must entail $a : D \sqcup E$ and I_2 does not entail $a : D$.

Example 7 *Let $\Sigma = \{a : \neg C, a : C \sqcup D, a : \neg D, a : C \sqcup E, a : D \sqcup E\}$ be a set of propositions. There are two conflict-minimal interpretations containing the following interpretation functions:*

- $\pi_1(a : C) = \{f\}$, $\pi_1(a : D) = \{t, f\}$, $\pi_1(a : E) = \{t\}$.
- $\pi_2(a : C) = \{t, f\}$, $\pi_2(a : D) = \{f\}$, $\pi_2(a : E) = \{t\}$.

In both interpretations $a : E$ is entailed.

Correctness and completeness proofs

In this section we investigate whether the proposed approach is correct. That is whether a proposition supported by an argument in every stable extension is entailed by every conflict-minimal interpretation. Moreover, we investigate whether the approach is complete. That is, whether a proposition entailed by every conflict-minimal interpretation is supported by an argument in every stable extension.

In the following theorem we will use the notion of "a complete set of arguments relevant to φ". This set of arguments $\mathscr{A}$ consists of all argument A supporting φ, all possible counter-arguments, all possible counter arguments of the counter-arguments, etc.

Definition 18 *A complete set of arguments $\mathscr{A}$ relevant to φ satisfies the following requirements:*

- $\{A \mid A \text{ supports } \varphi\} \subseteq \mathscr{A}$.
- *If $A \in \mathscr{A}$ and B is a counter-argument of A that we can derive, then $B \in \mathscr{A}$ and $(B, A) \in \longrightarrow$.*

Theorem 1 (correctness and completeness) *Let Σ be a set of propositions and let φ be a proposition. Moreover, let $\mathscr{A}$ be a complete set of arguments relevant to φ, let $\longrightarrow \subseteq \mathscr{A} \times \mathscr{A}$ be the attack relation determined by $\mathscr{A}$, and let $(\mathscr{A}, \longrightarrow)$ be the argumentation framework. Finally, let $\mathscr{E}_1, \ldots, \mathscr{E}_k$ be all stable extensions of the argumentation framework $(\mathscr{A}, \longrightarrow)$.*

Σ entails the proposition φ using the conflict-minimal three-valued semantics; i.e., $\Sigma \models_{\leq_c} \varphi$, iff φ is supported by an argument in every stable extension $\mathscr{E}_i$ of $(\mathscr{A}, \longrightarrow)$.

To prove Theorem 1, we need the following lemmas. In these lemmas we will use the following notations: We will use $I \models \mathbb{T}\alpha$ to denote that $t \in I(\alpha)$ ($I \models \alpha$), and $I \models \mathbb{F}\alpha$ to denote that $f \in I(\alpha)$. Moreover, we will use $\Sigma \models \mathbb{T}\alpha$ and $\Sigma \models \mathbb{F}\alpha$ to denote that $\mathbb{T}\alpha$ and $\mathbb{F}\alpha$, respectively, hold in all three-valued interpretations of Σ.

The first lemma proves the correctness of the arguments in $\mathscr{A}$.

Lemma 1 (correctness of arguments) *Let Σ be a set of propositions and let φ be a proposition. Moreover, let $\mathbb{L}$ be either the label $\mathbb{T}$ or $\mathbb{F}$.*

If a semantic tableaux with root $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ is weakly closed, and if $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ is the set of weak closure assumptions implied by all the weakly closed leafs of the tableaux, then

$$\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \models \mathbb{L}\varphi$$

Proof Suppose that $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \not\models \mathbb{L}\varphi$. Then there must be an interpretation I satisfying $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\}$ but not $\mathbb{L}\varphi$. So, $I \models \{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$. We can create a tableaux for $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ by adding the assumptions $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ to every node in the original tableaux with root Γ. Let $\Gamma^* = \{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ be the root of the resulting tableaux. Since $I \models \Gamma^*$, there must be a leaf Λ^* of the new tableaux and $I \models \Lambda^*$. The corresponding leaf Λ in the original tableaux with root Γ is either strongly or weakly closed.

- If Λ is strongly closed, then so is Λ^* and we have a contradiction.
- If Λ is weakly closed, then the weak closure implies one of the assumptions $\overline{\mathbb{C}}\alpha_i$ because $\{\mathbb{T}\alpha_i, \mathbb{F}\alpha_i\} \subseteq \Lambda$. Therefore, $\{\mathbb{T}\alpha_i, \mathbb{F}\alpha_i\} \subseteq \Lambda^*$. Since $\{\mathbb{T}\alpha_i, \mathbb{F}\alpha_i\}$ implies $\mathbb{C}\alpha_i$ and since $\overline{\mathbb{C}}\alpha_i \in \Lambda^*$, $I \not\models \Lambda^*$ The latter contradicts with $I \models \Lambda^*$.

Hence, the lemma holds. □

The above lemma implies that the assumptions of an argument $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{L}\varphi)$ together with Σ entail the conclusion of A.

The next lemma proves the completeness of the set of arguments $\mathscr{A}$.

Lemma 2 (completeness of arguments) *Let Σ be a set of propositions and let φ be a proposition. Moreover, let $\mathbb{L}$ be either the label $\mathbb{T}$ or $\mathbb{F}$.*

If $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ is a set of atomic assumptions with $\alpha_i = a_i : C_i$, $a_i \in \mathbf{N}$ and $C_i \in \mathbf{C_i}$, and if

$$\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \models \mathbb{L}\varphi$$

then there is a semantic tableaux with root $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$, and the tableaux is weakly closed.

Proof Let $\Gamma = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ be the root of a semantic tableaux.

Suppose that the tableaux is *not* weakly closed. Then there is an open leaf Λ. We can create a tableaux for $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ by adding the assumptions $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$ to every node in the original tableaux with root Γ. Let $\Gamma^* = \{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{L}}\varphi\}$ be the root of the resulting tableaux. Since $\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \models \mathbb{L}\varphi$, there exists no interpretation I such that $I \models \Gamma^*$. Therefore, there exists no interpretation I such that $I \models \Lambda^*$. Since we considered only atomic assumptions $\overline{\mathbb{C}}\alpha_i$, we cannot extend the tableaux by rewriting a proposition in Λ^*. Therefore, Λ^* must be strongly closed and for some α_i, $\{\mathbb{T}\alpha_i, \mathbb{F}\alpha_i\} \subseteq \Lambda^*$. This implies that $\{\mathbb{T}\alpha_i, \mathbb{F}\alpha_i\} \subseteq \Lambda$. Hence, Λ is weakly closed under the assumption $\overline{\mathbb{C}}\alpha_i$. Contradiction.

Hence, the lemma holds. □

The above lemma implies that we can find an argument $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{L}\varphi)$ for any set of assumption that, together with Σ, entails a conclusion $\mathbb{L}\varphi$.

The following lemma proves that for every conflict $\mathbb{C}\varphi$ entailed by a conflict-minimal interpretation, we can find an argument supporting $\mathbb{C}\varphi$ of which the assumptions are entailed by the conflict-minimal interpretation.

Lemma 3 *Let Σ be a set of propositions and let $I = \langle O, \pi\rangle$ be a conflict-minimal interpretation of Σ. Moreover, let φ be a proposition.*

If $I \models \mathbb{C}\varphi$ holds, then there is an argument $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\varphi)$ supporting $\mathbb{C}\varphi$ and for every assumption $\overline{\mathbb{C}}\alpha_i$, $I \models \overline{\mathbb{C}}\alpha_i$ holds.

Proof Let I be a conflict-minimal interpretation of Σ.

Suppose that $I \models \mathbb{C}\varphi$ holds. We can construct a tableaux for:

$$\begin{aligned}\Gamma \quad = \quad & \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{C}}\varphi\} \cup \\ & \{\overline{\mathbb{C}}\ a : C \mid C \in \mathbf{C}, \pi(a : C) \neq \{t, f\}\}\end{aligned}$$

Suppose that this tableaux is not strongly closed. Then there is an interpretation $I' = \langle O, \pi'\rangle$ satisfying the root Γ. Clearly, $I' <_c I$ because for every $a : C$ with $C \in \mathbf{C}$, if $\pi(a : C) \neq \{t, f\}$, then $\pi'(a : C) \neq \{t, f\}$. Since I is a conflict-minimal interpretation and since $I' \not\models \mathbb{C}\varphi$, we have a contradiction.

Hence, the tableaux is closed.

Since the tableaux with root Γ is closed, we can identify all assertions in $\{\overline{\mathbb{C}}\ a : C \mid C \in \mathbf{C}, \pi(a : C) \neq \{t, f\}\}$ that are **not** used to close a leaf of the tableaux. These assertions $\overline{\mathbb{C}}\ a : C$ play no role in the construction of the tableaux and can therefore be removed from every node of the tableaux. The result is still a valid and closed semantic tableaux with a new root Γ'. The assertions in $\{\overline{\mathbb{C}}\ a : C \mid C \in \mathbf{C}, \pi(a : C) \neq \{t, f\}\} \cap \Gamma'$ must all be used to strongly close leafs of the tableaux Γ', and also of Γ. A leaf that is strongly closed because of $\overline{\mathbb{C}}\ a : C$ can be closed weakly under the assumption $\overline{\mathbb{C}}\ a : C$. So, we may remove the remaining assertions $\overline{\mathbb{C}}\ a : C$ from the root Γ'. The result is still a valid semantic tableaux with root $\Gamma'' = \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mathbb{C}}\varphi\}$. This tableaux with root Γ'' is weakly closed, and by the construction of the tableaux, $I \models \overline{\mathbb{C}}\ a : C$ holds for every assumption $\overline{\mathbb{C}}\ a : C$ implied by a weak closure. Hence, we have constructed an argument $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\varphi)$ supporting $\mathbb{C}\varphi$ and for every assumption $\overline{\mathbb{C}}\alpha_i$, $I \models \overline{\mathbb{C}}\alpha_i$ holds.

Hence, the lemma holds. □

For the next lemma we need the following definition of a set of assumptions that is allowed by an extension.

Definition 19 *Let Ω be the set of all assumptions $\overline{\mathbb{C}}\alpha$ in the arguments $\mathscr{A}$. For any extension $\mathscr{E} \subseteq \mathscr{A}$,*

$$\Omega(\mathscr{E}) = \{\overline{\mathbb{C}}\alpha \in \Omega \mid \textit{no argument } A \in \mathscr{E} \textit{ supports } \mathbb{C}\alpha\}$$

is the set of assumptions allowed by the extension $\mathscr{E}$.

The last lemma proves that for every conflict-minimal interpretation there is a corresponding stable extension.

Lemma 4 *Let Σ be a set of propositions and let φ be a proposition. Moreover, let $\mathscr{A}$ be the complete set of arguments relevant to φ, let $\longrightarrow \subseteq \mathscr{A} \times \mathscr{A}$ be the attack relation determined by $\mathscr{A}$, and let $(\mathscr{A}, \longrightarrow)$ be the argumentation framework.*

For every conflict-minimal interpretation I of Σ, there is a stable extension $\mathscr{E}$ of $(\mathscr{A}, \longrightarrow)$ such that $I \models \Omega(\mathscr{E})$.

Proof Let I be a conflict-minimal interpretation and let

$$\mathscr{E} = \{(\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \varphi) \in \mathscr{A} \mid I \models \{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}\}$$

be the set of arguments $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \varphi)$ of which the assumptions are entailed by I.

Suppose $\mathscr{E}$ is not conflict-free. Then there is an argument $B \in \mathscr{E}$ such that $B \longrightarrow A$ with $A \in \mathscr{E}$. So, B supports $\mathbb{C}\psi$ and $\overline{\mathbb{C}}\psi$ is an assumption of A. Since I entails the assumptions of A, $I \not\models \mathbb{C}\psi$. Since I is a conflict-minimal interpretation of Σ entailing the assumptions of B, according to Lemma 1, $I \models \mathbb{C}\psi$. Contradiction.

Hence, $\mathscr{E}$ is a conflict-free set of argument.

Suppose that there exists an argument $A \in \mathscr{A}$ such that $A \notin \mathscr{E}$. Then, for some assumption $\overline{\mathbb{C}}\alpha$ of A, $I \not\models \overline{\mathbb{C}}\alpha$. So, $I \models \mathbb{C}\alpha$, and according to Lemma 3, there is an argument $B \in \mathscr{E}$ supporting $\mathbb{C}\alpha$. Therefore, $B \longrightarrow A$.

Hence, $\mathscr{E}$ attacks every argument $A \in \mathscr{A} \backslash \mathscr{E}$. Since $\mathscr{E}$ is also conflict-free, $\mathscr{E}$ is a *stable* extension of $(\mathscr{A}, \longrightarrow)$.

Suppose that $I \not\models \Omega(\mathscr{E})$. Then there is a $\overline{\mathbb{C}}\alpha \in \Omega(\mathscr{E})$ and $I \models \mathbb{C}\alpha$. According to Lemma 3, there is an argument $A = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\alpha)$ and $I \models \{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}$. So, $A \in \mathscr{E}$ and therefore, $\overline{\mathbb{C}}\alpha \notin \Omega(\mathscr{E})$. Contradiction.

Hence, $I \models \Omega(\mathscr{E})$. □

Using the results of the above lemmas, we can now prove the theorem.

Proof of Theorem 1
($\Rightarrow$) Let $\Sigma \models_{\leq_c} \varphi$.

> Suppose that there is stable extension $\mathscr{E}_i$ that does not contain an argument for φ. Then according to Lemma 2, $\{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \Omega(\mathscr{E}_i) \not\models \mathbb{T}\varphi$. So, there exists an interpretation I such that $I \models \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \cup \Omega(\mathscr{E}_i)$ but $I \not\models \mathbb{T}\varphi$. There must also exists a conflict-minimal interpretation I' of Σ and $I' \leq_c I$. Since the assumptions $\overline{\mathbb{C}}\, a : C \in \Omega(\mathscr{E}_i)$ all state that there is no conflict concerning the assertion $a : C$, $I' \models \Omega(\mathscr{E}_i)$ must hold. So, I' is a conflict-minimal interpretation of Σ and $I' \models \Omega(\mathscr{E}_i)$ but according to Lemma 2, $I' \not\models \mathbb{T}\varphi$. This implies $\Sigma \not\models_{\leq_c} \varphi$. Contradiction.

Hence, every stable extension $\mathscr{E}_i$ contains an argument for φ.

($\Leftarrow$) Let φ be supported by an argument in every stable extension $\mathscr{E}_i$.

> Suppose that $\Sigma \not\models_{\leq_c} \varphi$. Then there is a conflict-minimal interpretation I of Σ and $I \not\models \varphi$. Since I is a conflict-minimal interpretation of Σ, according to Lemma 4, there is a stable extension $\mathscr{E}_i$ and $I \models \Omega(\mathscr{E}_i)$. Since $\mathscr{E}_i$ contains an argument A supporting φ, the assumptions of A must be a subset of $\Omega(\mathscr{E}_i)$, and therefore I satisfies these assumptions. Then, according to Lemma 1, $I \models \varphi$. Contradiction.

Hence, $\Sigma \models_{\leq_c} \varphi$. □

In Example 5 in the previous section, we saw that one counter-argument implies multiple counter-arguments. The following proposition formalizes this observation.

Proposition 1 *Let* $A_0 = (\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\alpha_0)$.
Then $A_i = (\{\overline{\mathbb{C}}\alpha_0, \ldots, \overline{\mathbb{C}}\alpha_{i-1}, \overline{\mathbb{C}}\alpha_{i+1}, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\alpha_i)$ *is an argument for every* $1 \leq i \leq k$.

Proof The argument A_0 is the result of two tableaux, one for $\mathbb{T}\alpha_0$ and one for $\mathbb{F}\alpha_0$. Then, according to Lemma 1,

$$\{\overline{\mathbb{C}}\alpha_1, \ldots, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \models \mathbb{C}\alpha_0$$

where Σ the set of available propositions. This implies that

$$\{\overline{\mathbb{C}}\alpha_0, \ldots, \overline{\mathbb{C}}\alpha_{i-1}, \overline{\mathbb{C}}\alpha_{i+1}, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\} \models \mathbb{C}\alpha_i$$

So, $\{\overline{\mathbb{C}}\alpha_0, \ldots, \overline{\mathbb{C}}\alpha_{i-1}, \overline{\mathbb{C}}\alpha_{i+1}, \overline{\mathbb{C}}\alpha_k\} \cup \{\mathbb{T}\sigma \mid \sigma \in \Sigma\}$ entails both $\mathbb{T}\alpha_i$ and $\mathbb{F}\alpha_i$. Then, according to Lemma 2,

$$A_i = (\{\overline{\mathbb{C}}\alpha_0, \ldots, \overline{\mathbb{C}}\alpha_{i-1}, \overline{\mathbb{C}}\alpha_{i+1}, \ldots, \overline{\mathbb{C}}\alpha_k\}, \mathbb{C}\alpha_i)$$

is an argument for $\mathbb{C}\alpha_i$. □

Related Works

Reasoning in the presences of inconsistent information has been addressed using different approaches. Rescher (1964) proposed to focus on maximal consistent subsets of an inconsistent knowledge-base. This proposal was further developed by (Brewka 1989; Huang, van Harmelen, & ten Teije 2005; Poole 1988; Roos 1988; 1992). Brewka and Roos focus on preferred maximal consistent subsets of the knowledge-base while Poole and Huang et al. consider a single consistent subset of the knowledge-base supporting a conclusion. Roos (1992) defines a preferential semantics (Kraus, Lehmann, & Magidor 1990; Makinson 1994; Shoham 1987) entailing the conclusions that are entailed by every preferred maximal consistent subsets, and provides an assumption-based argumentation system capable of identifying the entailed conclusions.

Paraconsistent logics form another approach to handle inconsistent knowledge bases. Paraconsistent logics have a long history starting with Aristotle. From the beginning of the twentieth century, paraconsistent logics were developed by Orlov (1929), Asenjo (1966), da Costa (1974), Belnap (1977), Priest (1989) and others. For a survey of several paraconsistent logics, see for instance (Middelburg 2011).

This paper uses the semantics of the paraconsistent logic LP (Priest 1989; 1991) as starting point. Belnap's four-values semantics (1977) differs from the LP semantics in allowing the empty set of truth-values. Belnap's semantics was adapted to description logics by Patel-Schneider (1989). Ma et al. (2006; 2007; 2008; 2009) extend Patel-Schneider's work to more expressive description logics, and propose two new interpretations for the subsumption relation. Qiao and Roos (2011) propose another interpretation.

A proof theory based on the semantic tableaux method was first introduced by Beth (1955). The semantic tableaux methods have subsequently been developed for many logics. For an overview of several semantic tableaux methods, see (Hähnle 2001). Bloesch (1993) developed a semantic tableaux method for the paraconsistent logics LP and Belnap's 4-valued logic. This semantic tableaux method has been used as a starting point in this paper.

Argumentation theory has its roots in logic and rhetoric. It dates back to Greek philosophers such as Aristotle. Modern argumentation theory started with the work of Toulmin (1958). In Artificial Intelligence, the use of argumentation was promoted by authors such as Pollock (1987), Simari and Loui (1992), and others. Dung (1995) introduced the argumentation framework (AF) in which he abstracts from the structure of the argument and the way the argument is derived. In Dung's argumentation framework, arguments are represented by atoms over which an attack relation is defined. The argumentation framework is used to define an argumentation semantics in terms of sets of conflict-free arguments that defend themselves against attacking arguments. Dung defines three semantics for argumentation frameworks: the grounded, the stable and the preferred semantics. Other authors have proposed additional semantics to overcome some limitations of these three semantics. For an overview, see (Bench-Capon & Dunne 2007).

This paper uses a special type argumentation system called assumption-based argumentation (ABA). Assumption-based argumentation has been developed since the end of the 1980's (Bondarenko *et al.* 1997; Bondarenko, Toni, & Kowalski 1993; Gaertner & Toni 2007; Roos 1988; 1992). Dung et al. (2009) formalized assumption-based argumentation in terms of an argumentation framework.

Conclusions

This paper presented a three-valued semantics for $\mathcal{ALC}$, which is based on semantics of the paraconsistent logic LP. An assumption-based argumentation system for identifying conclusions supported by conflict-minimal interpretations was subsequently described. The assumption-based arguments are derived from open branches of a semantic tableaux. The assumptions close open branches by assuming that some proposition will not be assigned the truth-value CONFLICT. No assumptions are needed if conclusions hold is all three-valued interpretations. The described approach has also been implemented.

In future work we intend to extend the approach to the description logic $\mathcal{SROIQ}$. Moreover, we wish to investigate the computational efficiency of our approach in handling inconsistencies.

References

Asenjo, F. 1966. A calculus of antinomies. *Notre Dame Journal of Formal Logic* 7:103–105.

Baader, F.; Buchheit, M.; and Hollander, B. 1996. Cardinality restrictions on concepts. *Artificial Intelligence* 88(1–2):195–213.

Belnap, N. D. 1977. A useful four-valued logic. In Dunn, J. M., and Epstein, G., eds., *Modern Uses of Multiple-Valued Logic*. Reidel, Dordrecht. 8–37.

Bench-Capon, T., and Dunne, P. E. 2007. Argumentation in artificial intelligence. *Artificial Intelligence* 171:619–641.

Beth, E. W. 1955. *Semantic entailment and formal derivability*. Noord-Hollandsche Uitg. Mij.

Bloesch, A. 1993. A tableau style proof system for two paraconsistent logics. *Notre Dame Journal of Formal Logic* 34(2):295–301.

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.

Bondarenko, A.; Toni, F.; and Kowalski, R. 1993. An assumption-based framework for nonmonotonic reasoning. In *Proc. 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*. MIT Press.

Brewka, G. 1989. Preferred subtheories: an extended logical framework for default reasoning. In *International Joined Conference on Artificial Intelligence*, 1043–1048.

Buchheit, M.; Donini, F. M.; and Schaerf, A. 1993. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research* 1:109–138.

da Costa, N. 1974. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic* 15:497–510.

Dung, P. M.; Kowalski, R.; and Toni, F. 2009. Assumption-based argumentation. In Rahwan, I., and Simari, G., eds., *Argumentation in Artificial Intelligence*. Springer. 1–20.

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.

Gaertner, D., and Toni, F. 2007. Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems* 22(6):24–33.

Hähnle, R. 2001. *Tableaux and Related Methods*, volume 1. Elsevier and MIT Press. chapter 3, 100–178.

Huang, Z.; van Harmelen, F.; and ten Teije, A. 2005. Reasoning with inconsistent ontologies. In *IJCAI*, 454–459.

Kraus, S.; Lehmann, D.; and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44:167–207.

Ma, Y., and Hitzler, P. 2009. Paraconsistent reasoning for OWL 2. In Polleres, A., and Swift, T., eds., *Web Reasoning and Rule Systems*, volume 5837 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 197–211.

Ma, Y.; Hitzler, P.; and Lin, Z. 2007. Algorithms for paraconsistent reasoning with OWL. In Franconi, E.; Kifer, M.; and May, W., eds., *The Semantic Web: Research and Applications*, volume 4519 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 399–413.

Ma, Y.; Hitzler, P.; and Lin, Z. 2008. Paraconsistent reasoning for expressive and tractable description logics. In Baader, F.; Lutz, C.; and Motik, B., eds., *Proceedings of the 21st International Workshop on Description Logics, Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Ma, Y.; Lin, Z.; and Lin, Z. 2006. Inferring with inconsistent OWL DL ontology: A multi-valued logic approach. In Grust, T.; Höpfner, H.; Illarramendi, A.; Jablonski, S.; Mesiti, M.; Müller, S.; Patranjan, P.-L.; Sattler, K.-U.; Spiliopoulou, M.; and Wijsen, J., eds., *Current Trends in Database Technology - EDBT 2006*, volume 4254 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 535–553.

Makinson, D. 1994. Nonmonotonic reasoning and uncertain reasoning. In Gabbay, D., ed., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3. Oxford University Press. 35–110.

Middelburg, C. A. 2011. A survey of paraconsistent logics. *CoRR* abs/1103.4324.

Patel-Schneider, P. F. 1989. A four-valued semantics for terminological logics. *Artificial Intelligence* 38(3):319–351.

Pollock, J. L. 1987. Defeasible reasoning. *Cognitive Science* 11:481–518.

Poole, D. 1988. A logical framework for default reasoning. *Artificial Intelligence* 36:27–47.

Priest, G. 1989. Reasoning about truth. *Artificial Intelligence* 39(2):231–244.

Priest, G. 1991. Minimally inconsistent LP. *Studia Logica* 50:321–331.

Qiao, W., and Roos, N. 2011. Four-valued description logic for paraconsistent reasoning. In *BeNelux Conference on Artificial Intelligence (BNAIC)*.

Rescher, N. 1964. *Hypothetical Reasoning*. Studies in Logic. Amsterdam: North-Holland Publishing Co.

Roos, N. 1988. A preference logic for non-monotonic reasoning. Technical Report 88-94, Delft University of Technology, Faculty of Technical Mathematics and Informatics. ISSN 0922-5641.

Roos, N. 1992. A logic for reasoning with inconsistent knowledge. *Artificial Intelligence* 57:69–103.

Schmidt-Schauß, M., and Smolka, G. 1991. Attributive concept descriptions with complements. *Artificial Intelligence* 48(1):1–26.

Shoham, Y. 1987. A semantical approach to non-monotonic logics. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 388–392.

Simari, G. R., and Loui, R. P. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53:125–157.

Toulmin, S. 1958. *The uses of argument*. Cambridge University Press.

Some Thoughts about Benchmarks for NMR*

Daniel Le Berre
CNRS - Université d'Artois - France

Abstract

The NMR community would like to build a repository of benchmarks to push forward the design of systems implementing NMR as it has been the case for many other areas in AI. There are a number of lessons which can be learned from the experience of other communities. Here are a few thoughts about the requirements and choices to make before building such a repository.

What to expect

Over the last two decades, a huge number of communities have built repositories of benchmarks, mainly with the idea to evaluate running systems on a common set of problems. The oldest common input format for AI benchmarks is probably STRIPS (Fikes and Nilsson 1971), for planning systems. One of the oldest and most compelling one for reasoning engines is TPTP ("Thousands of Problems for Theorem Provers") (Sutcliffe 2009), the benchmarks library for First Order and Higher Order theorem provers. Such repository was built in 1993 and evolved since then as a companion to the CADE ATP System Competition (CASC) (Sutcliffe and Suttner 2006). There is an interplay between TPTP and CASC: TPTP is used to select benchmarks for CASC, benchmarks submitted to CASC are added eventually to TPTP and the solvers submitted to CASC are run on all TPTP benchmarks, and used to evaluate the practical complexity of those benchmarks. As such, over the years, benchmarks are ranked from hard to medium to easy with the improvements of the solvers. This is exactly the kind of virtuous circle one would like to see in each community. In the NMR community, a similar library exists with Asparagus[1], which feeds the ASP competition (Gebser et al. 2007).

There are however reasons which prevent it. Take for instance the SAT community. Its common input format is based on the Second Dimacs Challenge input format (Johnson and Trick 1996), one of the first SAT competitions. The benchmarks used for that competitive event has been a *de facto* standard for evaluating SAT solvers in practice. A system similar to TPTP was built by Laurent Simon in 2000: SatEx (Simon and Chatalic 2001). However, the number of SAT solvers available in the SAT community became quickly much larger than the number of ATP systems, because of its increasing practical interest in hardware verification, and because it is much easier to develop a SAT solver than a First Order theorem solver. As such, it became quickly impossible to run all SAT solvers on all available benchmarks. A tradeoff was to organize a yearly SAT competitive event since 2002 (Simon, Le Berre, and Hirsch 2005), to give a snapshot of the performances of recent solvers on a selection of benchmarks.

Modeling versus Benchmarking

One of the first question which arises when creating a benchmark format is to be clear about the target of the format. There are mainly two choices: one is to please the end user, by providing a format which simplifies modeling problems in that format, the other one is to please the solver designers, to make sure that they integrate a way to read that format. High level input format such as PDDL, TPTP, ASP, SMT and Minizinc (CSP) are clearly modeling oriented. Formats designed by the SAT community (SAT, MAXSAT, PBO, QBF, MUS ...) are clearly solver oriented.

There are advantages and inconveniences for both approaches. The user oriented format favors the submissions of problems by the community, because the input format is human understandable and easy to modify. However, such format may require a huge effort from the solver designer to adapt his solver to such format. This happened for instance for the SMT LIB 2 format, which was quite different from the original SMT LIB format, so it took time to be adopted by the SMT solver designers. Another issue with user oriented formats are the potential high learning curve to understand all its subtleties. For instance, it took several rounds in the Mancoosi International Solver Competition (MiSC) (Abate and Treinen 2011) to see all solvers answering correctly to the requests because the input format was assuming some domain knowledge not obvious for a solver designer.

The main advantage of the solver oriented format is to be easy to integrate into any exiting system. It is the way to go if the community wants to evaluate existing systems on a common basis. It was the idea behind the XCSP format for CSP solvers for instance (Lecoutre, Roussel, and van Dongen 2010). The major drawback of such approach is to force the end user to rely on an intermediate representation to generate those benchmarks, and to perform some

*This work has been supported in part by ANR Tuples.

[1] http://asparagus.cs.uni-potsdam.de

tasks by hand which may be automated using a higher level input format. For instance, in the case of SAT, it is required to translate the original problem into propositional variables and clauses. Many users are not aware of basic principles and advanced techniques to perform those tasks efficiently.

One way to please both part is to provide a end-user input format, to favor the contribution of problems, and a solver input format to please the solver designers, with a default translator from the first one to the second one. This is the spirit of the Minizinc and Flatzinc formats in the CSP community (Stuckey, Becket, and Fischer 2010).

Data versus Protocol

Another question raised when designing an input format is whether the benchmark represents data or whether it represents a full protocol. The problem is orthogonal to the abstraction level of the input format: it is directed by the nature of the problems to be solved.

In many cases, benchmarks represent data, in one or multiple files (e.g. rules and facts, domain and instance), and the system answers to a single query. There are other cases in which some interaction with the system is required: the SMT LIB 2 format (Barrett, Stump, and Tinelli 2010) for instance defines a protocol to communicate with the system to solve problems incrementally, which means that the system in that case is stateful. The Aiger format used in the hardware model checking competition (Biere and Jussila 2007) also provides some incremental capabilities, which corresponds to the unrolling of the Bounded Model Checking approach.

The protocol point of view is great for playing with toy examples, thus good for education. It also allows to interface with the solver without worrying about the details. From a system designer, it requires generally more effort to maintain the state of the system between queries. From an efficiency point of view, an API is usually preferred in practice for interacting with a system.

Checkable queries

Once a common benchmark format is setup, it is important to make sure that the benchmarks are correctly read by the systems, and that the queries to the systems provide answers checkable by a third party tool. In the case of SAT for instance, while the decision problem answer is yes or no, in practice, the SAT solvers have always been asked to provide a certificate (a model) in case of satisfiability. Such certificate can be checked by an independent tool: if it satisfies all clauses, then the answer is checked, else the answer is invalid. If two solvers disagreed on the satisfiability of a benchmark, checking the certificate of the yes answer allowed to spot incorrect solvers when that certificate was correct: the no answer is clearly incorrect in that case. Nothing could be decided if the certificate was invalid: there are many reasons why a SAT solvers could answer SAT and provide an incorrect certificate (complex pre-processing and inprocessing being the most probable case). There has been since 2005 an effort to also provide checkable no answers to SAT solvers (Van Gelder 2012), but very few solver designers implemented it until a simpler proof certificate requiring to add only a few lines of code in the solver was designed in 2013 (Heule, Jr., and Wetzler 2013). As such, SAT solvers answers can now be checked both in case of satisfiability and unsatisfiability.

Note that it is not always possible to check the system answer. It happens for instance for QBF solvers, for which a certificate would be a winning strategy for the existential player. During the QBF evaluations, many QBF solvers disagreed on the status of the benchmarks. As such, several approaches were taken to sort out the situation: majority voting, let the solvers play against each other (Narizzano et al. 2009), fuzz testing and delta debugging (Brummayer, Lonsing, and Biere 2010). It also happens when computing an optimal solutions in Pseudo-Boolean Optimization or MaxSat competitions: in that case, one just check the value of the certificate returned by the solver, and that no other solver found a better solution. A better but resource consuming approach would be to create a new benchmark to check that there is no better solution. In the same spirit, when tools for computing Minimal Unsatisfiable Subformula are used, it is very demanding to check for each answer that both the set of constraints is unsatisfiable and that removing any clause makes the set of constraints satisfiable. In the MUS track of the SAT competition 2011, only the first test was performed, offline.

It is important in the first place to provide both to the end users and the solver designers some sample benchmarks with their expected answer, or a basic solver able to solve small benchmarks. This is especially true if the input format is user oriented. For instance, the MISC competition introduced new features in the input format without providing sample benchmarks with those new features. Those features were not correctly implemented by all systems, thus the systems answered differently on some of the benchmarks, making comparisons between the systems hardly possible.

Chicken and egg problem

It is unlikely that people start providing benchmarks in one input format without having a system to test some reduced scale benchmarks. It is also unlikely that solver designers start supporting an input format without having some sample benchmarks to play with. That's the reason why a common input format is a community effort and it relies generally on a small group of people who are concerned by the subject. One can take as example the attempt during the SAT 2005 competition to push forward a non CNF input format for SAT[2]: a common input format was defined, allowing to define arbitrary gates, and a few sample instances were provided as part of a specific track of the competition. No submission of benchmarks nor systems were received for such track. Another attempt, using a more specific non clausal format (And Inverter Graph, AIG), but well suited for model checking, received more interest in 2007, and became a competition on its own for hardware model checking (Biere and Jussila 2007). The main difference between the two attempts was that a small community agreed to support

[2] `http://www.satcompetition.org/2005/`

AIG, some translators and checkers were available (AIGER tool suite[3]) and many model checking benchmarks were provided in such format.

The input format of a given system may become a *de facto* common input format. In the case of argumentation frameworks for instance, several systems based on different technologies have been designed by the same group, using a common input format[4]. Such input format could be a good starting point for creating a common argumentation system input framework.

If it is not possible to provide both some sample benchmarks and a basic solver, it is important to provide a way to check the answers. The minimum requirement here would be to provide the expected answer for each sample benchmark in a text file. A better approach would be to provide a way to check the answer thanks to a certificate using an independent checker software. Note that in such a case, a common output (certificate) format must also be defined.

Reusing benchmarks from other communities

Reusing benchmarks from other communities is certainly an easy way to start collecting benchmarks. Most benchmarks libraries contain well-known academic benchmarks (including randomly generated ones), benchmarks based on other community benchmarks (SAT has many benchmarks modeling properties to check on circuit benchmarks from ISCAS for instance), and finally dedicated benchmarks. The latter are the harder to find at the beginning. As such, reusing benchmarks from other communities is often the only way to retrieve non-academic benchmarks.

Note that there are some side effects in reusing benchmarks from other communities. The first one is to pay attention when evaluating systems on the origin of those systems. For instance, there are two optimization extensions to SAT for which benchmarks are available: MAXSAT and Pseudo Boolean Optimization. The PBO benchmarks appeared before the MAXSAT ones, and some benchmarks from PBO have been expressed as MAXSAT problems (optimization problems with one linear objective function and a set of clauses can be equally expressed in both frameworks). Some solvers designed to solve PBO problems have been extended to solve MAXSAT problems (e.g. Sat4j). Those solvers usually perform very well on the benchmarks originating from PBO. In the same spirit, some of the Pseudo Boolean benchmarks are coming from MIPLIB[5], a repository of Mixed Integer Linear Programming benchmarks used by MILP optimizers developers since 1992 to evaluate their systems. It is no surprise if tools such as CPLEX performs very well on those benchmarks when compared to "classical" Pseudo-Boolean solvers.

In the case of NMR, it is often the case that the systems have to deal with inconsistency. As such, it is tempting for instance to use unsatisfiable SAT benchmarks to evaluate NMR systems. But those systems usually require additional informations (e.g. a stratification of the clauses, a confidence for each clause, etc) and some arbitrary choices would have to be done to fit in the context (i.e. creating individual satisfiable sub-CNF for each agent in a multi-agent context). The additional information may be generated using a specific distribution of values (e.g. randomly and uniformly assigning the clauses to a given number of strata), or arbitrarily (e.g. make strata from sets of consecutive clauses, of identical or random sizes). Those benchmarks, despite not being related at all with a real NMR problem, do have the benefit to allow different systems to be compared on the same basis.

It is also interesting to note that there exists a format in the SAT community which is very close to stratified knowledge bases: *Group oriented CNF*, introduced in the MUS special track in the SAT 2011 competition [6]. The benchmarks in that format are coming from circuit designs (Nadel 2010; Ryvchin and Strichman 2011), where each group (stratum) of clauses correspond to a subcircuit, a specific group contains hard clauses which correspond to integrity constraints (i.e. knowledge) while the remaining groups are soft clauses which can be enabled or disabled altogether (i.e. beliefs). The benchmarks are not satisfiable if all groups of clauses are enabled. There exists 197 group oriented CNF benchmarks available from the SAT 2011 competition web site, all corresponding to "real" designs. They could be a good starting point to test systems requiring stratified knowledge bases.

The bias of benchmarking systems

It should also be clear that the benchmarks used to evaluate the systems drive in some sense which systems are going to be developed or improved by the community.

Anyone looking at the winners of the various SAT competitions[7] can check that solvers behave differently on randomly generated benchmarks and benchmarks coming from real applications or hard combinatorial problems. This is true for any community. Randomly generated benchmarks are interesting for two reasons: they are easy to generate and can generally be formally defined. Combinatorial benchmarks are important because they usually force the system to exhibit worst case behavior. Application benchmarks are interesting because they provide some hints about the practical complexity of the problem. Note that if application benchmarks in SAT tend to be "easier" in practice than say combinatorial benchmarks, it is only the case because people worked hard to find the right heuristics, data structures, etc. to manage those problems.

For that reason, one should always be very careful when looking at any competitive event results, or when evaluating his system on a given set of benchmarks. It took some time for the MAXSAT competition[8] to obtain benchmarks coming from real applications. Before 2008, SAT-based MAXSAT solvers performed relatively poorly on the problems available for the competition (mainly randomly

[3] http://fmv.jku.at/aiger/

[4] http://www.dbai.tuwien.ac.at/research/project/argumentation/

[5] http://miplib.zib.de

[6] http://www.satcompetition.org/2011/rules.pdf

[7] http://www.satcompetition.org/

[8] http://maxsat.ia.udl.cat/

generated, based on academic problems). Once application benchmarks became available, SAT-based MAXSAT solvers performed much better on those problems, especially core-guided MAXSAT solvers. So the benchmarks used to evaluate the systems eventually influence the development of those systems.

There are also subtle differences between benchmarks coming from real applications. The SAT community has been driven by Bounded Model Checking benchmarks from the end of the 90's to mid 2000's. As such, the solvers designed during that period were especially relevant to that application: the winners of the SAT competition could be directly integrated into model checkers. With an increase of the diversity of its applications, the available benchmarks for SAT are now quite different in structure from those BMC benchmarks. Which means that the best performing SAT solver during the SAT competition may not be the best solver for the particular case of BMC.

Benchmarks libraries

Benchmarks are usually made available to the community through a library: CSPLIB, SATLIB, PBLIB, SMTLIB, etc. However, it is an issue to manage those libraries in the long term. A good example is SATLIB (Hoos and Sttzle 2000). It was designed in 1999 to host the benchmarks made available to the SAT community. It did a good job at collecting the benchmarks generated during the 90's. However, the huge increase in number of benchmarks (and their size!) in early 2000 made it hard to catch up after 2001, so the SAT competition web sites have been providing the benchmarks used in the competitions since then. The situation is not ideal because there is no longer now in the SAT community a central place where the benchmarks can be accessed. Some of the benchmarks, which were made available to the research community by IBM (Zarpas 2006), can no longer be distributed. It is thus very difficult to reproduce some experiments, to evaluate the efficiency of new solvers on those benchmarks. Having a community driven central repository may help to avoid such situation.

The CSP library [9] succeeded in maintaining a library of problems for 15 years. Note that those problems are not in a uniform format, but rather described in their own format. The library is much about problems than benchmarks.

The library of benchmarks one community would like to mimic today are probably TPTP[10] or MIPLIB. Those libraries have been available for two decades now and are the central sources of benchmarks for their respective community. The benchmarks are ranked by difficulty, and updated regularly at the light of the performances of new systems.

Conclusion

Many communities built central repositories of benchmarks to be able to compare the performance of their systems. The success of those repositories relies first on the adoption of it format by the community, and second on the availability of benchmarks for which some information is provided: difficulty, expected answer, runtime of existing systems, etc.

For a community such as NMR, which addresses a wide range of different problems, the first step is to decide on which problems a first effort of standardization is required. The heuristics can be either the maturity of existing systems in the community or the importance of the problem for the community. In either case, the choice of the format for the benchmarks will be important: should it be user oriented or system oriented? data or protocol oriented?

Defining a format and providing benchmarks is not sufficient to reach adoption: sample results and answers checkers are essential components to allow system designers to adopt such format. In order to receive application benchmarks, some systems supporting that format should be provided as well, even if they are not very efficient: they are sufficient to discover the meaning of the benchmark format, or to check the answers of a system under development.

Both benchmarks providers and system developers can make mistakes. As such, tools which check the syntax of the input and the correctness of the system answers will help providing meaningful benchmarks and systems results.

In order to reuse benchmarks from other communities, tools which allow to translate to and from different formats are also welcome.

Organizing competitive events has been a great source of new benchmarks for many communities. I am looking forward the first NMR competition.

[9] http://www.csplib.org/

[10] http://www.tptp.org/

References

Abate, P., and Treinen, R. 2011. Mancoosi Deliverable D5.4: Report on the international competition. Rapport de recherche.

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB Standard: Version 2.0. In Gupta, A., and Kroening, D., eds., *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*.

Biere, A., and Jussila, T. 2007. Hardware model checking competition. http://fmv.jku.at/hwmcc07/.

Brummayer, R.; Lonsing, F.; and Biere, A. 2010. Automated testing and debugging of sat and qbf solvers. In Strichman, O., and Szeider, S., eds., *SAT*, volume 6175 of *Lecture Notes in Computer Science*, 44–57. Springer.

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.

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczynski, M. 2007. The first answer set programming system competition. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, 3–17. Springer.

Heule, M.; Jr., W. A. H.; and Wetzler, N. 2013. Verifying refutations with extended resolution. In Bonacina, M. P., ed., *CADE*, volume 7898 of *Lecture Notes in Computer Science*, 345–359. Springer.

Hoos, H. H., and Sttzle, T. 2000. Satlib: An online resource

for research on sat. In Gent, I. P.; van Maaren, H.; and Walsh, T., eds., *SAT 2000*, 283–292. IOS Press.

Johnson, D., and Trick, M., eds. 1996. *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Lecoutre, C.; Roussel, O.; and van Dongen, M. R. C. 2010. Promoting robust black-box solvers through competitions. *Constraints* 15(3):317–326.

Nadel, A. 2010. Boosting minimal unsatisfiable core extraction. In Bloem, R., and Sharygina, N., eds., *FMCAD*, 221–229. IEEE.

Narizzano, M.; Peschiera, C.; Pulina, L.; and Tacchella, A. 2009. Evaluating and certifying qbfs: A comparison of state-of-the-art tools. *AI Commun.* 22(4):191–210.

Ryvchin, V., and Strichman, O. 2011. Faster extraction of high-level minimal unsatisfiable cores. In Sakallah, K. A., and Simon, L., eds., *SAT*, volume 6695 of *Lecture Notes in Computer Science*, 174–187. Springer.

Simon, L., and Chatalic, P. 2001. Satex: A web-based framework for sat experimentation. *Electronic Notes in Discrete Mathematics* 9:129–149.

Simon, L.; Le Berre, D.; and Hirsch, E. A. 2005. The SAT2002 competition. *Ann. Math. Artif. Intell.* 43(1):307–342.

Stuckey, P. J.; Becket, R.; and Fischer, J. 2010. Philosophy of the minizinc challenge. *Constraints* 15(3):307–316.

Sutcliffe, G., and Suttner, C. 2006. The State of CASC. *AI Communications* 19(1):35–48.

Sutcliffe, G. 2009. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning* 43(4):337–362.

Van Gelder, A. 2012. Producing and verifying extremely large propositional refutations - have your cake and eat it too. *Ann. Math. Artif. Intell.* 65(4):329–372.

Zarpas, E. 2006. Back to the SAT05 Competition: an a Posteriori Analysis of Solver Performance on Industrial Benchmarks. *JSAT* 2(1-4):229–237.

Towards a Benchmark of Natural Language Arguments

Elena Cabrio and Serena Villata
INRIA Sophia Antipolis
France

Abstract

The connections among natural language processing and argumentation theory are becoming stronger in the latest years, with a growing amount of works going in this direction, in different scenarios and applying heterogeneous techniques. In this paper, we present two datasets we built to cope with the combination of the Textual Entailment framework and bipolar abstract argumentation. In our approach, such datasets are used to automatically identify through a Textual Entailment system the relations among the arguments (i.e., *attack*, *support*), and then the resulting bipolar argumentation graphs are analyzed to compute the accepted arguments.

Introduction

Until recent years, the idea of "argumentation" as the process of creating arguments for and against competing claims was a subject of interest to philosophers and lawyers. In recent years, however, there has been a growth of interest in the subject from formal and technical perspectives in Artificial Intelligence, and a wide use of argumentation technologies in practical applications. However, such applications are always constrained by the fact that natural language arguments cannot be automatically processed by such argumentation technologies. Arguments are usually presented either as the abstract nodes of a directed graph where the edges represent the relations of attack and support (e.g., in abstract argumentation theory (Dung 1995) and in bipolar argumentation (Cayrol and Lagasquie-Schiex 2005), respectively).

Natural language arguments are usually used in the argumentation literature to provide *ad-hoc* examples to help the reader in the understanding of the rationale behind the formal approach which is then introduced, but the need to find automatic ways to process natural language arguments is becoming more and more important. On the one side, when dealing with natural language processing techniques, the first step consists in finding the data on which the system is trained and evaluated. On the other side, in argumentation theory there is a growing need to define benchmarks for argumentation to test implemented systems and proposed theories. In this paper, we address the following research question: *how to build a dataset of natural language arguments?*

The definition of a dataset of natural language arguments is not a straightforward task: first, there is the need to identify the kind of natural language arguments to be collected (e.g., online debates, newspaper articles, blogs and forums, etc.), and second, there is the need to annotate the data according to the addressed task from the natural language processing point of view (e.g., classification, textual entailment (Dagan et al. 2009), etc.).

Our goal (Cabrio and Villata 2013) is to analyze natural language debates in order to understand, given a huge debate, what are the winning arguments (through *acceptability semantics*) and who proposed them. In order to achieve such goal, we have identified two different scenarios to extract our data: *(i)* online debate platforms like Debatepedia[1] and ProCon[2] present a set of topics to be discussed, and participants argue about the issue the platform proposes on a selected topic, highlighting whether their "arguments" are in favor or against the central issue, or with respect to the other participants' arguments, and *(ii)* the screenplay of a movie titled "Twelve Angry Men" where the jurors of a trial discuss in order to decide whether a young boy is guilty or not, and before the end of each act they vote to verify whether they all agree about his guiltiness. These two scenarios lead to two different resources: the online debates resource collects the arguments in favor or against the main issue or the other arguments into small bipolar argumentation graphs, while the "Twelve Angry Men" resource collects again pro and con arguments but they compose three bipolar argumentation graphs whose complexity is higher than debates graphs. Note that the first resource consists of an integration of the dataset of natural language arguments we presented in (Cabrio and Villata 2013) with new data extracted from the ProCon debate platform.

These two resources represent a first step towards the construction of a benchmark of natural language arguments, to be exploited by existing argumentation systems as data-driven examples of argumentation frameworks. In our datasets, arguments are cast into pairs where the two arguments composing the pair are linked by a positive relation (a *support* relation in argumentation) or a negative relation (an *attack* relation in argumentation). From these pairs, the argumentation graphs are constructed.

The remainder of the paper is organized as follows:

[1] http://idebate.org/debatabase
[2] http://www.procon.org/

the next section presents the two datasets from Debatepedia/ProCon and Twelve Angry Men and how they have been extracted and annotated, then some conclusions are drawn.

Natural Language Arguments: datasets

As introduced before, the rationale underlying the datasets of natural language arguments we created was to support the task of understanding, given a huge debate, what are the winning arguments, and who proposed them. In an application framework, we can divide such task into two consecutive subtasks, namely *i)* the recognition of the semantic relations between couples of arguments in a debate (i.e. if one statement is supporting or attacking another claim), *ii)* and given all the arguments that are part of a debate and the acceptability semantics, to reason over the graph of arguments with the aim of deciding which are the accepted ones.

To reflect this separation into two subtasks, each dataset that we will describe in detail in the following subsections is therefore composed of two layers. Given a set of arguments linked among them (e.g in a debate):

1. we couple each argument with the argument to which it is related (i.e. that it attacks or supports). The first layer of the dataset is therefore composed of couples of arguments (each one labeled with a univocal ID), annotated with the semantic relations linking them (i.e. *attack* or *support*);
2. starting from the pairs of arguments in the first layer of the dataset, we then build a bipolar entailment graph for each of the topics in the dataset. In the second layer of the dataset, we find therefore graphs of arguments, where the arguments are the nodes of the graph, and the relations among the arguments correspond to the edges of the graphs.

To create the data set of arguments pairs, we follow the criteria defined and used by the organizers of the Recognizing Textual Entailment challenge.[3] To test the progress of TE systems in a comparable setting, the participants to RTE challenge are provided with data sets composed of T-H pairs involving various levels of entailment reasoning (e.g. lexical, syntactic), and TE systems are required to produce a correct judgment on the given pairs (i.e. to say if the meaning of one text snippet can be inferred from the other). Two kinds of judgments are allowed: two-way (yes or no entailment) or three-way judgment (entailment, contradiction, unknown). To perform the latter, in case there is no entailment between T and H systems must be able to distinguish whether the truth of H is contradicted by T, or remains unknown on the basis of the information contained in T. To correctly judge each single pair inside the RTE data sets, systems are expected to cope both with the different linguistic phenomena involved in TE, and with the complex ways in which they interact. The data available for the RTE challenges are not suitable for our goal, since the pairs are extracted from news and are not linked among each others (i.e. they do not report opinions on a certain topic). However, the task of recognizing semantic relations among pairs of textual fragments is very close to ours, and therefore we follow the guidelines provided by the organizers of RTE for the creation of their datasets. For instance, in (Cabrio and Villata 2013) we experiment with the application of a TE (Dagan et al. 2009) to automatically identify the arguments in the text and to specify which kind of relation links each couple of arguments.

Debatepedia dataset

To build our first benchmark of natural language arguments, we selected Debatepedia and ProCon, two encyclopedias of pro and con arguments on critical issues. To fill in the first layer of the dataset, we manually selected a set of topics (Table 2 column *Topics*) of Debatepedia/ProCon debates, and for each topic we apply the following procedure:

1. the main issue (i.e., the title of the debate in its affirmative form) is considered as the starting argument;
2. each user opinion is extracted and considered as an argument;
3. since *attack* and *support* are binary relations, the arguments are coupled with:
   (a) the starting argument, or
   (b) other arguments in the same discussion to which the most recent argument refers (i.e., when a user opinion supports or attacks an argument previously expressed by another user, we couple the former with the latter), following the chronological order to maintain the dialogue structure;
4. the resulting pairs of arguments are then tagged with the appropriate relation, i.e., *attack* or *support*[4].

Using Debatepedia/ProCon as case study provides us with already annotated arguments (*pro* $\Rightarrow$ *entailment*[5], and *con* $\Rightarrow$ *contradiction*), and casts our task as a yes/no entailment task. To show a step-by-step application of the procedure, let us consider the debated issue *Can coca be classified as a narcotic?*. At step 1, we transform its title into the affirmative form, and we consider it as the starting argument (a). Then, at step 2, we extract all the users opinions concerning this issue (both pro and con), e.g., (b), (c) and (d):

Example 1.
(a) *Coca can be classified as a narcotic.*

(b) *In 1992 the World Health Organization's Expert Committee on Drug Dependence (ECDD) undertook a "prereview" of coca leaf at its 28th meeting. The 28th ECDD report concluded that, "the coca leaf is appropriately scheduled as a narcotic under the Single Convention on Narcotic Drugs, 1961, since cocaine is readily extractable from the leaf." This ease of extraction makes coca*

[3] Since its inception in 2004, the PASCAL RTE Challenges have promoted research in RTE `http://www.nist.gov/tac/2010/RTE/`

[4] The data set is freely available at `http://www-sop.inria.fr/NoDE/`.

[5] Here we consider only arguments implying another argument. Arguments "supporting" another argument, but not inferring it will be discussed in the next subsection.

and cocaine inextricably linked. Therefore, because cocaine is defined as a narcotic, coca must also be defined in this way.

(c) *Coca in its natural state is not a narcotic. What is absurd about the 1961 convention is that it considers the coca leaf in its natural, unaltered state to be a narcotic. The paste or the concentrate that is extracted from the coca leaf, commonly known as cocaine, is indeed a narcotic, but the plant itself is not.*

(d) *Coca is not cocaine. Coca is distinct from cocaine. Coca is a natural leaf with very mild effects when chewed. Cocaine is a highly processed and concentrated drug using derivatives from coca, and therefore should not be considered as a narcotic.*

At step 3a we couple the arguments (b) and (d) with the starting issue since they are directly linked with it, and at step 3b we couple argument (c) with argument (b), and argument (d) with argument (c) since they follow one another in the discussion (i.e. user expressing argument (c) answers back to user expressing argument (b), so the arguments are concatenated - the same for arguments (d) and (c)).
At step 4, the resulting pairs of arguments are then tagged with the appropriate relation: **(b)** *supports* **(a)**, **(d)** *attacks* **(a)**, **(c)** *attacks* **(b)** and **(d)** *supports* **(c)**.

We have collected 260 T-H pairs (Table 2), 160 to train and 100 to test the TE system. The training set is composed by 85 entailment and 75 contradiction pairs, while the test set by 55 entailment and 45 contradiction pairs. The pairs considered for the test set concern completely new topics.

Basing on the TE definition, an annotator with skills in linguistics has carried out a first phase of manual annotation of the Debatepedia data set. Then, to assess the validity of the annotation task and the reliability of the obtained data set, the same annotation task has been independently carried out also by a second annotator, so as to compute inter-annotator agreement. It has been calculated on a sample of 100 argument pairs (randomly extracted).

The statistical measure usually used in NLP to calculate the inter-rater agreement for categorical items is Cohen's kappa coefficient (Carletta 1996), that is generally thought to be a more robust measure than simple percent agreement calculation since κ takes into account the agreement occurring by chance. More specifically, Cohen's kappa measures the agreement between two raters who each classifies N items into C mutually exclusive categories. The equation for κ is:

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)} \quad (1)$$

where Pr(a) is the relative observed agreement among raters, and Pr(e) is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly saying each category. If the raters are in complete agreement then κ = 1. If there is no agreement among the raters other than what would be expected by chance (as defined by Pr(e)), κ = 0. For NLP tasks, the inter-annotator agreement is considered as significant when κ >0.6. Applying the formula (1) to our data, the inter-annotator agreement results in $\kappa = 0.7$. As a rule of thumb, this is a satisfactory agreement, therefore we consider these annotated data sets as the *goldstandard*. The goldstandard is the reference data set to which the performances of automated systems can be compared.

Training set				
Topic	**#argum**	**#pairs**		
		TOT.	**yes**	**no**
Violent games/aggressiveness	16	15	8	7
China one-child policy	11	10	6	4
Consider coca as a narcotic	15	14	7	7
Child beauty contests	12	11	7	4
Arming Libyan rebels	10	9	4	5
Random alcohol breath tests	8	7	4	3
Osama death photo	11	10	5	5
Privatizing social security	11	10	5	5
Internet access as a right	15	14	9	5
Tablets vs. Textbooks	22	21	11	10
Obesity	16	15	7	8
Abortion	25	24	12	12
TOTAL	109	**100**	**55**	**45**
Test set				
Topic	**#argum**	**#pairs**		
		TOT.	**yes**	**no**
Ground zero mosque	9	8	3	5
Mandatory military service	11	10	3	7
No fly zone over Libya	11	10	6	4
Airport security profiling	9	8	4	4
Solar energy	16	15	11	4
Natural gas vehicles	12	11	5	6
Use of cell phones/driving	11	10	5	5
Marijuana legalization	17	16	10	6
Gay marriage as a right	7	6	4	2
Vegetarianism	7	6	4	2
TOTAL	110	**160**	**85**	**75**

Table 1: The Debatepedia/ProCon data set

To build the bipolar argumentation graphs associated to the Debatepedia dataset, we have considered the pairs annotated in the first layer and we have built a bipolar entailment graph for each of the topic in the dataset (12 topics in the training set and 10 topics in the test set, listed in Table 2).

Figure 1 shows the average dimension of a bipolar argumentation graph in the Debatepedia/ProCon dataset. Note that no cycle is present, as well as in all the other graphs of such dataset. All graphs are available online, together with the XML data set.

Debatepedia extended dataset The dataset described in the previous section was created respecting the assumption that the TE relation and the support relation are equivalent, i.e. in all the previously collected pairs both TE and support relations (or contradiction and attack relations) hold.

For the second study described in (Cabrio and Villata 2013) we wanted to move a step further, to understand whether it is always the case that support is equivalent to TE

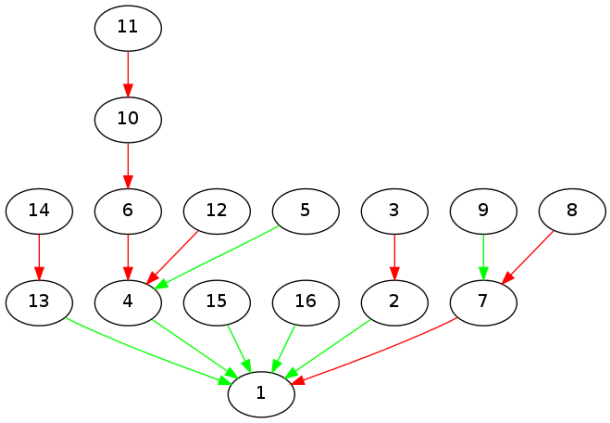


Figure 1: The bipolar argumentation framework resulting from the topic "Obesity" of Pro/Con (red edges represent attack and green ones represent support).

(and contradiction to attack). We therefore apply again the extraction methodology described in the previous section to extend our data set. In total, our new data set contains 310 different arguments and 320 argument pairs (179 expressing the *support* relation among the involved arguments, and 141 expressing the *attack* relation, see Table 2). We consider the obtained data set as representative of human debates in a non-controlled setting (Debatepedia users position their arguments with respect to the others as PRO or CON, the data are not biased).

***Debatepedia* extended data set**		
Topic	**#argum**	**#pairs**
Violent games/aggressiveness	17	23
China one-child policy	11	14
Consider coca as a narcotic	17	22
Child beauty contests	13	17
Arming Libyan rebels	13	15
Random alcohol breath tests	11	14
Osama death photo	22	24
Privatizing social security	12	13
Internet access as a right	15	17
Ground zero mosque	11	12
Mandatory military service	15	17
No fly zone over Libya	18	19
Airport security profiling	12	13
Solar energy	18	19
Natural gas vehicles	16	17
Use of cell phones/driving	16	16
Marijuana legalization	23	25
Gay marriage as a right	10	10
Vegetarianism	14	13
TOTAL	310	**320**

Table 2: Debatepedia extended data set

Again, an annotator with skills in linguistics has carried out a first phase of annotation of the extended Debatepedia data set. The goal of such annotation was to individually consider each pair of *support* and *attack* among arguments, and to additionally tag them as *entailment*, *contradiction* or *null*. The *null* judgment can be assigned in case an argument is supporting another argument without inferring it, or the argument is attacking another argument without contradicting it. As exemplified in Example 1, a correct entailment pair is **(b)** $\Rightarrow$ **(a)**, while a contradiction is **(d)** $\not\Rightarrow$ **(a)**. A *null* judgment is assigned to **(d) - (c)**, since the former argument supports the latter without inferring it. Our data set is an extended version of (Cabrio and Villata 2012)'s one allowing for a deeper investigation.

Again, to assess the validity of the annotation task, we have calculated the inter-annotator agreement. Another annotator with skills in linguistics has therefore independently annotated a sample of 100 pairs of the data set. We calculated the inter-annotator agreement considering the argument pairs tagged as *support* and *attacks* by both annotators, and we verify the agreement between the pairs tagged as *entailment* and as *null* (i.e. no entailment), and as *contradiction* and as *null* (i.e. no contradiction), respectively. Applying κ to our data, the agreement for our task is $\kappa = 0.74$. As a rule of thumb, this is a satisfactory agreement. Table 3 reports the results of the annotation on our Debatepedia data set, as resulting after a reconciliation phase carried out by the annotators[6].

Relations		**% arg. (# arg.)**
support	+ *entailment*	61.6 (111)
	- *entailment (null)*	38.4 (69)
attack	+ *contradiction*	71.4 (100)
	- *contradiction (null)*	28.6 (40)

Table 3: Support and TE relations on Debatepedia data set.

On the 320 pairs of the data set, 180 represent a *support* relation, while 140 are *attacks*. Considering only the *supports*, 111 argument pairs (i.e., 61.6%) are an actual entailment, while in 38.4% of the cases the first argument of the pair supports the second one without inferring it (e.g. **(d) - (c)** in Example 1). With respect to the *attacks*, 100 argument pairs (i.e., 71.4%) are both attack and contradiction, while only the 28.6% of the argument pairs does not contradict the arguments they are attacking, as in Example 2.

Example 2.
(e) *Coca chewing is bad for human health. The decision to ban coca chewing fifty years ago was based on a 1950 report elaborated by the UN Commission of Inquiry on the Coca Leaf with a mandate from ECOSOC: "We believe that the daily, inveterate use of coca leaves by chewing is thoroughly noxious and therefore detrimental".*

(f) *Chewing coca offers an energy boost. Coca provides an energy boost for working or for combating fatigue and cold.*

Differently from the relation between support-entailment, the difference between attack and contradiction is more sub-

[6] In this phase, the annotators discuss the results to find an agreement on the annotation to be released.

tle, and it is not always straightforward to say whether an argument attacks another argument without contradicting it. In Example 2, we consider that **(e)** does not contradict **(f)** even if it attacks **(f)**, since chewing coca can offer an energy boost, and still be bad for human health. This kind of attacks is less frequent than the attacks-contradictions (see Table 3).

Debatepedia additional attacks dataset Starting from the comparative study addressed by (Cayrol and Lagasquie-Schiex 2011), in the third study of (Cabrio and Villata 2013) we have considered four additional attacks proposed in the literature: *supported* (if argument a supports argument b and b attacks argument c, then a attacks c) and *secondary* (if a supports b and c attacks a, then c attacks b) attacks (Cayrol and Lagasquie-Schiex 2010), *mediated* attacks (Boella et al. 2010) (if a supports b and c attacks b, then c attacks a), and *extended* attacks (Nouioua and Risch 2010; 2011) (if a supports b and a attacks c, then b attacks c).

In order to investigate the presence and the distribution of these attacks in NL debates, we extended again the data set extracted from Debatepedia to consider all these additional attacks, and we showed that all these models are verified in human debates, even if with a different frequency. More specifically, we took the original argumentation framework of each topic in our data set (Table 2), the following procedure is applied: the *supported* (secondary, mediated, and extended, respectively) attacks are added, and the argument pairs resulting from coupling the arguments linked by this relation are collected in the data set "supported (secondary, mediated, and extended, respectively) attack". Collecting the argument pairs generated from the different types of complex attacks in separate data sets allows us to independently analyze each type, and to perform a more accurate evaluation.[7] Figures 2a-d show the four AFs resulting from the addition of the complex attacks in the example *Can coca be classified as a narcotic?*. Note that the AF in Figure 2a, where the supported attack is introduced, is the same of Figure 2b where the mediated attack is introduced. Notice that, even if the additional attack which is introduced coincide, i.e., d attacks b, this is due indeed to different interactions among supports and attacks (as highlighted in the figure), i.e., in the case of supported attacks this is due to the support from d to c and the attack from c to b, while in the case of mediated attacks this is due to the support from b to a and the attack from d to a.

A second annotation phase is then carried out on the data set, to verify if the generated argument pairs of the four data sets are actually attacks (i.e., if the models of complex attacks proposed in the literature are represented in real data). More specifically, an argument pair resulting from the application of a complex attack can be annotated as: *attack* (if it is a correct attack) or as *unrelated* (in case the meanings of the two arguments are not in conflict). For instance, the argument pair **(g)-(h)** (Example 3) resulting from the insertion of a *supported* attack, cannot be considered as an attack since the arguments are considering two different aspects of the issue.

Example 3.
(g) *Chewing coca offers an energy boost. Coca provides an energy boost for working or for combating fatigue and cold.*

(h) *Coca can be classified as a narcotic.*

In the annotation, *attacks* are then annotated also as *contradiction* (if the first argument contradicts the other) or *null* (in case the first argument does not contradict the argument it is attacking, as in Example 2). Due to the complexity of the annotation, the same annotation task has been independently carried out also by a second annotator, so as to compute inter-annotator agreement. It has been calculated on a sample of 80 argument pairs (20 pairs randomly extracted from each of the "complex attacks" data set), and it has the goal to assess the validity of the annotation task (counting when the judges agree on the same annotation). We calculated the inter-annotator agreement for our annotation task in two steps. We *(i)* verify the agreement of the two judges on the argument pairs classification *attacks/unrelated*, and *(ii)* consider only the argument pairs tagged as *attacks* by both annotators, and we verify the agreement between the pairs tagged as *contradiction* and as *null* (i.e. no contradiction). Applying κ to our data, the agreement for the first step is $\kappa = 0.77$, while for the second step $\kappa = 0.71$. As a rule of thumb, both agreements are satisfactory, although they reflect the higher complexity of the second annotation (*contradiction/null*).

The distribution of complex attacks in the Debatepedia data set, as resulting after a reconciliation phase carried out by the annotators, is shown in Table 4. As can be noticed, the *mediated* attack is the most frequent type of attack, generating 335 new argument pairs in the NL sample we considered (i.e. the conditions that allow the application of this kind of complex attacks appear more frequently in real debates). Together with *secondary* attacks, they appear in the AFs of all the debated topics. On the contrary, *extended* attacks are added in 11 out of 19 topics, and *supported* attacks in 17 out of 19 topics. Considering all the topics, on average only 6 pairs generated from the additional attacks were already present in the original data set, meaning that considering also these attacks is a way to hugely enrich our data set of NL debates.

Proposed models	**# occ.**	**attacks**		**unrelated**
		+ contr (null)	*- contr (null)*	
Supported attacks	47	23	17	7
Secondary attacks	53	29	18	6
Mediated attacks	335	84	148	103
Extended attacks	28	15	10	3

Table 4: Complex attacks distribution in our data set.

Twelve Angry Men

As a second scenario to extract natural language arguments we chose the scripts of "Twelve Angry Men". The play con-

[7] Data sets freely available for research purposes at `http://www-sop.inria.fr/NoDE/NoDE-xml.html#debatepedia`

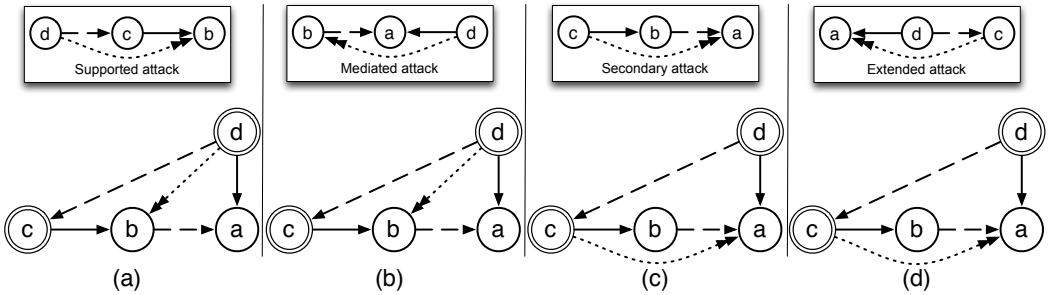


Figure 2: The bipolar argumentation framework with the introduction of complex attacks. The top figures show which combination of support and attack generates the new additional attack.

cerns the deliberations of the jury of a homicide trial. As in most American criminal cases, the twelve men must unanimously decide on a verdict of "guilty" or "not guilty". At the beginning, they have a nearly unanimous decision of guilty, with a single dissenter of not guilty, who throughout the play sows a seed of reasonable doubt.

The play is divided into three acts: the end of each act corresponds to a fixed point in time (i.e. the halfway votes of the jury, before the official one), according to which we want to be able to extract a set of consistent arguments. For each act, we manually selected the arguments (excluding sentences which cannot be considered as self-contained arguments), and we coupled each argument with the argument it is supporting or attacking in the dialogue flow (as shown in Examples 4 to 7). More specifically, in discussions, one character's argument comes after the other (entailing or contradicting one of the arguments previously expressed by another character): therefore, we create our pairs in the graph connecting the former to the latter (more recent arguments are placed as T and the argument w.r.t. whom we want to detect the relation is placed as H). For instance, in Example 6, juror 1 claims argument **(o)**, and he is attacked by juror 2, claiming argument **(l)**. Juror 3 claims then argument **(i)** to support juror's 2 opinion. In the dataset we have therefore annotated the following couples: **(o)** is contradicted by **(l)**; **(l)** is entailed by **(i)**.

In Example 7, juror 1 claims argument **(l)** supported by juror 2 (argument **(i)**); juror 3 attacks juror's 2 opinion with argument **(p)**. More specifically, **(l)** is entailed by **(i)**; **(i)** is contradicted by **(p)**.

Example 4.
(i) *Maybe the old man didn't hear the boy yelling "I'm going to kill you". I mean with the el noise.*
(l) *I don't think the old man could have heard the boy yelling.*

Example 5.
(m) *I never saw a guiltier man in my life. You sat right in court and heard the same thing I did. The man's a dangerous killer.*
(n) *I don't know if he is guilty.*

Example 6.
(i) *Maybe the old man didn't hear the boy yelling "I'm going to kill you". I mean with the el noise.*
(l) *I don't think the old man could have heard the boy yelling.*
(o) *The old man said the boy yelled "I'm going to kill you" out. That's enough for me.*

Example 7.
(p) *The old man cannot be a liar, he must have heard the boy yelling.*
(i) *Maybe the old man didn't hear the boy yelling "I'm going to kill you". I mean with the el noise.*
(l) *I don't think the old man could have heard the boy yelling.*

Given the complexity of the play, and the fact that in human linguistic interactions a lot is left implicit, we simplified the arguments: *i)* adding the required context in T to make the pairs self-contained (in the TE framework entailment is detected based on the evidences provided in T); and *ii)* solving intra document coreferences, as in: *Nobody has to prove that!*, transformed into *Nobody has to prove [that he is not guilty]*.

We collected 80 T-H pairs[8], composed by 25 entailment pairs, 41 contradiction and 14 unknown pairs (contradiction and unknown pairs are then collapsed in the judgment *non entailment* for the two-way classification task).[9] To calculate the inter annotator agreement, the same annotation task has been independently carried out on half of argument pairs (40 T-H pairs) also by a second annotator. Cohen's kappa (Carletta 1996) is 0.74. Again, this is a satisfactory agreement, confirming the reliability of the obtained resource.

Also in this scenario, we consider the pairs annotated in the first layer and we then build a bipolar entailment graph for each of the topic in the dataset (the three acts of the play). Again, the arguments are the nodes of the graph, and the relations among the arguments correspond to the edges of the graphs. The complexity of the graphs obtained for the Twelve Angry Men scenario is higher than the debates graphs (on average, 27 links per graph with respect to 9 links per graph in the Debatepedia dataset).

[8]The dataset is available at `http://www-sop.inria.fr/NoDE/NoDE-xml.html#12AngryMen`. It is built in standard RTE format.

[9]The unknown pairs in the dataset are arguments attacking each others, without contradicting. Collapsing both judgments into one category for our experiments does not impact on our framework evaluation.

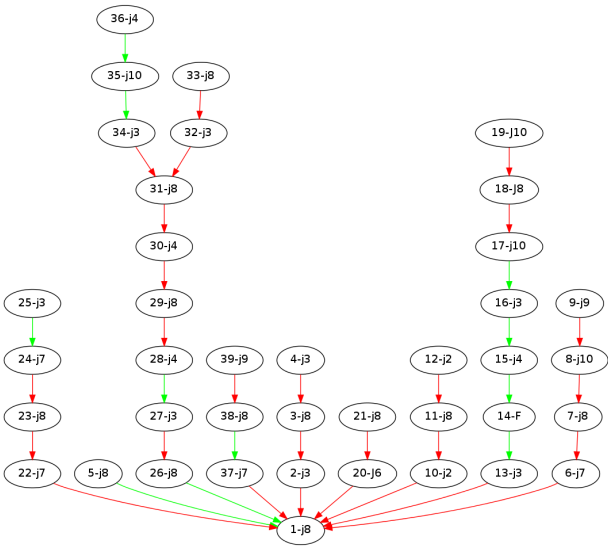


Figure 3: The bipolar argumentation framework resulting from Act 1 of Twelve Angry Men (red edges represent attack and green ones represent support).

Figure 3 shows the average dimension of a bipolar argumentation graph in the Twelve Angry Men dataset. Note that no cycle is present, as well as in all the other graphs of such dataset.

Conclusions

In this paper, we describe two datasets of natural language arguments used in the context of debates. The only existing dataset composed of natural language arguments proposed and exploited in the argumentation community is Araucaria.[10] Araucaria (Reed and Rowe 2004) is based on argumentation schemes (Walton, Reed, and Macagno 2008), and it is an online repository of arguments from heterogenous sources like newspapers (e.g., Wall Street Journal), parliamentary records (e.g., UK House of Parliament debates) and discussion fora (e.g., BBC talking point). Arguments are classified by argumentation schemes. Also in the context of argumentation schemes, (Cabrio, Tonelli, and Villata 2013) propose a new resource based on the Penn Discourse Treebank (PDTB), where a part of the corpus has been annotated with a selection of five argumentation schemes. This effort goes in the direction of trying to export a well known existing benchmark in the field of natural language processing (i.e., PDTB) into the argumentation field, through the identification and annotation of the argumentation schemes.

The benchmark of natural language arguments we presented in this paper has several potential uses. As all the data we presented is available on the Web in a machine-readable format, researchers interested in testing their own argumentation-based tool (both for arguments visualization and for reasoning) are allowed to download the data sets and verify on real data the performances of the tool. Moreover, also from the theoretical point of view, the data set can be used by argumentation researchers to find real world example supporting the introduction of new theoretical frameworks. One of the aims of such benchmark is actually to move from artificial natural language examples of argumentation towards more realistic ones where other problems, maybe far from the ones addressed at the present stage in current argumentation research, emerge.

It is interesting to note that the abstract (bipolar) argumentation graphs resulting from our datasets result to be rather simple structures, where usually arguments are inserted in reinstatement chains, rather than complex structures with the presence of several odd and even cycles, as usually challenged in the argumentation literature. In this perspective, we plan to consider other sources of arguments, like costumer's opinions about a service or a product, to see whether more complex structures are identified, with the final goal to built a complete resource where also such complex patterns are present.

A further point which deserves investigation concerns the use of abstract argumentation. Some of the examples we provided may suggest that in some cases adopting abstract argumentation might not be fully appropriate since such natural language arguments have (possibly complex) internal structures and may include sub-arguments (for example argument *(d)* of the "Coca as narcotic" example). We will investigate how to build a dataset of structured arguments, taking into account the discourse relations.

Finally, in this paper, we have presented a benchmark of natural language arguments manually annotated by humans with skills in linguistics. Given the complexity of the annotation task, a manual annotation was the best choice ensuring an high quality of the data sets. However, in other tasks like discourse relations extraction, it is possible to adopt automated extraction techniques then further verified by human annotators to ensure an high resource's confidence.

[10] `http://araucaria.computing.dundee.ac.uk`

References

Boella, G.; Gabbay, D. M.; van der Torre, L.; and Villata, S. 2010. Support in abstract argumentation. In *Procs of COMMA, Frontiers in Artificial Intelligence and Applications 216*, 111–122.

Cabrio, E., and Villata, S. 2012. Natural language arguments: A combined approach. In *Procs of ECAI, Frontiers in Artificial Intelligence and Applications 242*, 205–210.

Cabrio, E., and Villata, S. 2013. A natural language bipolar argumentation approach to support users in online debate interactions;. *Argument & Computation* 4(3):209–230.

Cabrio, E.; Tonelli, S.; and Villata, S. 2013. A natural language account for argumentation schemes. In Baldoni, M.; Baroglio, C.; Boella, G.; and Micalizio, R., eds., *AI*IA*, volume 8249 of *Lecture Notes in Computer Science*, 181–192. Springer.

Carletta, J. 1996. Assessing agreement on classification tasks: the kappa statistic. *Comput. Linguist.* 22(2):249–254.

Cayrol, C., and Lagasquie-Schiex, M.-C. 2005. On the acceptability of arguments in bipolar argumentation frameworks. In *Procs of ECSQARU, LNCS 3571*, 378–389.

Cayrol, C., and Lagasquie-Schiex, M.-C. 2010. Coalitions of arguments: A tool for handling bipolar argumentation frameworks. *Int. J. Intell. Syst.* 25(1):83–109.

Cayrol, C., and Lagasquie-Schiex, M.-C. 2011. Bipolarity in argumentation graphs: Towards a better understanding. In *Procs of SUM, LNCS 6929*, 137–148.

Dagan, I.; Dolan, B.; Magnini, B.; and Roth, D. 2009. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering (JNLE)* 15(04):i–xvii.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321–358.

Nouioua, F., and Risch, V. 2010. Bipolar argumentation frameworks with specialized supports. In *Procs of ICTAI*, 215–218. IEEE Computer Society.

Nouioua, F., and Risch, V. 2011. Argumentation frameworks with necessities. In *Procs of SUM, LNCS 6929*, 163–176.

Reed, C., and Rowe, G. 2004. Araucaria: Software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools* 13(4):961–980.

Walton, D.; Reed, C.; and Macagno, F. 2008. *Argumentation Schemes*. Cambridge University Press.

Analysis of Dialogical Argumentation via Finite State Machines

Anthony Hunter
Department of Computer Science,
University College London,
Gower Street, London WC1E 6BT, UK

Abstract

Dialogical argumentation is an important cognitive activity by which agents exchange arguments and counterarguments as part of some process such as discussion, debate, persuasion and negotiation. Whilst numerous formal systems have been proposed, there is a lack of frameworks for implementing and evaluating these proposals. First-order executable logic has been proposed as a general framework for specifying and analysing dialogical argumentation. In this paper[1], we investigate how we can implement systems for dialogical argumentation using propositional executable logic. Our approach is to present and evaluate an algorithm that generates a finite state machine that reflects a propositional executable logic specification for a dialogical argumentation together with an initial state. We also consider how the finite state machines can be analysed, with the minimax strategy being used as an illustration of the kinds of empirical analysis that can be undertaken.

Introduction

Dialogical argumentation involves agents exchanging arguments in activities such as discussion, debate, persuasion, and negotiation (Besnard and Hunter 2008). Dialogue games are now a common approach to characterizing argumentation-based agent dialogues (e.g. (Amgoud, Maudet, and Parsons 2000; Black and Hunter 2009; Dignum, Dunin-Keplicz, and Verbrugge 2000; Fan and Toni 2011; Hamblin 1971; Mackenzie 1979; McBurney and Parsons 2002; McBurney et al. 2003; Parsons, Wooldridge, and Amgoud 2003; Prakken 2005; Walton and Krabbe 1995)). Dialogue games are normally made up of a set of communicative acts called moves, and a protocol specifying which moves can be made at each step of the dialogue. In order to compare and evaluate dialogical argumentation systems, we proposed in a previous paper that first-order executable logic could be used as common theoretical framework to specify and analyse dialogical argumentation systems (Black and Hunter 2012).

In this paper, we explore the implementation of dialogical argumentation systems in executable logic. For this, we focus on propositional executable logic as a special case, and investigate how a finite state machine (FSM) can be generated as a representation of the possible dialogues that can emanate from an initial state. The FSM is a useful structure for investigating various properties of the dialogue, including conformance to protocols, and application of strategies. We provide empirical results on generating FSMs for dialogical argumentation, and how they can be analysed using the minimax strategy. We demonstrate through preliminary implementation that it is computationally viable to generate the FSMs and to analyse them. This has wider implications in using executable logic for applying dialogical argumentation in practical uncertainty management applications, since we can now empirically investigate the performance of the systems in handling inconsistency in data and knowledge.

[1]This paper has already been published in the Proceedings of the International Conference on Scalable Uncertainty Management (SUM'13), LNCS 8078, Pages 1-14, Springer, 2013.

Propositional executable logic

In this section, we present a propositional version of the executable logic which we will show is amenable to implementation. This is a simplified version of the framework for first-order executable logic in (Black and Hunter 2012).

We assume a set of atoms which we use to form propositional formulae in the usual way using disjunction, conjunction, and negation connectives. We construct modal formulae using the $\boxplus$, $\boxminus$, $\oplus$, and $\ominus$ modal operators. We only allow literals to be in the scope of a modal operator. If α is a literal, then each of $\oplus\alpha$, $\ominus\alpha$, $\boxplus\alpha$, and $\boxminus\alpha$ is an **action unit**. Informally, we describe the meaning of action units as follows: $\oplus\alpha$ means that the action by an agent is to add the literal α to its next private state; $\ominus\alpha$ means that the action by an agent is to delete the literal α from its next private state; $\boxplus\alpha$ means that the action by an agent is to add the literal α to the next public state; and $\boxminus\alpha$ means that the action by an agent is to delete the literal α from the next public state.

We use the action units to form **action formulae** as follows using the disjunction and conjunction connectives: (1) If ϕ is an action unit, then ϕ is an action formula; And (2) If α and β are action formulae, then $\alpha \vee \beta$ and $\alpha \wedge \beta$ are action formulae. Then, we define the action rules as follows: If ϕ is a classical formula and ψ is an action formula then $\phi \Rightarrow \psi$ is an **action rule**. For instance, $\mathtt{b}(\mathtt{a}) \Rightarrow \boxplus\mathtt{c}(\mathtt{a})$ is an action rule (which we might use in an example where b denotes belief, and c denotes claim, and a is some information).

Implicit in the definitions for the language is the fact that we can use it as a meta-language (Wooldridge, McBurney,

and Parsons 2005). For this, the object-language will be represented by terms in this meta-language. For instance, the object-level formula $\mathtt{p(a,b)} \rightarrow \mathtt{q(a,b)}$ can be represented by a term where the object-level literals $\mathtt{p(a,b)}$ and $\mathtt{q(a,b)}$ are represented by constant symbols, and $\rightarrow$ is represented by a function symbol. Then we can form the atom $\mathtt{belief(p(a,b)} \rightarrow \mathtt{q(a,b))}$ where belief is a predicate symbol. Note, in general, no special meaning is ascribed the predicate symbols or terms. They are used as in classical logic. Also, the terms and predicates are all ground, and so it is essentially a propositional language.

We use a state-based model of dialogical argumentation with the following definition of an execution state. To simplify the presentation, we restrict consideration in this paper to two agents. An execution represents a finite or infinite sequence of execution states. If the sequence is finite, then t denotes the terminal state, otherwise $t = \infty$.

Definition 1 *An* **execution** e *is a tuple* $e = (s_1, a_1, p, a_2, s_2, t)$*, where for each* $n \in \mathbb{N}$ *where* $0 \leq n \leq t$*,* $s_1(n)$ *is a set of ground literals,* $a_1(n)$ *is a set of ground action units,* $p(n)$ *is a set of ground literals,* $a_2(n)$ *is a set of ground action units,* $s_2(n)$ *is a set of ground literals, and* $t \in \mathbb{N} \cup \{\infty\}$*. For each* $n \in \mathbb{N}$*, if* $0 \leq n \leq t$*, then an* **execution state** *is* $e(n) = (s_1(n), a_1(n), p(n), a_2(n), s_2(n))$ *where* $e(0)$ *is the* **initial state***. We assume* $a_1(0) = a_2(0) = \emptyset$*. We call* $s_1(n)$ *the private state of agent 1 at time* n*,* $a_1(n)$ *the action state of agent 1 at time* n*,* $p(n)$ *the public state at time* n*,* $a_2(n)$ *the action state of agent 2 at time* n*,* $s_2(n)$ *the private state of agent 2 at time* n*.*

In general, there is no restriction on the literals that can appear in the private and public state. The choice depends on the specific dialogical argumentation we want to specify. This flexibility means we can capture diverse kinds of information in the private state about agents by assuming predicate symbols for their own beliefs, objectives, preferences, arguments, etc, and for what they know about other agents. The flexibility also means we can capture diverse information in the public state about moves made, commitments made, etc.

Example 1 *The first 5 steps of an infinite execution where each row in the table is an execution state where* b *denotes belief, and* c *denotes claim.*

n	$s_1(n)$	$a_1(n)$	$p(n)$	$a_2(n)$	$s_2(n)$
0	$\mathtt{b(a)}$				$\mathtt{b}(\neg\mathtt{a})$
1	$\mathtt{b(a)}$	$\boxplus\mathtt{c(a)}$ $\boxminus\mathtt{c}(\neg\mathtt{a})$			$\mathtt{b}(\neg\mathtt{a})$
2	$\mathtt{b(a)}$		$\mathtt{c(a)}$	$\boxplus\mathtt{c}(\neg\mathtt{a})$ $\boxminus\mathtt{c(a)}$	$\mathtt{b}(\neg\mathtt{a})$ $\mathtt{b}(\neg\mathtt{a})$
3	$\mathtt{b(a)}$	$\boxplus\mathtt{c(a)}$ $\boxminus\mathtt{c}(\neg\mathtt{a})$	$\mathtt{c}(\neg\mathtt{a})$		$\mathtt{b}(\neg\mathtt{a})$
4	$\mathtt{b(a)}$		$\mathtt{c(a)}$	$\boxplus\mathtt{c}(\neg\mathtt{a})$ $\boxminus\mathtt{c(a)}$	$\mathtt{b}(\neg\mathtt{a})$
5	...	...	...	...	...

We define a system in terms of the action rules for each agent, which specify what moves the agent can potentially make based on the current state of the dialogue. In this paper, we assume agents take turns, and at each time point the actions are from the head of just one rule (as defined in the rest of this section).

Definition 2 *A* **system** *is a tuple* $(Rules_x, Initials)$ *where* $Rules_x$ *is the set of action rules for agent* $x \in \{1, 2\}$*, and* $Initials$ *is the set of initial states.*

Given the current state of an execution, the following definition captures which rules are fired. For agent x, these are the rules that have the condition literals satisfied by the current private state $s_x(n)$ and public state $p(n)$. We use classical entailment, denoted $\models$, for satisfaction, but other relations could be used (e.g. Belnap's four valued logic). In order to relate an action state in an execution with an action formula, we require the following definition.

Definition 3 *For an action state* $a_x(n)$*, and an action formula* ϕ*,* $a_x(n)$ **satisfies** ϕ*, denoted* $a_x(n) \mid\sim \phi$*, as follows.*

1. $a_x(n) \mid\sim \alpha$ *iff* $\alpha \in a_x(n)$ *when* α *is an action unit*
2. $a_x(n) \mid\sim \alpha \wedge \beta$ *iff* $a_x(n) \mid\sim \alpha$ *and* $a_x(n) \mid\sim \beta$
3. $a_x(n) \mid\sim \alpha \vee \beta$ *iff* $a_x(n) \mid\sim \alpha$ *or* $a_x(n) \mid\sim \beta$

For an action state $a_x(n)$*, and an action formula* ϕ*,* $a_x(n)$ **minimally satisfies** ϕ*, denoted* $a_x(n) \Vdash \phi$*, iff* $a_x(n) \mid\sim \phi$ *and for all* $X \subset a_x(n)$*,* $X \not\mid\sim \phi$*.*

Example 2 *Consider the execution in Example 1. For agent 1 at n = 1, we have* $a_1(1) \Vdash \boxplus\mathtt{c(a)} \wedge \boxminus\mathtt{c}(\neg\mathtt{a})$*.*

We give two constraints on an execution to ensure that they are well-behaved. The first (propagated) ensures that each subsequent private state (respectively each subsequent public state) is the current private state (respectively current public state) for the agent updated by the actions given in the action state. The second (engaged) ensures that an execution does not have one state with no actions followed immediately by another state with no actions (otherwise the dialogue can lapse) except at the end of the dialogue where neither agent has further actions.

Definition 4 *An execution* $(s_1, a_1, p, a_2, s_2, t)$ *is* **propagated** *iff for all* $x \in \{1, 2\}$*, for all* $n \in \{0, \ldots, t-1\}$*, where* $a(n) = a_1(n) \cup a_2(n)$

1. $s_x(n+1) = (s_x(n) \setminus \{\phi \mid \ominus\phi \in a_x(n)\}) \cup \{\phi \mid \oplus\phi \in a_x(n)\}$
2. $p(n+1) = (p(n) \setminus \{\phi \mid \boxminus\phi \in a(n)\}) \cup \{\phi \mid \boxplus\phi \in a(n)\}$

Definition 5 *Let* $e = (s_1, a_1, p, a_2, s_2, t)$ *be an execution and* $a(n) = a_1(n) \cup a_2(n)$*.* e *is* **finitely engaged** *iff (1)* $t \neq \infty$*; (2) for all* $n \in \{1, \ldots, t-2\}$*, if* $a(n) = \emptyset$*, then* $a(n+1) \neq \emptyset$ *(3)* $a(t-1) = \emptyset$*; and (4)* $a(t) = \emptyset$*.* e *is* **infinitely engaged** *iff (1)* $t = \infty$*; and (2) for all* $n \in \mathbb{N}$*, if* $a(n) = \emptyset$*, then* $a(n+1) \neq \emptyset$*.*

The next definition shows how a system provides the initial state of an execution and the actions that can appear in an execution. It also ensures turn taking by the two agents.

Definition 6 *Let* $S = (Rules_x, Initials)$ *be a system and* $e = (s_1, a_1, p, a_2, s_2, t)$ *be an execution.* S **generates** e *iff (1)* e *is propogated; (2)* e *is finitely engaged or infinitely engaged; (3)* $e(0) \in Initials$*; and (4) for all* $m \in \{1, \ldots, t-1\}$

1. *If m is odd, then $a_2(m) = \emptyset$ and either $a_1(m) = \emptyset$ or there is an $\phi \Rightarrow \psi \in Rules_1$ s.t. $s_1(m) \cup p(m) \models \phi$ and $a_1(m) \Vdash \psi$*
2. *If m is even, then $a_1(m) = \emptyset$ and either $a_2(m) = \emptyset$ or there is an $\phi \Rightarrow \psi \in Rules_2$ s.t. $s_1(m) \cup p(m) \models \phi$ and $a_2(m) \Vdash \psi$*

Example 3 *We can obtain the execution in Example 1 with the following rules: (1)* $\mathtt{b(a)} \Rightarrow \boxplus\mathtt{c(a)} \wedge \boxminus\mathtt{c}(\neg\mathtt{a})$*; And (2)* $\mathtt{b}(\neg\mathtt{a}) \Rightarrow \boxplus\mathtt{c}(\neg\mathtt{a}) \wedge \boxminus\mathtt{c(a)}$.

Generation of finite state machines

In (Black and Hunter 2012), we showed that for any executable logic system with a finite set of ground action rules, and an initial state, there is an FSM that consumes exactly the finite execution sequences of the system for that initial state. That result assumes that each agent makes all its possible actions at each step of the execution. Also that result only showed that there exist these FSMs, and did not give any way of obtaining them.

In this paper, we focus on propositional executable logic where the agents take it in turn, and only one head of one action rule is used, and show how we can construct an FSM that represents the set of executions for an initial state for a system. For this, each state is a tuple $(r, s_1(n), p(n), s_2(n))$, and each letter in the alphabet is a tuple $(a_1(n), a_2(n))$, where n is an execution step and r is the agent holding the turn when $n < t$ and r is 0 when $n = t$.

Definition 7 *A finite state machine (FSM)* $M = (States, Trans, Start, Term, Alphabet)$ **represents** *a system* $S = (Rules_x, Initials)$ *for an initial state* $I \in Initials$ *iff*

$(1) States = \{(y, s_1(n), p(n), s_2(n)) \mid$
there is an execution $e = (s_1, a_1, p, a_2, s_2, t)$
s.t. S *generates* e
and $I = (s_1(0), a_1(0), p(0), a_2(0), s_2(0))$
and there is an $n \leq t$
s.t. $y = 0$ *when* $n = t$
and $y = 1$ *when* $n < t$ *and* n *is odd*
and $y = 2$ *when* $n < t$ *and* n *is even* $\}$

$(2) Term = \{(y, s_1(n), p(n), s_2(n)) \in States \mid y = 0\}$

$(3) Alphabet = \{(a_1(n), a_2(n)) \mid$ *there is an* $n \leq t$
and there is an execution e
s.t. S *generates* e
and $e(0) = I$
and $e = (s_1, a_1, p, a_2, s_2, t)\}$

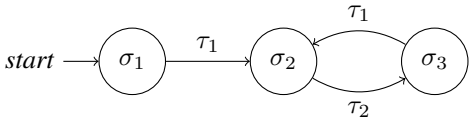
$(4) Start = (1, s_1(0), p(0), s_2(0))$
where $I = (s_1(0), a_1(0), p(0), a_2(0), s_2(0))$

(5)*Trans is the smallest subset of* $States \times Alphabet \times States$ *s.t. for all executions* e *and for all* $n < t$ *there is a transition* $(\sigma_1, \tau, \sigma_2) \in Trans$ *such that*

$$\sigma_1 = (x, s_1(n), p(n), s_2(n))$$
$$\tau = (a_1(n), a_2(n))$$
$$\sigma_2 = (y, s_1(n+1), p(n+1), s_2(n+1))$$

where x is 1 when n is odd, x is 2 when n is even, y is 1 when $n + 1 < t$ and n is odd, y is 2 when $n + 1 < t$ and n is even, and y is 0 when $n + 1 = t$.

Example 4 *Let M be the following FSM where* $\sigma_1 = (1, \{\mathtt{b(a)}\}, \{\}, \{\mathtt{b}(\neg\mathtt{a})\})$; $\sigma_2 = (2, \{\mathtt{b(a)}\}, \{c(a)\}, \{b(\neg a)\})$; $\sigma_3 = (1, \{\mathtt{b(a)}\}, \{\mathtt{c}(\neg\mathtt{a})\}, \{b(\neg a)\})$. $\tau_1 = (\{\boxplus\mathtt{c(a)}, \boxminus\mathtt{c}(\neg\mathtt{a})\}, \emptyset)$*; and* $\tau_2 = (\emptyset, \{\boxplus\mathtt{c}(\neg\mathtt{a}), \boxminus\mathtt{c(a)}\})$. *$M$ represents the system in Ex 1.*



Proposition 1 *For each $S = (Rules_x, Initials)$, then there is an FSM M such that M represents S for an initial state $I \in Initials$.*

Definition 8 *A string ρ* **reflects** *an execution $e = (s_1, a_1, p, a_2, s_2, t)$ iff ρ is the string $\tau_1 \ldots \tau_{t-1}$ and for each $1 \leq n < t$, τ_n is the tuple $(a_1(n), a_2(n))$.*

Proposition 2 *Let $S = (Rules_x, Initials)$ be a system. and let M be an FSM that represents S for $I \in Initials$.*

1. *for all ρ s.t. M accepts ρ, there is an e s.t. S generates e and $e(0) = I$ and ρ reflects e,*
2. *for all finite e s.t. S generates e and $e(0) = I$, then there is a ρ such that M accepts ρ and ρ reflects e.*

So for each initial state for a system, we can obtain an FSM that is a concise representation of the executions of the system for that initial state. In Figure 3, we provide an algorithm for generating these FSMs. We show correctness for the algorithm as follows.

Proposition 3 *Let $S = (Rules_x, Initials)$ be a system and let $I \in Initials$. If M represents S w.r.t. I and* $\mathsf{BuildMachine}(Rules_x, I) = M'$*, then $M = M'$.*

An FSM provides a more efficient representation of all the possible executions than the set of executions for an initial state. For instance, if there is a set of states that appear in some permutation of each of the executions then this can be more compactly represented by an FSM. And if there are infinite sequences, then again this can be more compactly represented by an FSM.

Once we have an FSM of a system with an initial state, we can ask obvious simple questions such as is termination possible, is termination guaranteed, and is one system subsumed by another? So by translating a system into an FSM, we can harness substantial theory and tools for analysing FSMs.

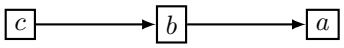
Next we give a couple of very simple examples of FSMs obtained from executable logic. In these examples, we assume that agent 1 is trying to win an argument with agent 2. We assume that agent 1 has a goal. This is represented by the predicate $\mathtt{g(c)}$ in the private state of agent 1 for some argument $\mathtt{c}$. In its private state, each agent has zero or more arguments represented by the predicate $\mathtt{n(c)}$, and zero or more attacks $\mathtt{e(d, c)}$ from $\mathtt{d}$ to $\mathtt{c}$. In the public state, each argument $\mathtt{c}$ is represented by the predicate $\mathtt{a(c)}$. Each agent can add attacks $\mathtt{e(d, c)}$ to the public state, if the attacked argument is already in the public state (i.e. $\mathtt{a(c)}$ is in the public

state), and the agent also has the attacker in its private state (i.e. $\mathtt{n(d)}$ is in the private state). We have encoded the rules so that after an argument has been used as an attacker, it is removed from the private state of the agent so that it does not keep firing the action rule (this is one of a number of ways that we can avoid repetition of moves).

Example 5 *For the following action rules, with the initial state where the private state of agent 1 is* $\{\mathtt{g(a)}, \mathtt{n(a)}, \mathtt{n(c)}, \mathtt{e(c,b)}\}$*, the public state is empty, and the private state of agent 2 is* $\{\mathtt{n(b)}, \mathtt{e(b,a)}\}$*), we get the FSM in Figure 1.*

$$\mathtt{g(a)} \wedge \mathtt{n(a)} \Rightarrow \boxplus\mathtt{a(a)} \wedge \ominus\mathtt{n(a)}$$
$$\mathtt{a(a)} \wedge \mathtt{n(b)} \wedge \mathtt{e(b,a)} \Rightarrow \boxplus\mathtt{a(b,a)} \wedge \ominus\mathtt{n(b)}$$
$$\mathtt{a(b)} \wedge \mathtt{n(c)} \wedge \mathtt{e(c,b)} \Rightarrow \boxplus\mathtt{a(c,b)} \wedge \ominus\mathtt{n(c)}$$

The terminal state therefore contains the following argument graph.

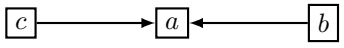


Hence the goal argument $\mathtt{a}$ *is in the grounded extension of the graph (as defined in (Dung 1995)).*

Example 6 *For the following action rules, with the initial state where the private state of agent 1 is* $\{\mathtt{g(a)}, \mathtt{n(a)}\}$*, the public state is empty, and the private state of agent 2 is* $\{\mathtt{n(b)}, \mathtt{n(c)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\}$*), we get the FSM in Figure 2*

$$\mathtt{g(a)} \wedge \mathtt{n(a)} \Rightarrow \boxplus\mathtt{a(a)} \wedge \ominus\mathtt{n(a)}$$
$$\mathtt{a(a)} \wedge \mathtt{n(b)} \wedge \mathtt{e(b,a)} \Rightarrow \boxplus\mathtt{a(b,a)} \wedge \ominus\mathtt{n(b)}$$
$$\mathtt{a(a)} \wedge \mathtt{n(c)} \wedge \mathtt{e(c,a)} \Rightarrow \boxplus\mathtt{a(c,a)} \wedge \ominus\mathtt{n(c)}$$

The terminal state therefore contains the following argument graph.



Hence the goal argument $\mathtt{a}$ *is in the grounded extension of the graph.*

In the above examples, we have considered a formalisation of dialogical argumentation where agents exchange abstract arguments and attacks. It is straightforward to formalize other kinds of example to exchange a wider range of moves, richer content (e.g. logical arguments composed of premises and conclusion (Parsons, Wooldridge, and Amgoud 2003)), and richer notions (e.g. value-based argumentation (Bench-Capon 2003)).

Minimax analysis of finite state machines

Minimax analysis is applied to two-person games for deciding which moves to make. We assume two players called MIN and MAX. MAX moves first, and they take turns until the game is over. An **end function** determines when the game is over. Each state where the game has ended is an **end state**. A **utility function** (i.e. a payoff function) gives the outcome of the game (eg chess has win, draw, and loose). The **minimax strategy** is that MAX aims to get to an end state that maximizes its utility regardless of what MIN does

We can apply the minimax strategy to the FSM machines generated for dialogical argumentation as follows: (1) Undertake breadth-first search of the FSM; (2) Stop searching at a node on a branch if the node is an end state according to the end function (note, this is not necessarily a terminal state in the FSM); (3) Apply the utility function to each leaf node n (i.e. to each end state) in the search tree to give the value $value(n)$ of the node; (4) Traverse the tree in post-order, and calculate the value of each non-leaf node as follows where the non-leaf node n is at depth d and with children $\{n_1, .., n_k\}$:

- If d is odd, then $value(n)$ is the maximum of $value(n_1)$,.., $value(n_k)$.
- If d is even, then $value(n)$ is the minimum of $value(n_1)$,.., $value(n_k)$.

There are numerous types of dialogical argumentation that can be modelled using propositional executable logic and analysed using the minimax strategy. Before we discuss some of these options, we consider some simple examples where we assume that the search tree is exhaustive, (so each branch only terminates when it reaches a terminal state in the FSM), and the utility function returns 1 if the goal argument is in the grounded extension of the graph in the terminal state, and returns 0 otherwise.

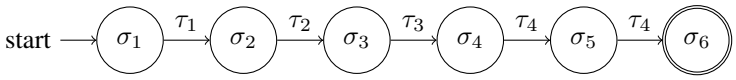
Example 7 *From the FSM in Example 5, we get the minimax search tree in Figure 5a, and from the FSM in Example 6, we get the minimax search tree in Figure 5b. In each case, the terminal states contains an argument graph in which the goal argument is in the grounded extension of the graph. So each leaf of the minimax tree has a utility of 1, and each non-node has the value 1. Hence, agent 1 is guaranteed to win each dialogue whatever agent 2 does.*

The next example is more interesting from the point of view of using the minimax strategy since agent 1 has a choice of what moves it can make and this can affect whether or not it wins.

Example 8 *In this example, we assume agent 1 has two goals* $\mathtt{a}$ *and* $\mathtt{b}$*, but it can only present arguments for one of them. So if it makes the wrong choice it can loose the game. The executable logic rules are given below and the resulting FSM is given in Figure 4. For the minimax tree (given in Figure 5c) the left branch results in an argument graph in which the goal is not in the grounded extension, whereas the right branch terminates in an argument graph in which the goal is in the grounded extension. By a minimax analysis, agent 1 wins.*

$$\mathtt{g(a)} \wedge \mathtt{n(a)} \Rightarrow \boxplus\mathtt{a(a)} \wedge \ominus\mathtt{n(a)} \wedge \ominus\mathtt{g(b)}$$
$$\mathtt{g(b)} \wedge \mathtt{n(b)} \Rightarrow \boxplus\mathtt{a(b)} \wedge \ominus\mathtt{n(b)} \wedge \ominus\mathtt{g(a)}$$
$$\mathtt{a(a)} \wedge \mathtt{n(c)} \wedge \mathtt{e(c,a)} \Rightarrow \boxplus\mathtt{a(c,a)} \wedge \ominus\mathtt{n(c)}$$

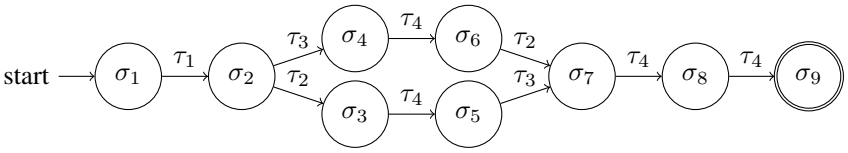
We can use any criterion for identifying the end state. In the above, we have used the **exhaustive end function** giving an end state (i.e. the leaf node in the search tree) which is a terminal state in the FSM followed by two empty transitions. If the branch does not come to a terminal state in the FSM, then it is an infinite branch. We could use a **non-repetitive end function** where the search tree stops when there are no new nodes to visit. For instance, for example 4, we could use the non-repetitive end function to give a search tree that contains one branch $\sigma_1, \sigma_2, \sigma_3$ where σ_1 is the root and σ_3 is



$$\sigma_1 = (1, \{\mathtt{g(a)}, \mathtt{n(a)}, \mathtt{n(c)}, \mathtt{e(c,b)}\}, \{\}, \{\mathtt{n(b)}, \mathtt{e(b,a)}\})$$
$$\sigma_2 = (2, \{\mathtt{g(a)}, \mathtt{n(c)}, \mathtt{e(c,b)}\}, \{\mathtt{a(a)}\}, \{\mathtt{n(b)}, \mathtt{e(b,a)}\})$$
$$\sigma_3 = (1, \{\mathtt{g(a)}, \mathtt{n(c)}, \mathtt{e(c,b)}\}, \{\mathtt{a(a)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}\})$$
$$\sigma_4 = (2, \{\mathtt{g(a)}, \mathtt{e(c,b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,b)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}\})$$
$$\sigma_5 = (1, \{\mathtt{g(a)}, \mathtt{e(c,b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,b)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}\})$$
$$\sigma_6 = (0, \{\mathtt{g(a)}, \mathtt{e(c,b)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,b)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}\})$$

$$\tau_1 = (\{\boxplus\mathtt{a(a)}, \ominus\mathtt{n(a)}\}, \emptyset)$$
$$\tau_2 = (\emptyset, \{\boxplus\mathtt{a(b,a)}, \ominus\mathtt{n(b)}\})$$
$$\tau_3 = (\{\boxplus\mathtt{a(c,b)}, \ominus\mathtt{n(c)}\}, \emptyset)$$
$$\tau_4 = (\emptyset, \emptyset)$$

Figure 1: The FSM for Example 5



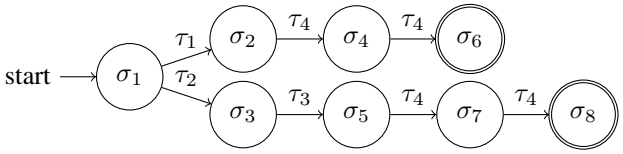
$$\sigma_1 = (1, \{\mathtt{g(a)}, \mathtt{n(a)}\}, \{\}, \{\mathtt{n(b)}, \mathtt{n(c)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_2 = (2, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}\}, \{\mathtt{n(b)}, \mathtt{n(c)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_3 = (1, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(b,a)}\}, \{\mathtt{n(c)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_4 = (1, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(c)}, \mathtt{a(c,a)}\}, \{\mathtt{n(b)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_5 = (2, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(b,a)}\}, \{\mathtt{n(c)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_6 = (2, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(c)}, \mathtt{a(c,a)}\}, \{\mathtt{n(b)}, \mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_7 = (1, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,a)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_8 = (2, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,a)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$
$$\sigma_9 = (0, \{\mathtt{g(a)}\}, \{\mathtt{a(a)}, \mathtt{a(b)}, \mathtt{a(c)}, \mathtt{a(c,a)}, \mathtt{a(b,a)}\}, \{\mathtt{e(b,a)}, \mathtt{e(c,a)}\})$$

$$\tau_1 = (\{\boxplus\mathtt{a(a)}, \ominus\mathtt{n(a)}\}, \emptyset))$$
$$\tau_2 = (\emptyset, \{\boxplus\mathtt{a(b,a)}, \ominus\mathtt{n(b)})$$
$$\tau_3 = (\emptyset, \{\boxplus\mathtt{a(c,a)}, \ominus\mathtt{n(c)})$$
$$\tau_4 = (\emptyset, \emptyset)$$

Figure 2: The FSM for Example 6

01 BuildMachine($Rules_x, I$)
02 $Start = (1, S_1, P, S_2)$ where $I = (S_1, A_1, P, A_2, S_2)$
03 $States_1 = NewStates_1 = \{Start\}$
04 $States_2 = Trans_1 = Trans_2 = \emptyset$
05 $x = 1, y = 2$
06 While $NewStates_x \neq \emptyset$
07 $NextStates = NextTrans = \emptyset$
08 For $(x, S_1, P, S_2) \in NewStates_x$
09 $Fired = \{\psi \mid \phi \Rightarrow \psi \in Rules_x$ and $S_x \cup P \models \phi\}$
10 If $Fired == \emptyset$
11 Then $NextTrans = NextTrans \cup \{((x, S_1, P, S_2), (\emptyset, \emptyset), (y, S_1, P, S_2))\}$
12 Else for $A \in$ Disjuncts($Fired$)
13 $NewS = S_x \setminus \{\alpha \mid \ominus\alpha \in A\} \cup \{\alpha \mid \oplus\alpha \in A\}$
14 $NewP = P \setminus \{\alpha \mid \boxminus\alpha \in A\} \cup \{\alpha \mid \boxplus\alpha \in A\}$
15 If $x == 1, NextState = (2, NewS, P, S_2)$ and $Label = (A, \emptyset)$
16 Else $NextState = (1, S_1, P, NewS)$ and $Label = (\emptyset, A)$
17 $NextStates = NextStates \cup \{NextState\}$
18 $NextTrans = NextTrans \cup \{((x, S_1, P, S_2), Label, NextState)\}$
19 If $x == 1$, then $x = 2$ and $y = 1$, else $x = 1$ and $y = 2$
20 $NewStates_x = NextStates \setminus States_x$
21 $States_x = States_x \cup NextStates$
22 $Trans_x = Trans_x \cup NextTrans$
23 $Close = \{\sigma'' \mid (\sigma, \tau, \sigma'), (\sigma', \tau, \sigma'') \in Trans_1 \cup Trans_2\}$
24 $Trans =$ MarkTrans($Trans_1 \cup Trans_2, Close$)
25 $States =$ MarkStates($States_1 \cup States_2, Close$)
26 $Term =$ MarkTerm($Close$)
27 $Alphabet = \{\tau \mid (\sigma, \tau, \sigma') \in States\}$
28 Return $(States, Trans, Start, Term, Alphabet)$

Figure 3: An algorithm for generating an FSM from a system S = $(Rules_x, Initials)$ and an initial state I. The subsidiary function Disjuncts($Fired$) is $\{\{\psi_1^1, .., \psi_{k_1}^1\}, .., \{\psi_1^i, .., \psi_{k_i}^1\} \mid ((\psi_1^1 \wedge .. \wedge \psi_{k_1}^1) \vee .. \vee (\psi_1^i \wedge .. \wedge \psi_{k_i}^1)) \in Fired)\}$. For turn-taking, for agent x, $State_x$ is the set of expanded states and $NewStates_x$ is the set of unexpanded states. Lines 02-05 set up the construction with agent 1 being the agent to expand the initial state. At lines 06-18, when it is turn of x, each unexpanded state in $NewStates_x$ is expanded by identifying the fired rules. At lines 10-11, if there are no fired rules, then the empty transition (i.e. $(\emptyset, \emptyset)$) is obtained, otherwise at lines 12-17, each disjunct for each fired rule gives a next state and transition that is added to $NextStates$ and $NextTrans$ accordingly. At lines 19-22, the turn is passed to the other agent, and $NewStates_x$, $States_x$, and $Trans_x$ updated. At line 23, the terminal states are identified from the transitions. At line 24, the MarkTrans function returns the union of the transitions for each agent but for each $\sigma = (x, S1, P, S2) \in Term$, σ is changed to $(0, S1, P, S2)$ in order to mark it as a terminal state in the FSM. At line 25, the MarkStates function returns the union of the states for each agent but for each $\sigma = (x, S1, P, S2) \in Term$, σ is changed to $(0, S1, P, S2)$, and similarly at line 26, MarkTerm function returns the set $Close$ but with each state being of the form $(0, S1, P, S2)$.



$$\sigma_1 = (1, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(a)}, \mathtt{n(b)}\}, \{\}, \{\mathtt{n(c)}, \mathtt{e(c,a)}\})$$
$$\sigma_2 = (2, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(a)}\}, \{\mathtt{a(b)}\}, \{\mathtt{n(c)}, \mathtt{e(c,a)}\})$$
$$\sigma_3 = (2, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(b)}\}, \{\mathtt{a(a)}\}, \{\mathtt{n(c)}, \mathtt{e(c,a)}\})$$
$$\sigma_4 = (1, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(a)}\}, \{\mathtt{a(b)}\}, \{\mathtt{n(c)}, \mathtt{e(c,a)}\})$$
$$\sigma_5 = (1, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(b)}\}, \{\mathtt{a(a)}, \mathtt{a(c)}, \mathtt{a(c,a)}\}, \{\mathtt{e(c,a)}\})$$
$$\sigma_6 = (0, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(a)}\}, \{\mathtt{a(b)}\}, \{\mathtt{n(c)}, \mathtt{e(c,a)}\})$$
$$\sigma_7 = (2, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(b)}\}, \{\mathtt{a(a)}, \mathtt{a(c)}, \mathtt{a(c,a)}\}, \{\mathtt{e(c,a)}\})$$
$$\sigma_8 = (0, \{\mathtt{g(a)}, \mathtt{g(b)}, \mathtt{n(b)}\}, \{\mathtt{a(a)}, \mathtt{a(c)}, \mathtt{a(c,a)}\}, \{\mathtt{e(c,a)}\})$$

$$\tau_1 = (\{\boxplus\mathtt{a(b)}, \ominus\mathtt{n(b)}, \ominus\mathtt{g(a)}\}, \emptyset)$$
$$\tau_2 = (\{\boxplus\mathtt{a(a)}, \ominus\mathtt{n(a)}, \ominus\mathtt{g(b)}\}, \emptyset)$$
$$\tau_3 = (\emptyset, \{\boxplus\mathtt{a(c,a)}, \ominus\mathtt{n(c)}\})$$
$$\tau_4 = (\emptyset, \emptyset)$$

Figure 4: The FSM for Example 8

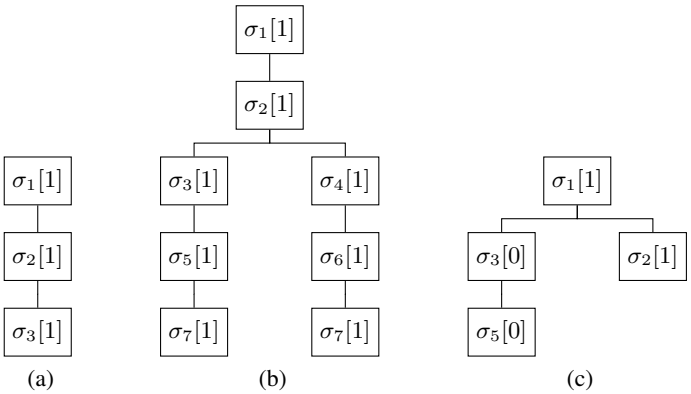


Figure 5: Minimax trees for Examples 7 and 8. Since each terminal state in an FSM is a copy of the previous two states, we save space by not giving these copies in the search tree. The minimax value for a node is given in the square brackets within the node. (a) is for Example 5, (b) is for Example 6 and (c) is for Example 8

the leaf. Another simple option is a **fixed-depth end function** which has a specified maximum depth for any branch of the search tree. More advanced options for end functions include **concession end function** when an agent has a loosing position, and it knows that it cannot add anything to change the position, then it concedes.

There is also a range of options for the utility function. In the examples, we have used grounded semantics to determine whether a goal argument is in the grounded extension of the argument graph specified in the terminal public state. A refinement is the **weighted utility function** which weights the utility assigned by the grounded utility function by $1/d$ where d is the depth of the leaf. The aim of this is to favour shorter dialogues. Further definitions for utility functions arise from using other semantics such as preferred or stable semantics and richer formalisms such as valued-based argumentation (Bench-Capon 2003).

Implementation study

In this study, we have implemented three algorithms: The generator algorithm for taking an initial state and a set of action rules for each agent, and outputting the fabricated FSM; A breadth-first search algorithm for taking an FSM and a choice of termination function, and outputting a search tree; And a minimax assignment algorithm for taking a search tree and a choice of utility function, and outputting a minimax tree. These implemented algorithms were used together so that given an initial state and rules for each agent, the overall output was a minimax tree. This could then be used to determine whether or not agent 1 had a winning strategy (given the initial state). The implementation incorporates the exhaustive termination function, and two choices of utility function (grounded and weighted grounded).

The implementation is in Python 2.6 and was run on a Windows XP PC with Intel Core 2 Duo CPU E8500 at 3.16 GHz and 3.25 GB RAM. For the evaluation, we also implemented an algorithm for generating tests inputs. Each test input comprised an initial state, and a set of action rules for each agent. Each initial state involved 20 arguments randomly assigned to the two agents and up to 20 attacks per agent. For each attack in an agent's private state, the attacker is an argument in the agent's private state, and the attacked argument is an argument in the other agent's private state. The results are presented in Table 1.

As can be seen from these results, up to about 15 attacks per agent, the implementation runs in negligible time. However, above 15 attacks per agent, the time did increase markedly, and a substantially minority of these timed out. To indicate the size of the larger FSMs, consider the last line of the table where the runs had an average of 18.02 attacks per agent: For this set, 8 out of 100 runs had 80+ nodes in the FSM. Of these 8 runs, the number of states was between 80 and 163, and the number of transitions was between 223 and 514.

The algorithm is somewhat naive in a number of respects. For instance, the algorithm for finding the grounded extension considers every subset of the set of arguments (i.e. 2^{20} sets). Clearly more efficient algorithms can be developed or calculation subcontracted to a system such as ASPARTIX (Egly, Gaggl, and Woltran 2008). Nonetheless, there are interesting applications where 20 arguments would be a reasonable, and so we have shown that we can analyse such situations successfully using the Minimax strategy, and with some refinement of the algorithms, it is likely that larger FSMs can be constructed and analysed.

Since the main aim was to show that FSMs can be generated and analysed, we only used a simple kind of argumentation dialogue. It is straightforward to develop alternative and more complex scenarios, using the language of propositional executable logic e.g. for capturing beliefs, goals, uncertainty etc, for specifying richer behaviour.

Discussion

In this paper, we have investigated a uniform way of presenting and executing dialogical argumentation systems based on a propositional executable logic. As a result different dialogical argumentation systems can be compared and implemented more easily than before. The implementation is generic in that any action rules and initial states can be used to generate the FSM and properties of them can be identified empirically.

In the examples in this paper, we have assumed that when an agent presents an argument, the only reaction the other agent can have is to present a counterargument (if it has one) from a set that is fixed in advance of the dialogue. Yet when agents argue, one agent can reveal information that can be used by the other agent to create new arguments. We illustrate this in the context of logical arguments. Here, we assume that each argument is a tuple $\langle \Phi, \psi \rangle$ where Φ is a set of formulae that entails a formula ψ. In Figure 6a, we see an argument graph instantiated with logical arguments. Suppose arguments A_1, A_3 and A_4 are presented by agent 1, and arguments A_2, A_5 and A_6 are presented by agent 2. Since agent 1 is being exhaustive in the arguments it presents, agent 2 can get a formula that it can use to create a counterargument. In Figure 6b, agent 1 is selective in the arguments it presents, and as a result, agent 2 lacks a formula in order to construct the counterarguments it needs. We can model this argumentation in propositional executable logic, generate the corresponding FSM, and provide an analysis in terms of minimax strategy that would ensure that agent 1 would provide A_4 and not A_3, thereby ensuring that it behaves more intelligently. We can capture each of these arguments as a proposition and use the minimax strategy in our implementation to obtain the tree in Figure 6b.

General frameworks for dialogue games have been proposed (Maudet and Evrard 1998; McBurney and Parsons 2002). They offer insights on dialogical argumentation systems, but they do not provide sufficient detail to formally analyse or implement specific systems. A more detailed framework, that is based on situation calculus, has been proposed by Brewka (Brewka 2001), though the emphasis is on modelling the protocols for the moves made in dialogical argumentation based on the public state rather than on strategies based on the private states of the agents.

The minimax strategy has been considered elsewhere in models of argumentation (such as for determining argument strength (Matt and Toni 2008) and for marking strategies for

Average no. attacks	Average no. FSM nodes	Average no. FSM transitions	Average no. tree nodes	Average run time	Median run time	No. of runs timed out
9.64	6.29	9.59	31.43	0.27	0.18	0
11.47	16.01	39.48	1049.14	6.75	0.18	1
13.29	12.03	27.74	973.84	9.09	0.18	2
14.96	12.50	27.77	668.65	6.41	0.19	13
16.98	19.81	49.96	2229.64	25.09	0.20	19
18.02	19.01	47.81	2992.24	43.43	0.23	30

Table 1: The results from the implementation study. Each row is produced from 100 runs. Each run (i.e. a single initial state and action rules for each agent) was timed. If the time exceeded 100 seconds for the generator algorithm, the run was terminated

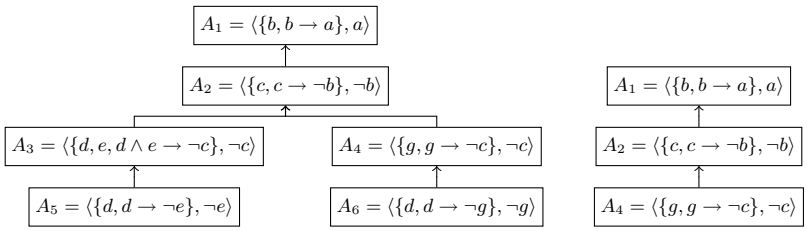


Figure 6: Consider the following knowledgebases for each agent $\Delta_1 = \{b, d, e, g, b \to a, d \wedge e \to \neg c, g \to \neg c\}$ and $\Delta_2 = \{c, c \to \neg b, d \to \neg e, d \to \neg g\}$. (a) Agent 1 is exhaustive in the arguments posited, thereby allowing agent 2 to construct arguments that cause the root to be defeated. (b)Agent is selective in the arguments posited, thereby ensuring that the root is undefeated.

dialectical trees (Rotstein, Moguillansky, and Simari 2009), for deciding on utterances in a specific dialogical argumentation (Oren and Norman 2009)). However, this paper appears to be the first empirical study of using the minimax strategy in dialogical argumentation.

In future work, we will extend the analytical techniques for imperfect games where only a partial search tree is constructed before the utility function is applied, and extend the representation with weights on transitions (e.g. weights based on tropical semirings to capture probabilistic transitions) to explore the choices of transition based on preference or uncertainty.

References

Amgoud, L.; Maudet, N.; and Parsons, S. 2000. Arguments, dialogue and negotiation. In *European Conf. on Artificial Intelligence (ECAI 2000)*, 338–342. IOS Press.

Bench-Capon, T. 2003. Persuasion in practical argument using value based argumentation frameworks. *Journal of Logic and Computation* 13(3):429–448.

Besnard, P., and Hunter, A. 2008. *Elements of Argumentation*. MIT Press.

Black, E., and Hunter, A. 2009. An inquiry dialogue system. *Autonomous Agents and Multi-Agent Systems* 19(2):173–209.

Black, E., and Hunter, A. 2012. Executable logic for dialogical argumentation. In *European Conf. on Artificial Intelligence (ECAI'12)*, 15–20. IOS Press.

Brewka, G. 2001. Dynamic argument systems: A formal model of argumentation processes based on situation calculus. *J. Logic & Comp.* 11(2):257–282.

Dignum, F.; Dunin-Keplicz, B.; and Verbrugge, R. 2000. Dialogue in team formation. In *Issues in Agent Communication*. Springer. 264–280.

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):321–357.

Egly, U.; Gaggl, S.; and Woltran, S. 2008. Aspartix: Implementing argumentation frameworks using answer-set programming. In *Proceedings of the Twenty-Fourth International Conference on Logic Programming (ICLP'08),*, volume 5366 of *LNCS*, 734–738. Springer.

Fan, X., and Toni, F. 2011. Assumption-based argumentation dialogues. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'11)*, 198–203.

Hamblin, C. 1971. Mathematical models of dialogue. *Theoria* 37:567–583.

Mackenzie, J. 1979. Question begging in non-cumulative systems. *Journal of Philosophical Logic* 8:117–133.

Matt, P., and Toni, F. 2008. A game-theoretic measure of argument strength for abstract argumentation. In *Logics in A.I.*, volume 5293 of *LNCS*, 285–297.

Maudet, N., and Evrard, F. 1998. A generic framework for dialogue game implementation. In *Proc. 2nd Workshop on Formal Semantics & Pragmatics of Dialogue*, 185198. University of Twente.

McBurney, P., and Parsons, S. 2002. Games that agents play:

A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information* 11:315–334.

McBurney, P.; van Eijk, R.; Parsons, S.; and Amgoud, L. 2003. A dialogue-game protocol for agent purchase negotiations. *Journal of Autonomous Agents and Multi-Agent Systems* 7:235–273.

Oren, N., and Norman, T. 2009. Arguing using opponent models. In *Argumentation in Multi-agent Systems*, volume 6057 of *LNCS*, 160–174.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2003. Properties and complexity of some formal inter-agent dialogues. *J. of Logic and Comp.* 13(3):347–376.

Prakken, H. 2005. Coherence and flexibility in dialogue games for argumentation. *J. of Logic and Comp.* 15(6):1009–1040.

Rotstein, N.; Moguillansky, M.; and Simari, G. 2009. Dialectical abstract argumentation. In *Proceedings of IJCAI'09*, 898–903.

Walton, D., and Krabbe, E. 1995. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY Press.

Wooldridge, M.; McBurney, P.; and Parsons, S. 2005. On the meta-logic of arguments. In *Argumentatoin in Multi-agent Systems*, volume 4049 of *LNCS*, 42–56. Springer.

Abduction in Argumentation: Dialogical Proof Procedures and Instantiation

Richard Booth[1] and **Dov Gabbay**[2] and **Souhila Kaci**[3]

Tjitze Rienstra[1,3] and **Leendert van der Torre**[1]

[1]**University of Luxembourg**
Computer Science and Communication
6 rue Richard Coudenhove-Kalergi, Luxembourg
richard.booth/tjitze.rienstra/leon.vandertorre@uni.lu

[2]**King's College London**
Deptartment of Computer Science
Strand, London WC2R 2LS, UK
dov.gabbay@kcl.ac.uk

[3]**University of Montpellier 2**
LIRMM
161 rue Ada, Montpellier, France
souhila.kaci@lirmm.fr

Abstract

We develop a model of abduction in abstract argumentation, where changes to an argumentation framework act as hypotheses to explain the support of an observation. We present dialogical proof theories for the main decision problems (i.e., finding hypotheses that explain skeptical/credulous support) and we show that our model can be instantiated on the basis of abductive logic programs.

Introduction

In the context of abstract argumentation (Dung 1995), abduction can be seen as the problem of finding *changes* to an argumentation framework (or *AF* for short) with the goal of explaining observations that can be justified by making arguments accepted. The general problem of whether and how an AF can be changed with the goal of changing the status of arguments has been studied by Baumann and Brewka (2010), who called it the *enforcing* problem, as well as Bisquert et al. (2013), Perotti et al. (2011) and Kontarinis et al. (2013). None of these works, however, made any explicit link with abduction. Sakama (2013), on the other hand, explicitly focused on abduction, and presented a model in which additions as well as removals of arguments from an abstract AF act as explanations for the observation that an argument is accepted or rejected.

While Sakama did address computation in his framework, his method was based on translating abstract AFs into logic programs. Proof theories in argumentation are, however, often formulated as *dialogical* proof theories, which aim at relating the problem they address with stereotypical patterns found in real world dialogue. For example, proof theories for skeptical/credulous acceptance have been modelled as dialogues in which a proponent persuades an opponent to accept the necessity/possibility of an argument (Modgil and Caminada 2009), while credulous acceptance has also been related to Socratic style dialogue (Caminada 2010). Thus, the question of how decision problems in abduction in argumentation can similarly be modelled as dialogues remains open.

Furthermore, argumentation is often used as an abstract model for non-monotonic reasoning formalisms. For example, an *instantiated* AF can be generated on the basis of a logic program. Consequences can then be computed by looking at the extensions of the instantiated AF (Dung 1995). In the context of abduction, one may ask whether a model of abduction in argumentation can similarly be seen as an abstraction of *abductive* logic programming. Sakama, however, did not explore the instantiation of his model, meaning that this question too remains open.

This brings us to the contribution of this paper. We first present a model of abduction in abstract argumentation, based on the notion of an AAF (abductive argumentation framework) that encodes different possible changes to an AF, each of which may act as a hypothesis to explain an observation that can be justified by making an argument accepted. We then do two things:

1. We present sound and complete dialogical proof procedures for the main decision problems, i.e., finding hypotheses that explain skeptical/credulous acceptance of arguments in support of an observation. These proof procedures show that the problem of abduction is related to an extended form of persuasion, where the proponent uses *hypothetical* moves to persuade the opponent.
2. We show that AAFs can be instantiated by ALPs (abductive logic programs) in such a way that the hypotheses generated for an observation by the ALP can be computed by translating the ALP into an AAF. The type of ALPs we focus on are based on Sakama and Inoue's model of *extended* abduction (1995; 1999), in which hypotheses have a positive as well as a negative element (i.e., facts added to the logic program as well as facts removed from it).

In sum, our contribution is a model of abduction in argumentation with dialogical proof theories for the main decision problems, which can be seen as an abstraction of abduction in logic programming.

The overview of this paper is as follows. After introducing the necessary preliminaries we present in section *Abductive AFs* our model of abduction in argumentation. In section *Explanation dialogues* we present dialogical proof procedures for the main decision problems (explaining skeptical/credulous acceptance). In section *Abduction in logic programming* we show that our model of abduction can be used to instantiate abduction in logic programming. We conclude with the two sections *Related work* and *Conclusions and future work*.

Preliminaries

An argumentation framework consists of a set A of arguments and a binary *attack* relation $\rightsquigarrow$ over A (Dung 1995). We assume in this paper that A is a finite subset of a fixed set $\mathcal{U}$ called the *universe of arguments*.

Definition 1. *Given a countably infinite set $\mathcal{U}$ called the* universe of arguments*, an* argumentation framework *(*AF*, for short) is a pair $F = (A, \rightsquigarrow)$ where A is a finite subset of $\mathcal{U}$ and $\rightsquigarrow$ a binary relation over A. If $a \rightsquigarrow b$ we say that a* attacks *b. $\mathcal{F}$ denotes the set of all AFs.*

Extensions are sets of arguments that represent different viewpoints on the acceptance of the arguments of an AF. A *semantics* is a method to select extensions that qualify as somehow justifiable. We focus on one of the most basic ones, namely the *complete* semantics (Dung 1995).

Definition 2. *Let $F = (A, \rightsquigarrow)$. An* extension *of F is a set $E \subseteq A$. An extension E is* conflict-free *iff for no $a, b \in E$ it holds that $a \rightsquigarrow b$. An argument $a \in A$ is* defended in F *by E iff for all b such that $b \rightsquigarrow a$ there is a $c \in E$ such that $c \rightsquigarrow b$. Given an extension E, we define $Def_F(E)$ by $Def_F(E) = \{a \in A \mid E \text{ defends } a \text{ in } F\}$. An extension E is* admissible *iff E is conflict-free and $E \subseteq Def_F(E)$, and* complete *iff E is conflict-free and $E = Def_F(E)$. The set of complete extension of F will be denoted by $Co(F)$. Furthermore, the grounded extension (denoted by $Gr(F)$) is the unique minimal (w.r.t. $\subseteq$) complete extension of F.*

An argument is said to be *skeptically* (resp. *credulously*) accepted w.r.t. the complete semantics iff it is a member of all (resp. some) complete extensions. Note that the set of skeptically accepted arguments coincides with the grounded extension. Furthermore, an argument is a member of a complete extension iff it is a member of a *preferred* extension, which is a maximal (w.r.t. $\subseteq$) complete extension. Consequently, credulous acceptance under the preferred semantics (as studied e.g. in (Modgil and Caminada 2009)) coincides with credulous acceptance under the complete semantics.

Abductive AFs

Abduction is a form of reasoning that goes from an observation to a hypothesis. We assume that an observation translates into a set $X \subseteq A$. Intuitively, X is a set of arguments that each individually support the observation. If at least one argument $x \in X$ is skeptically (resp. credulously) accepted w.r.t. the complete semantics, we say that the observation X is skeptically (resp. credulously) *supported*.

Definition 3. *Given an AF $F = (A, \rightsquigarrow)$, an observation $X \subseteq A$ is skeptically (resp. credulously) supported iff for all (resp. some) $E \in Co(F)$ it holds that $x \in E$ for some $x \in X$.*

The following proposition implies that checking whether an observation X is skeptically supported can be done by checking whether an individual argument $x \in X$ is in the grounded extension.

Proposition 1. *Let $F = (A, \rightsquigarrow)$ and $X \subseteq A$. It holds that F skeptically supports X iff $x \in Gr(F)$ for some $x \in X$.*

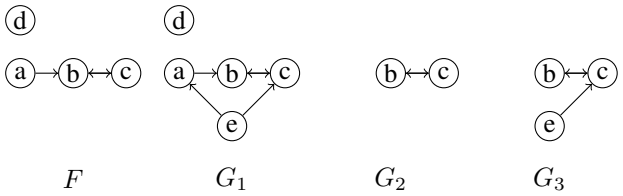


Figure 1: The AFs of the AAF $(F, \{F, G_1, G_2, G_3\})$.

Proof of proposition 1. The if direction is immediate. For the only if direction, assume $F = (A, \rightsquigarrow)$ explains skeptical support for X. Then for every complete extension E of F, there is an $x \in X$ s.t. $x \in E$. Define G by $G = (A \cup \{a, b\}, \rightsquigarrow \cup \{(x, a) \mid x \in X\} \cup \{(a, b)\})$, where $a, b \notin A$. Then for every complete extension E of G it holds that $b \in E$, hence $b \in Gr(G)$. Thus $x \in Gr(G)$ for some $x \in X$. But $Gr(F) = Gr(G) \cap A$, hence $x \in Gr(F)$ for some $x \in X$. □

It may be that an AF F does not skeptically or credulously support an observation X. Abduction then amounts to finding a change to F so that X is supported. We use the following definition of an *AAF* (*Abductive AF*) to capture the changes w.r.t. F (each change represented by an AF G called an *abducible* AF) that an agent considers. We assume that F itself is also an abducible AF, namely one that captures the case where no change is necessary. Other abducible AFs may be formed by addition of arguments and attacks to F, removal of arguments and attacks from F, or a combination of both.

Definition 4. *An* abductive AF *is a pair $M = (F, I)$ where F is an AF and $I \subseteq \mathcal{F}$ a set of AFs called* abducible *such that $F \in I$.*

Given an AAF (F, I) and observation X, skeptical/credulous support for X can be explained by the change from F to some $G \in I$ that skeptically/credulously supports X. In this case we say that G *explains* skeptical/credulous support for X. The arguments/attacks added to and absent from G can be seen as the actual explanation.

Definition 5. *Let $M = (F, I)$ be an AAF. An abducible AF $G \in I$* explains *skeptical (resp. credulous) support for an observation X iff G skeptically (resp. credulously) supports X.*

One can focus on explanations satisfying additional criteria, such as minimality w.r.t. the added or removed arguments/attacks. We leave the formal treatment of such criteria for future work.

Example 1. *Let $M = (F, \{F, G_1, G_2, G_3\})$, where F, G_1, G_2 and G_3 are as defined in figure 1. Let $X = \{b\}$ be an observation. It holds that G_1 and G_3 both explain skeptical support for X, while G_2 only explains credulous support for X.*

Remark 1. *The main difference between Sakama's (2013) model of abduction in abstract argumentation and the one presented here, is that he takes an explanation to be a set of independently selectable abducible arguments, while we*

take it to be a change to the AF that is applied as a whole. In section we show that this is necessary when applying the abstract model in an instantiated setting.

Explanation dialogues

In this section we present methods to determine, *given an AAF $M = (F, I)$ (for $F = (A, \rightsquigarrow)$) whether an abducible AF $G \in I$ explains credulous or skeptical support for an observation $X \subseteq A$*. We build on ideas behind the *grounded* and *preferred games*, which are dialogical procedures that determine skeptical or credulous acceptance of an argument (Modgil and Caminada 2009). To sketch the idea behind these games (for a detailed discussion cf. (Modgil and Caminada 2009)): two imaginary players (PRO and OPP) take alternating turns in putting forward arguments according to a set of rules, PRO either as an initial claim or in defence against OPP's attacks, while OPP initiates different disputes by attacking the arguments put forward by PRO. Skeptical or credulous acceptance is proven if PRO can win the game by ending every dispute in its favour according to a "last-word" principle.

Our method adapts this idea so that the moves made by PRO are essentially *hypothetical* moves. That is, to defend the initial claim (i.e., to explain an observation) PRO can put forward, by way of hypothesis, any attack $x \rightsquigarrow y$ present in some $G \in I$. This marks a choice of PRO to focus only on those abducible AFs in which the attack $x \rightsquigarrow y$ is present. Similarly, PRO can reply to an attack $x \rightsquigarrow y$, put forward by OPP, with the claim that this attack is invalid, marking the choice of PRO to focus only on the abducible AFs in which the attack $x \rightsquigarrow y$ is *not* present. Thus, each move by PRO narrows down the set of abducible AFs in which all of PRO's moves are valid. The objective is to end the dialogue with a non-empty set of abducible AFs. Such a dialogue represents a proof that these abducible AFs explain skeptical or credulous support for the observation.

Alternatively, such dialogues can be seen as games that determine skeptical/credulous support of an observation by an AF that are played simultaneously over all abducible AFs in the AAF. In this view, the objective is to end the dialogue in such a way that it represents a proof for at least one abducible AF. Indeed, in the case where $M = (F, \{F\})$, our method reduces simply to a proof theory for skeptical or credulous support of an observation by F.

Before we move on we need to introduce some notation.

Definition 6. *Given a set I of AFs we define:*

- $A_I = \cup\{A \mid (A, \rightsquigarrow) \in I\}$,
- $\rightsquigarrow_I = \cup\{\rightsquigarrow \mid (A, \rightsquigarrow) \in I\}$,
- $I_{x \rightsquigarrow y} = \{(A, \rightsquigarrow) \in I \mid x, y \in A, x \rightsquigarrow y\}$,
- $I_X = \{(A, \rightsquigarrow) \in I \mid X \subseteq A\}$.

We model dialogues as sequences of *moves*, each move being of a certain type, and made either by PRO or OPP.

Definition 7. *Let $M = (F, I)$ be an AAF. A* dialogue based on *M is a sequence $S = (m_1, \ldots, m_n)$, where each m_i is either:*

- *an* OPP attack *"**OPP:** $x \rightsquigarrow y$", where $x \rightsquigarrow_I y$,*
- *a* hypothetical PRO defence *"**PRO:** $y \rightsquigarrow^+ x$", where $y \rightsquigarrow_I x$,*
- *a* hypothetical PRO negation *"**PRO:** $y \rightsquigarrow^- x$", where $y \rightsquigarrow_I x$,*
- *a* conceding move *"**OPP: ok**",*
- *a* success claim move *"**PRO: win**".*

We denote by $S \cdot S'$ the concatenation of S and S'.

Intuitively, a move **OPP:** $y \rightsquigarrow x$ represents an attack by OPP on the argument x by putting forward the attacker y. A hypothetical PRO defence **PRO:** $y \rightsquigarrow^+ x$ represents a defence by PRO who puts forward y to attack the argument x put forward by OPP. A hypothetical PRO negation **PRO:** $y \rightsquigarrow^- x$, on the other hand, represents a claim by PRO that the attack $y \rightsquigarrow x$ is *not* a valid attack. The conceding move **OPP: ok** is made whenever OPP runs out of possibilities to attack a given argument, while the move **PRO: win** is made when PRO is able to claim success.

In the following sections we specify how dialogues are structured. Before doing so, we introduce some notation that we use to keep track of the abducible AFs on which PRO chooses to focus in a dialogue D. We call this set the *information state* of D after a given move. While it initially contains all abducible AFs in *M*, it is restricted when PRO makes a move **PRO:** $x \rightsquigarrow^+ y$ or **PRO:** $x \rightsquigarrow^- y$.

Definition 8. *Let $M = (F, I)$ be an AAF. Let $D = (m_1, \ldots, m_n)$ be a dialogue based on M. We denote the* information state in D after move i by $J(D, i)$, *which is defined recursively by:*

$$J(D,i) = \begin{cases} I & \text{if } i = 0, \\ J(D, i-1) \cap I_{x \rightsquigarrow y} & \text{if } m_i = \textbf{PRO:}\ x \rightsquigarrow^+ y, \\ J(D, i-1) \setminus I_{x \rightsquigarrow y} & \text{if } m_i = \textbf{PRO:}\ x \rightsquigarrow^- y, \\ J(D, i-1) & \text{otherwise.} \end{cases}$$

We denote by $J(D)$ the information state $J(D, n)$.

Skeptical explanation dialogues

We define the rules of a dialogue using a set of production rules that recursively define the set of sequences constituting dialogues. (The same methodology was used by Booth et al. (2013) in defining a dialogical proof theory related to preference-based argumentation.) In a skeptical explanation dialogue for an observation X, an initial argument $x \in X$ is challenged by the opponent, who puts forward all possible attacks **OPP:** $y \rightsquigarrow x$ present in any of the abducible AFs present in the AAF, followed by **OPP: ok**. We call this a *skeptical OPP reply* to x. For each move **OPP:** $y \rightsquigarrow x$, PRO responds with a *skeptical PRO reply to* $y \rightsquigarrow x$, which is either a hypothetical defence **PRO:** $z \rightsquigarrow^+ y$ (in turn followed by a skeptical OPP reply to z) or a hypothetical negation **PRO:** $y \rightsquigarrow^- x$. Formally:

Definition 9 (Skeptical explanation dialogue)**.** *Let $F = (A, \rightsquigarrow)$, $M = (F, I)$ and $x \in A$.*

- *A* skeptical OPP reply to x *is a finite sequence* $(\textbf{OPP:}\ y_1 \rightsquigarrow x) \cdot S_1 \cdot \ldots \cdot (\textbf{OPP:}\ y_n \rightsquigarrow x) \cdot S_n \cdot (\textbf{OPP: ok})$ *where $\{y_1, \ldots, y_n\} = \{y \mid y \rightsquigarrow_I x\}$ and each S_i is a skeptical PRO reply to $y_i \rightsquigarrow x$.*

- *A* skeptical PRO reply to $y \rightsquigarrow x$ *is either: (1) A sequence* (***PRO:*** $z \rightsquigarrow^+ y$)$\cdot S$ *where* $z \rightsquigarrow_I y$ *and where* S *is a skeptical OPP reply to* z*, or (2) The sequence* (***PRO:*** $y \rightsquigarrow^- x$).

Given an observation $X \subseteq A$ *we say that M generates the* skeptical explanation dialogue D *for* X *iff* $D = S \cdot$ (***PRO: win***)*, where* S *is a skeptical OPP reply to some* $x \in X$.

The following theorem establishes soundness and completeness.

Theorem 1. *Let* $M = (F, I)$ *be an AAF where* $F = (A, \rightsquigarrow)$. *Let* $X \subseteq A$ *and* $G \in I$. *It holds that* G *explains skeptical support for* X *iff M generates a skeptical explanation dialogue* D *for* X *such that* $G \in J(D)$.

The proof requires the following definitions and results.

Definition 10. *(Dung 1995) Given an AF* $F = (A, \rightsquigarrow)$ *the* characteristic function $C_F : 2^A \rightarrow 2^A$ *is defined by* $C_F(S) = \{x \in A \mid S \text{ defends } x\}$.

Lemma 1. *(Dung 1995) Given an AF* F, $Gr(F)$ *coincides with the least fixed point of* C_F.

Definition 11. *Given an AF* $F = (A, \rightsquigarrow)$ *we define the* degree $Deg_F(x)$ *of an argument* $x \in Gr(F)$ *to be the smallest positive integer* n *s.t.* $x \in C_F^n(\emptyset)$.

Lemma 2. *Given an AF* $F = (A, \rightsquigarrow)$ *and* $x \in Gr(F)$. *For every* $y \in A$ *s.t.* $y \rightsquigarrow x$ *there is a* $z \in Gr(F)$ *such that* $z \rightsquigarrow y$ *and* $Deg_F(z) < Deg_F(x)$.

Proof of lemma 2. Let $F = (A, \rightsquigarrow)$, $x \in Gr(F)$ and $y \in A$ an argument s.t. $y \rightsquigarrow x$. Definition 2 implies that there is a $z \in Gr(F)$ s.t. $z \rightsquigarrow y$. Definition 10 furthermore implies that for every $X \subseteq A$, if $x \in C_F(X)$ then $z \in X$. Definition 11 now implies that $Deg_F(x) > Deg_F(z)$. □

Proof of theorem 1. Let $M = (F, I)$ be an AAF where $F = (A, \rightsquigarrow)$. Let $X \subseteq A$ and $G \in I$.

Only if: Assume that G explains skeptical support for X. Proposition 1 implies that there is an $x \in X$ such that $x \in Gr(F)$. We prove that *M* generates a skeptical OPP reply D to x such that $G \in J(D)$. We prove this by strong induction on $Deg_G(x)$.

Let the induction hypothesis $H(i)$ stand for: *If* $x \in Gr(G)$ *and* $Deg_G(x) = i$ *then there is a skeptical OPP reply* D *to* x *s.t.* $G \in J(D)$.

Assume $H(i)$ holds for all $0 < i < k$. We prove $H(k)$. Assume $x \in Gr(G)$ and $Deg_G(x) = k$. We construct an OPP reply D to x such that $G \in J(D)$. Given an argument $y \in A_G$ s.t. $y \rightsquigarrow_G x$ we denote by $Z(y)$ the set $\{z \mid z \rightsquigarrow_G y, z \in Gr(G)\}$. Definition 2 implies that for every $y \in A_G$ s.t. $y \rightsquigarrow x$, $Z(y) \neq \emptyset$. Furthermore lemma 2 implies that for every $y \in A_G$ s.t. $y \rightsquigarrow x$ and for every $z \in Z(y)$ it holds that $Deg_G(z) < k$. We can now define D by $D = D_1 \cdot D_2 \cdot$ (**OPP: ok**) where: $D_1 =$ (**OPP:** $y_1 \rightsquigarrow x$)$\cdot$ (**PRO:** $y_1 \rightsquigarrow^- x$)$\cdot \ldots \cdot$ (**OPP:** $y_n \rightsquigarrow x$)$\cdot$ (**PRO:** $y_n \rightsquigarrow^- x$) where $\{y_1, \ldots, y_n\} = \{y \in A_I \mid y \rightsquigarrow_I x, y \not\rightsquigarrow_G x\}$, and $D_2 =$ (**OPP:** $y'_1 \rightsquigarrow x$)$\cdot$ (**PRO:** $z_1 \rightsquigarrow^+ y'_1$)$\cdot D_{z_1} \cdot \ldots \cdot$ (**OPP:** $y'_m \rightsquigarrow x$)$\cdot$ (**PRO:** $z_m \rightsquigarrow^+ y'_m$)$\cdot D_{z_m}$ where $\{y'_1, \ldots, y'_m\} = \{y \in A_I \mid y \rightsquigarrow_G x\}$, for each $j \in \{1, \ldots, m\}$, $z_j \in Z(y_j)$ and D_{z_j} is a skeptical OPP reply to z_j (because $Deg_G(z_j) < k$ and $H(i)$ holds for all $0 < i < k$, this skeptical OPP reply exists). It holds that D is a skeptical OPP reply to x. Furthermore it holds that $G \in J(D_1)$ and $G \in J(D_2)$ and hence $G \in J(D)$.

By the principle of strong induction it follows that there exists a skeptical OPP reply D to x such that $G \in J(D)$. Hence *M* generates a skeptical explanation dialogue $D \cdot$ (**PRO: win**) for X such that $G \in J(D \cdot$ (**PRO: win**)).

If: We prove that if D is a skeptical OPP reply to some $x \in X$ such that $G \in J(D)$ then $x \in Gr(G)$. We prove this by induction on the structure of D.

Assume that for every proper subsequence D' of D that is a skeptical OPP reply to an argument z it holds that $z \in Gr(G)$ and $G \in J(D)$. (The base case is the special case where no proper subsequence of D is a skeptical OPP reply.) We prove that $x \in Gr(G)$. We write D as (**OPP:** $y_1 \rightsquigarrow x$)$\cdot D_1 \cdot \ldots \cdot$ (**OPP:** $y_n \rightsquigarrow x$)$\cdot D_n \cdot$ (**OPP: ok**). Then every D_i (for $1 \leq i \leq n$) is either of the form **PRO:** $y_i \rightsquigarrow^- x$ or of the form **PRO:** $z \rightsquigarrow^+ y_i \cdot D'$, where D' is a proper subsequence of D that is a skeptical OPP reply to some argument z and $G \in J(D')$. Thus, for every $y \in A_I$ s.t. $y \rightsquigarrow_I x$ it holds that either $y \not\rightsquigarrow_G x$, or y is attacked by some z s.t. $z \in Gr(G)$. It follows that $x \in Gr(G)$.

By the principle of induction it follows that if D is a skeptical OPP reply to some $x \in X$ such that $G \in J(D)$ then $x \in Gr(G)$. Thus, if *M* generates a skeptical explanation dialogue $D \cdot$ (**PRO: win**) for X such that $G \in J(D \cdot$ (**PRO: win**)) then D is a skeptical OPP reply to some $x \in X$ and therefore it holds that $x \in Gr(G)$ and finally that G explains skeptical support for X. □

Example 2. *The listing below shows a skeptical explanation dialogue* $D = \{m_1, \ldots, m_8\}$ *for the observation* $\{b\}$ *that is generated by the AAF defined in example 1.*

i	m_i	$J(D, i)$
1	***OPP:*** $c \rightsquigarrow b$	$\{F, G_1, G_2, G_3\}$
2	***PRO:*** $e \rightsquigarrow^+ c$	$\{G_1, G_3\}$
3	***OPP: ok***	$\{G_1, G_3\}$
4	***OPP:*** $a \rightsquigarrow b$	$\{G_1, G_3\}$
5	***PRO:*** $e \rightsquigarrow^+ a$	$\{G_1\}$
6	***OPP: ok***	$\{G_1\}$
7	***OPP: ok***	$\{G_1\}$
8	***PRO: win***	$\{G_1\}$

The sequence $(m_1, \ldots, m_7)$ *is a skeptical OPP reply to* b*, in which OPP puts forward the two attacks* $c \rightsquigarrow b$ *and* $a \rightsquigarrow b$. *PRO defends* b *from both* c *and* a *by putting forward the attacker* e *(move 2 and 5). This leads to the focus first on the abducible AFs* G_1, G_3 *(in which the attack* $e \rightsquigarrow c$ *exists) and then on* G_1 *(in which the attack* $e \rightsquigarrow a$ *exists). This proves that* G_1 *explains skeptical support for the observation* $\{b\}$. *Another dialogue is shown below.*

i	m_i	$J(D,i)$
1	***OPP:*** $c \rightsquigarrow b$	$\{F, G_1, G_2, G_3\}$
2	***PRO:*** $e \rightsquigarrow^+ c$	$\{G_1, G_3\}$
3	***OPP: ok***	$\{G_1, G_3\}$
4	***OPP:*** $a \rightsquigarrow b$	$\{G_1, G_3\}$
5	***PRO:*** $a \rightsquigarrow^- b$	$\{G_3\}$
6	***OPP: ok***	$\{G_3\}$
7	***PRO: win***	$\{G_3\}$

Here, PRO defends b from c by using the argument e, but defends b from a by claiming that the attack $a \rightsquigarrow b$ is invalid. This leads to the focus first on the abducible AFs G_1, G_3 (in which the attack $e \rightsquigarrow c$ exists) and then on G_3 (in which the attack $a \rightsquigarrow b$ does not exist). This dialogue proves that G_3 explains skeptical support for $\{b\}$.

Credulous explanation dialogues

The definition of a credulous explanation dialogue is similar to that of a skeptical one. The difference lies in what constitutes an acceptable defence. To show that an argument x is skeptically accepted, x must be defended from its attackers by arguments other than x itself. For credulous acceptance, however, it suffices to show that x is a member of an admissible set, and hence x may be defended from its attackers by any argument, including x itself. To achieve this we need to keep track of the arguments that are, according to the moves made by PRO, accepted. Once an argument x is accepted, PRO does not need to defend x again, if this argument is put forward a second time.

Formally a *credulous OPP reply* to (x, Z) (for some $x \in A_I$ and set $Z \subseteq A_I$ used to keep track of accepted arguments) consists of all possible attacks **OPP:** $y \rightsquigarrow x$ on x, followed by **OPP: ok** when all attacks have been put forward. For each move **OPP:** $y \rightsquigarrow x$, PRO responds either by putting forward a hypothetical defence **PRO:** $z \rightsquigarrow^+ y$ which (this time *only if* $z \notin Z$) is followed by a credulous OPP reply to $(z, Z \cup \{z\})$, or by putting forward a hypothetical negation **PRO:** $y \rightsquigarrow^- x$. We call this response a *credulous PRO reply to* $(y \rightsquigarrow x, Z)$. A credulous explanation dialogue for a set X consists of a credulous OPP reply to $(x, \{x\})$ for some $x \in X$, followed by a success claim **PRO: win**.

In addition, arguments put forward by PRO in defence of the observation may not conflict. Such a conflict occurs when OPP puts forward **OPP:** $x \rightsquigarrow y$ and **OPP:** $y \rightsquigarrow z$ (indicating that both y and z are accepted) while PRO does not put forward **PRO:** $y \rightsquigarrow^- z$. If this situation does not occur we say that the dialogue is *conflict-free*.

Definition 12 (Credulous explanation dialogue)**.** *Let $F = (A, \rightsquigarrow)$, $M = (F, I)$, $x \in A$ and $Z \subseteq A$.*

- *A* credulous OPP reply to (x, Z) *is a finite sequence* $(\textbf{\textit{OPP:}}\ y_1 \rightsquigarrow x) \cdot S_1 \cdot \ldots \cdot (\textbf{\textit{OPP:}}\ y_n \rightsquigarrow x) \cdot S_n \cdot (\textbf{\textit{OPP: ok}})$ *where $\{y_1, \ldots, y_n\} = \{y \mid y \rightsquigarrow_I x\}$ and each S_i is a credulous PRO reply to $(y_i \rightsquigarrow x, Z)$.*
- *A* credulous PRO reply to $(y \rightsquigarrow x, Z)$ *is either: (1) a sequence $(\textbf{\textit{PRO:}}\ z \rightsquigarrow^+ y) \cdot S$ such that $z \rightsquigarrow_I y$, $z \notin Z$ and S is a credulous OPP reply to $(z, Z \cup \{z\})$, (2) a sequence $(\textbf{\textit{PRO:}}\ z \rightsquigarrow^+ y)$ such that $z \rightsquigarrow_I y$ and $z \in Z$, or (3) the sequence $(\textbf{\textit{PRO:}}\ y \rightsquigarrow^- x)$.*

Given a set $X \subseteq A$ we say that M generates the credulous explanation dialogue *D for X iff $D = S \cdot (\textbf{\textit{PRO: win}})$, where S is a credulous OPP reply to $(x, \{x\})$ for some $x \in X$. We say that D is* conflict-free *iff for all $x, y, z \in A_I$ it holds that if D contains the moves $\textbf{\textit{OPP:}}\ x \rightsquigarrow y$ and $\textbf{\textit{OPP:}}\ y \rightsquigarrow z$ then it contains the move $\textbf{\textit{PRO:}}\ y \rightsquigarrow^- z$.*

The following theorem establishes soundness and completeness.

Theorem 2. *Let $M = (F, I)$ be an AAF where $F = (A, \rightsquigarrow)$. Let $X \subseteq A$ and $G \in I$. It holds that G explains credulous support for X iff M generates a conflict-free credulous explanation dialogue D for X such that $G \in J(D)$.*

Proof of theorem 2. Let $M = (F, I)$ be an AAF where $F = (A, \rightsquigarrow)$. Let $X \subseteq A$ and $G \in I$.

Only if: Assume that G explains credulous support for X. Then there is an admissible set E of G such that $a \in E$ for some $a \in X$. Based on E and a we construct a conflict-free credulous explanation dialogue D for X such that $G \in J(D)$. Given an argument $x \in E$ we define the credulous OPP reply $D(x, Z)$ recursively by $D(x, Z) = (\textbf{OPP:}\ y_1 \rightsquigarrow x) \cdot S_1 \cdot \ldots \cdot (\textbf{OPP:}\ y_n \rightsquigarrow x) \cdot S_n \cdot (\textbf{OPP: ok})$

where $\{y_1, \ldots, y_n\} = \{y \mid y \rightsquigarrow_I x\}$ and each S_i is a credulous PRO reply defined by the following cases:

- Case 1: $y_i \rightsquigarrow_G x$. Let z be an argument such that $z \in E$ and $z \rightsquigarrow_G y_i$. (Admissibility of E guarantees the existence of z.)
  - Case 1.1: $z \notin Z$: Then $S_i = \textbf{PRO:}\ z \rightsquigarrow^+ y_i \cdot D(z, Z \cup \{z\})$.
  - Case 1.2: $z \in Z$: Then $S_i = \textbf{PRO:}\ z \rightsquigarrow^+ y_i$.
- Case 2: $y_i \not\rightsquigarrow_G x$: Then $S_i = \textbf{PRO:}\ y_i \rightsquigarrow^- x$.

Let $D = (m_1, \ldots, m_n) = D(a, \{a\}) \cdot (\textbf{PRO: win})$. It can be checked that D is a credulous explanation dialogue for $\{a\}$. We need to prove that:

- $G \in J(D)$. This follows from the fact that for all $i \in \{1, \ldots, n\}$, $m_i = \textbf{PRO:}\ x \rightsquigarrow^- y$ only if $x \not\rightsquigarrow_G y$ and $m_i = \textbf{PRO:}\ x \rightsquigarrow^+ y$ only if $x \rightsquigarrow_G y$.
- D is finite. This follows from the fact that for every credulous OPP reply $D(x, Z)$ that is a subsequence of a credulous OPP reply $D(y, Z')$ it holds that Z is a strict superset of Z', together with the fact that $Z \subseteq A_I$ and A_I is finite.
- D is conflict-free. We prove this by contradiction. Thus we assume that for some x, y, z there are moves **OPP:** $x \rightsquigarrow y$ and **OPP:** $y \rightsquigarrow z$ and no move **PRO:** $y \rightsquigarrow^- z$. By the construction of D it follows that $y, z \in E$. Furthermore if $y \not\rightsquigarrow_G z$ then by the construction of D, the move **OPP:** $y \rightsquigarrow z$ is followed by **PRO:** $y \rightsquigarrow^- z$, which is a contradiction. Hence $y \rightsquigarrow_G z$. Thus E is not a conflict-free set of G, contradicting our assumption that E is an admissible set of G. Hence D is conflict-free.

Hence there is a conflict-free credulous explanation dialogue D for X such that $G \in J(D)$.

If: Let D be a conflict-free credulous explanation dialogue for an observation X such that $G \in J(D)$. We prove that there is an admissible set E of G s.t. $a \in E$ for some $a \in X$.

We define E by $E = \{a\} \cup \{x \mid \textbf{PRO:}\ x \rightsquigarrow^+ z \in D\}$. To prove that E is an admissible set of G we show that (1) for every $x \in E$ and every $y \in A$ such that $y \rightsquigarrow_G x$, there is a $z \in E$ such that $y \rightsquigarrow_G z$ and (2) that E is a conflict-free set of G.

1. Let $x \in E$. Then either $x = a$ or there is a move $m_i = \textbf{PRO:}\ x \rightsquigarrow^+ y$ in D. It follows either that m_{i+1} is a credulous OPP reply to (x, Z) or not, in which case there is a move m_j (for $j < i$) that is a credulous OPP reply to Hence for some $Z \subseteq A_I$ there is an OPP reply to (x, Z) in D. For m_{i+1} there are two cases:
   - $m_{i+1} = \textbf{PRO:}\ z \rightsquigarrow^+ y$. Then $z \in E$ and, because $G \in J(D)$, $z \rightsquigarrow_G y$.
   - $m_{i+1} = \textbf{PRO:}\ y \rightsquigarrow^- x$. But $y \rightsquigarrow_G x$, hence $G \notin J(D)$, which is a contradiction. Thus, this case is not possible.

   Thus for every $x \in E$ and every $y \in A$ s.t. $y \rightsquigarrow_G x$, there is a $z \in E$ such that $z \rightsquigarrow_G y$.
2. Assume the contrary, i.e., E is not conflict-free. Then for some $y, z \in E$ it holds that $y \rightsquigarrow_G z$. From (1) it follows that there is also an $x \in E$ such that $x \rightsquigarrow_G y$. By the construction of E it follows that either $y = a$ or for some x' there is a move $\textbf{PRO:}\ y \rightsquigarrow^+ x'$ in D, and similarly either $z = a$ or for some x' there is a move $\textbf{PRO:}\ z \rightsquigarrow^+ x'$ in D. Hence there are moves $\textbf{OPP:}\ x \rightsquigarrow y$ and $\textbf{OPP:}\ y \rightsquigarrow z$ in D. From the fact that $G \in J(D)$ and $y \rightsquigarrow_G z$ it follows that there is no move $\textbf{PRO:}\ y \rightsquigarrow^- z$ in D. Hence D is not conflict-free, which is a contradiction. It follows that E is a conflict-free set of G.

It finally follows that E is an admissible set of G and $a \in E$ and hence G explains credulous support for X.

□

Example 3. *The listing below shows a conflict-free credulous explanation dialogue $D = (m_1, \ldots, m_6)$ for the observation $\{b\}$ generated by the AAF defined in example 1.*

i	m_i	$J(D,i)$
1	***OPP:*** $c \rightsquigarrow b$	$\{F, G_1, G_2, G_3\}$
2	***PRO:*** $b \rightsquigarrow^+ c$	$\{F, G_1, G_2, G_3\}$
3	***OPP:*** $a \rightsquigarrow b$	$\{F, G_1, G_2, G_3\}$
4	***PRO:*** $a \rightsquigarrow^- b$	$\{G_2, G_3\}$
5	***OPP: ok***	$\{G_2, G_3\}$
6	***PRO: win***	$\{G_2, G_3\}$

Here, the sequence $(m_1, \ldots, m_5)$ is a credulous OPP reply to $(b, \{b\})$. PRO defends b from OPP's attack $c \rightsquigarrow b$ by putting forward the attack $b \rightsquigarrow c$. Since b was already assumed to be accepted, this suffices. At move m_4, PRO defends itself from the attack $a \rightsquigarrow b$ by negating it. This restricts the focus on the abducible AFs G_2 and G_3. The dialogue proves that these two abducible AFs explain credulous support for the observation $\{b\}$. Finally, the skeptical explanation dialogues from example 2 are also credulous explanation dialogues.

Abduction in logic programming

In this section we show that AAFs can be instantiated with abductive logic programs, in the same way that regular AFs can be instantiated with regular logic programs. In sections and we recall the necessary basics of logic programming and the relevant results regarding logic programming as instantiated argumentation. In section we present a model of abductive logic programming based on Sakama and Inoue's model of extended abduction (1995; 1999), and in section we show how this model can be instantiated using AAFs.

Logic programs and partial stable semantics

A logic program P is a finite set of rules, each rule being of the form $C \leftarrow A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m$ where $C, A_1, \ldots, A_n, B_1, \ldots, B_m$ are *atoms*. If $m = 0$ then the rule is called *definite*. If both $n = 0$ and $m = 0$ then the rule is called a *fact* and we identify it with the atom C. We assume that logic programs are ground. Alternatively, P can be regarded as the set of ground instances of a set of non-ground rules. We denote by At_P the set of all (ground) atoms occurring in P. The logic programming semantics we focus on can be defined using *3-valued interpretations* (Przymusinski 1990):

Definition 13. *A 3-valued interpretation I of a logic program P is a pair $I = (T, F)$ where $T, F \subseteq At_P$ and $T \cap F = \emptyset$. An atom $A \in At(P)$ is* true *(resp.* false*,* undecided*) in I iff $A \in T$ (resp. $A \in F$, $A \in At_P \setminus (T \cup F)$).*

The following definition of a *partial stable model* is due to Przymusinski (1990). Given a logic program P and 3-valued interpretation I of P, the *GL-transformation* $\frac{P}{I}$ is a logic program obtained by replacing in every rule in P every premise $\sim B$ such that B is true (resp. undecided, false) in I by the atoms 0 (resp. $\frac{1}{2}$, 1), where 0 (resp. $\frac{1}{2}$, 1) are defined to be false (resp. undecided, true) in every interpretation. It holds that for all 3-valued interpretations I of P, $\frac{P}{I}$ is definite (i.e., consists only of definite rules). This means that $\frac{P}{I}$ has a unique *least* 3-valued interpretation (T, F) with minimal T and maximal F that satisfies all rules. That is, for all rules $C \leftarrow A_1, \ldots, A_n$, in $\frac{P}{I}$, C is true (resp. *not* false) in (T, F) if for all $i \in \{1, \ldots, n\}$, A_i is true (resp. *not* false) in (T, F). Given a 3-valued interpretation I, the least 3-valued interpretation of $\frac{P}{I}$ is denoted by $\Gamma(I)$. This leads to the following definition of a *partial stable model* of a logic program, along with the associated notions of consequence.

Definition 14. *(Przymusinski 1990) Let P be a logic program. A 3-valued interpretation I is a* partial stable model *of P iff $I = \Gamma(I)$. We say that an atom C is a skeptical (resp. credulous) consequence of P iff C is true in all (resp. some) partial stable models of P.*

It has been shown that the above defined notion of skeptical consequence coincides with the *well-founded* semantics (Przymusinski 1990).

Logic programming as argumentation

Wu et al. (2009) have shown that a logic program P can be transformed into an AF F in such a way that the conse-

quences of P under the partial stable semantics can be computed by looking at the complete extensions of F. The idea is that an argument consists of a conclusion $C \in At_P$, a set of rules $R \subseteq P$ used to derive C and a set $N \subseteq At_P$ of atoms that must be underivable in order for the argument to be acceptable. The argument is attacked by another argument with a conclusion C' iff $C' \in N$. The following definition, apart from notation, is due to Wu et al. (2009).

Definition 15. *Let P be a logic program. An instantiated argument is a triple (C, R, N), where $C \in At_P$, $R \subseteq P$ and $N \subseteq At_P$. We say that P generates (C, R, N) iff either:*

- *$r = C \leftarrow \sim B_1, \ldots, \sim B_m$ is a rule in P, $R = \{r\}$ and $N = \{B_1, \ldots, B_m\}$.*
- *(1) $r = C \leftarrow A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m$ is a rule in P, (2) P generates, for each $i \in \{1, \ldots, n\}$ an argument (A_i, R_i, N_i) such that $r \notin R_i$, and (3) $R = \{r\} \cup R_1 \cup \ldots \cup R_n$ and $N = \{B_1, \ldots, B_m\} \cup N_1 \cup \ldots \cup N_n$.*

We denote the set of arguments generated by P by A_P. Furthermore, the attack relation generated by P is denoted by $\rightsquigarrow_P$ and is defined by $(C, R, N) \rightsquigarrow_P (C', R', N')$ iff $C \in N'$.

The following theorem states that skeptical (resp. credulous) acceptance in $(A_P, \rightsquigarrow_P)$ corresponds with skeptical (resp. credulous) consequences in P as defined in definition 14. It follows from theorems 15 and 16 due to Wu et al. (2009).

Theorem 3. *Let P be a logic program. An atom $C \in At_P$ is a skeptical (resp. credulous) consequence of P iff some $(C, R, N) \in A_P$ is skeptically (resp. credulously) accepted in $(A_P, \rightsquigarrow_P)$.*

Abduction in logic programming

The model of abduction in logic programming that we use is based on the model of *extended* abduction studied by Inoue and Sakama (1995; 1999). They define an abductive logic program (ALP) to consist of a logic program and a set of atoms called *abducibles*.

Definition 16. *An abductive logic program is a pair (P, U) where P is a logic program and $U \subseteq At_P$ a set of facts called abducibles.*

Note that, as before, the set U consists of ground facts of the form $C \leftarrow$ (identified with the atom C) and can alternatively be regarded as the set of ground instances of a set of non-ground facts. A hypothesis, according to Inoue and Sakama's model, consists of both a positive element (i.e., abducibles added to P) and a negative element (i.e., abducibles removed from P).

Definition 17. *Let ALP $= (P, U)$ be an abductive logic program. A hypothesis is a pair (Δ^+, Δ^-) such that $\Delta^+, \Delta^- \subseteq U$ and $\Delta^+ \cap \Delta^- = \emptyset$. A hypothesis (Δ^+, Δ^-) skeptically (resp. credulously) explains a query $Q \in At_P$ if and only if Q is a skeptical (resp. credulous) consequence of $(P \cup \Delta^+) \setminus \Delta^-$.*

Note that Sakama and Inoue focus on computation of explanations under the stable model semantics of P, and require P to be acyclic to ensure that a stable model of P exists and is unique (1999). We, however, define explanation in terms of the consequences according to the partial stable models of P, which always exist even if P is not acyclic (Przymusinski 1990), so that we do not need this requirement.

The following example demonstrates the previous two definitions.

Example 4. *Let ALP $= (P, U)$ where $P = \{(p \leftarrow \sim s, r), (p \leftarrow \sim s, \sim q), (q \leftarrow \sim p), r\}$ and $U = \{r, s\}$. The hypothesis $(\{s\}, \emptyset)$ skeptically explains q, witnessed by the unique model $I = (\{r, s, q\}, \{p\})$ satisfying $I = \Gamma(I)$. Similarly, $(\{s\}, \{r\}))$ skeptically explains q and $(\emptyset, \{r\}))$ credulously explains q.*

Instantiated abduction in argumentation

In this section we show that an AAF (F, I) can be instantiated on the basis of an abductive logic program (P, U). The idea is that every possible hypothesis (Δ^+, Δ^-) maps to an abducible AF generated by the logic program $(P \cup \Delta^+) \setminus \Delta^-$. The hypotheses for a query Q then correspond to the abducible AFs that explain the observation X consisting of all arguments with conclusion Q. The construction of (F, I) on the basis of (P, U) is defined as follows.

Definition 18. *Let ALP $= (P, U)$ be an abductive logic program. Given a hypothesis (Δ^+, Δ^-), we denote by $F_{(\Delta^+, \Delta^-)}$ the AF $(A_{(P \cup \Delta^+) \setminus \Delta^-}, \rightsquigarrow_{(P \cup \Delta^+) \setminus \Delta^-})$. The AAF* generated by *ALP is denoted by M_{ALP} and defined by $M_{ALP} = (F_P, I_{ALP})$, where $I_{ALP} = \{F_{(\Delta^+, \Delta^-)} \mid \Delta^+, \Delta^- \subseteq U, \Delta^+ \cap \Delta^- = \emptyset\}$.*

The following theorem states the correspondence between the explanations of a query Q in an abductive logic program *ALP* and the explanations of an observation X in the AAF M_{ALP}.

Theorem 4. *Let ALP $= (P, U)$ be an abductive logic program, $Q \in At_P$ a query and (Δ^+, Δ^-) a hypothesis. Let $M_{ALP} = (F_{ALP}, I_{ALP})$. We denote by X_Q the set $\{(C, R, N) \in A_P \mid C = Q\}$. It holds that (Δ^+, Δ^-) skeptically (resp. credulously) explains Q iff $F_{(\Delta^+, \Delta^-)}$ skeptically (resp. credulously) explains X_Q.*

Proof of theorem 4. Follows directly from theorem 3 and definitions 17 and 18. □

This theorem shows that our model of abduction in argumentation can indeed be seen as an abstraction of abductive logic programming.

Example 5. *Let ALP $= (P, U)$ be the ALP as defined in example 4. All arguments generated by ALP are:*

$a = (p, \{(p \leftarrow \sim s, r), r\}, \{s\})$ $\quad$ $d = (r, \{r\}, \emptyset)$
$b = (q, \{(q \leftarrow \sim p)\}, \{p\})$ $\quad$ $e = (s, \{s\}, \emptyset)$
$c = (p, \{(p \leftarrow \sim s, \sim q)\}, \{s, q\})$

Given these definitions, the AAF in example 1 is equivalent to M_{ALP}. In example 4 we saw that the query q is skeptically explained by the hypotheses $(\{s\}, \emptyset)$ and $(\{s\}, \{r\})$, while $(\emptyset, \{r\})$ only credulously explains it. Indeed, looking again at example 1, we see that $G_1 = F_{(\{s\}, \emptyset)}$ and

$G_3 = F_{(\{s\},\{r\})}$ *explain skeptical support for the observation* $\{b\} = X_q$*, while* $G_2 = F_{(\emptyset,\{r\})}$ *only explains credulous support.*

Remark 2. *This method of instantiation shows that, on the abstract level, hypotheses cannot be represented by independently selectable abducible arguments. The running example shows e.g. that* a *and* d *cannot be added or removed independently. (Cf. remark 1.)*

Related work

We already referred a number of times to Sakama's (2013) model of abduction in argumentation and discussed the differences. On the one hand, we are more general in that we consider a hypothesis to be a change to the AF that is applied as a whole, instead of a set of independently selectable abducible arguments. On the other hand, Sakama's method of computation supports a larger range semantics, including the semi-stable, stable and skeptical preferred semantics. Furthermore, Sakama also considers the possibility that observations force arguments to be rejected, which we do not.

Some of the ideas we applied also appear in work by Wakaki et al. (2009). In their model, an ALP generates an instantiated AF and each hypothesis yields a different division into active/inactive arguments. Unlike our model, as well as Sakama's (2013), Wakaki et al. do not consider removal of arguments as explanation.

Kontarinis et al. (2013) use term rewriting logic to compute changes to an abstract AF with the goal of changing the status of an argument. There are two similarities between their approach and ours. Firstly, we use production rules to generate dialogues and these rules can be seen as a kind of term rewriting rules. Secondly, their approach amounts to rewriting goals into statements to the effect that certain attacks in the AF are enabled or disabled. These statements resemble the moves **PRO:** $x \rightsquigarrow^+ y$ and **PRO:** $x \rightsquigarrow^- y$ in our system. However, they treat attacks as entities that can be enabled or disabled independently. As discussed, different arguments (or in this case attacks associated with arguments) cannot be regarded as independent entities, if the abstract model is instantiated.

Other work dealing with the change of an AF with the goal of changing the status of arguments include Baumann (2012), Baumann and Brewka (2010), Bisquert et al. (2013) and Perotti et al. (2011). Furthermore, Booth et al. (Booth et al. 2013) and Coste-Marquis et al. (2013) frame it as a problem of *belief revision*. Other studies in which changes to AFs are considered include (Boella, Kaci, and van der Torre 2009; Cayrol, Dupin de Saint-Cyr, and Lagasquie-Schiex 2010; Liao, Jin, and Koons 2011; Oikarinen and Woltran 2011).

Conclusions and future work

We developed a model of abduction in abstract argumentation, in which changes to an AF act as explanations for skeptical/credulous support for observations. We presented sound and complete dialogical proof procedures for the main decision problems, i.e., finding explanations for skeptical/credulous support. In addition, we showed that our model of abduction in abstract argumentation can be seen as an abstract form of abduction in logic programming.

As a possible direction for future work, we consider the incorporation of additional criteria for the selection of good explanations, such as minimality with respect to the added and removed arguments/attacks, as well as the use of arbitrary preferences over different abducible AFs. An interesting question is whether the proof theory can be adapted so as to yield only the preferred explanations.

References

Baumann, R., and Brewka, G. 2010. Expanding argumentation frameworks: Enforcing and monotonicity results. In *Proc. COMMA*, 75–86.

Baumann, R. 2012. Normal and strong expansion equivalence for argumentation frameworks. *Artif. Intell.* 193:18–44.

Bisquert, P.; Cayrol, C.; de Saint-Cyr, F. D.; and Lagasquie-Schiex, M.-C. 2013. Enforcement in argumentation is a kind of update. In *SUM* (2013), 30–43.

Boella, G.; Gabbay, D. M.; Perotti, A.; van der Torre, L.; and Villata, S. 2011. Conditional labelling for abstract argumentation. In *TAFA*, 232–248.

Boella, G.; Kaci, S.; and van der Torre, L. 2009. Dynamics in argumentation with single extensions: Attack refinement and the grounded extension (extended version). In *ArgMAS*, 150–159.

Booth, R.; Kaci, S.; Rienstra, T.; and van der Torre, L. 2013. A logical theory about dynamics in abstract argumentation. In *SUM* (2013), 148–161.

Booth, R.; Kaci, S.; and Rienstra, T. 2013. Property-based preferences in abstract argumentation. In *ADT*, 86–100.

Caminada, M. 2010. Preferred semantics as socratic discussion. In *Proceedings of the 11th AI* IA Symposium on Artificial Intelligence*, 209–216.

Cayrol, C.; Dupin de Saint-Cyr, F.; and Lagasquie-Schiex, M.-C. 2010. Change in abstract argumentation frameworks: Adding an argument. *Journal of Artificial Intelligence Research* 38(1):49–84.

Coste-Marquis, S.; Konieczny, S.; Mailly, J.-G.; and Marquis, P. 2013. On the revision of argumentation systems: Minimal change of arguments status. *Proc. TAFA*.

2013. *Scalable Uncertainty Management - 7th International Conference, SUM 2013, Washington, DC, USA, September 16-18, 2013. Proceedings*.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321–358.

Inoue, K., and Sakama, C. 1995. Abductive framework for nonmonotonic theory change. In *IJCAI*, 204–210. Morgan Kaufmann.

Inoue, K., and Sakama, C. 1999. Computing extended abduction through transaction programs. *Ann. Math. Artif. Intell.* 25(3-4):339–367.

Kontarinis, D.; Bonzon, E.; Maudet, N.; Perotti, A.; van der Torre, L.; and Villata, S. 2013. Rewriting rules for the computation of goal-oriented changes in an argumentation system. In *Computational Logic in Multi-Agent Systems*. Springer. 51–68.

Liao, B.; Jin, L.; and Koons, R. C. 2011. Dynamics of argumentation systems: A division-based method. *Artif. Intell.* 175(11):1790–1814.

Modgil, S., and Caminada, M. 2009. Proof theories and algorithms for abstract argumentation frameworks. In *Argumentation in Artificial Intelligence*. 105–129.

Oikarinen, E., and Woltran, S. 2011. Characterizing strong equivalence for argumentation frameworks. *Artificial intelligence* 175(14-15):1985–2009.

Przymusinski, T. C. 1990. The well-founded semantics coincides with the three-valued stable semantics. *Fundam. Inform.* 13(4):445–463.

Sakama, C. 2013. Abduction in argumentation frameworks and its use in debate games. In *Proceedings of the 1st International Workshop on Argument for Agreement and Assurance (AAA)*.

Wakaki, T.; Nitta, K.; and Sawamura, H. 2009. Computing abductive argumentation in answer set programming. In *Proc. ArgMAS*, 195–215.

Wu, Y.; Caminada, M.; and Gabbay, D. M. 2009. Complete extensions in argumentation coincide with 3-valued stable models in logic programming. *Studia Logica* 93(2-3):383–403.

Non-Monotonic Reasoning and Story Comprehension

Irene-Anna Diakidoy
University of Cyprus
eddiak@ucy.ac.cy

Antonis Kakas
University of Cyprus
antonis@cs.ucy.ac.cy

Loizos Michael
Open University of Cyprus
loizos@ouc.ac.cy

Rob Miller
University College London
rsm@ucl.ac.uk

Abstract

This paper develops a Reasoning about Actions and Change framework integrated with Default Reasoning, suitable as a Knowledge Representation and Reasoning framework for Story Comprehension. The proposed framework, which is guided strongly by existing knowhow from the Psychology of Reading and Comprehension, is based on the theory of argumentation from AI. It uses argumentation to capture appropriate solutions to the frame, ramification and qualification problems and generalizations of these problems required for text comprehension. In this first part of the study the work concentrates on the central problem of integration (or elaboration) of the explicit information from the narrative in the text with the implicit (in the reader's mind) common sense world knowledge pertaining to the topic(s) of the story given in the text. We also report on our empirical efforts to gather background common sense world knowledge used by humans when reading a story and to evaluate, through a prototype system, the ability of our approach to capture both the majority and the variability of understanding of a story by the human readers in the experiments.

Introduction

Text comprehension has long been identified as a key test for Artificial Intelligence (AI). Aside from its central position in many forms of the Turing Test, it is clear that human computer interaction could benefit enormously from this and other forms of natural language processing. The rise of computing over the Internet, where so much data is in the form of textual information, has given even greater importance to this topic. This paper reports on a research program aiming to learn from the (extensive) study of text comprehension in Psychology in order to draw guidelines for developing frameworks for automating narrative text comprehension and in particular, *story comprehension* (SC).

Our research program brings together knowhow from Psychology and AI, in particular, our understanding of Reasoning about Actions and Change and Argumentation in AI, to provide a formal framework of representation and a computational framework for SC, that can be empirically evaluated and iteratively developed given the results of the evaluation. This empirical evaluation, which forms an important part of the program, is based on the following methodology: (i) set up a set of stories and a set of questions to test different aspects of story comprehension; (ii) harness the world knowledge on which human readers base their comprehension; (iii) use this world knowledge in our framework and automated system and compare its comprehension behaviour with that of the source of the world knowledge.

In this paper we will concentrate on the development of an appropriate Reasoning about Actions and Change and Default Reasoning framework for representing narratives extracted from stories together with the background *world knowledge* needed for the underlying central process for story comprehension of *synthesizing and elaborating* the explicit text information with new inferences through the implicit world knowledge of the reader. In order to place this specific consideration in the overall process of story comprehension we present here a brief summary of the problem of story comprehension from the psychological point of view.

A Psychological Account of Story Comprehension

Comprehending text entails the construction of a mental representation of the information contained in the text. However, no text specifies clearly and completely all the implications of text ideas or the relations between them. Therefore, comprehension depends on the ability to mentally represent the text-given information and to generate **bridging and elaborative inferences** that connect and elaborate text ideas resulting in a mental or **comprehension model** of the story. Inference generation is necessary in order to comprehend any text as a whole, i.e., as a single network of interconnected propositions instead of as a series of isolated sentences, and to appreciate the suspense and surprise that characterize narrative texts or stories, in particular (Brewer and Lichtenstein 1982; McNamara and Magliano 2009).

Although inference generation is based on the activation of background **world knowledge**, the process is constrained by text information. Concepts encountered in the text activate related conceptual knowledge in the readers' long-term memory (Kintsch 1988). In the case of stories, knowledge about mental states, emotions, and motivations is also relevant as the events depicted tend to revolve around them. Nevertheless, at any given point in the process, only a small subset of all the possible knowledge-based inferences remain activated and become part of the mental representation: those that connect and elaborate text information in a way that contributes to the **coherence** of the mental model (McNamara and Magliano 2009; Rapp and den Broek 2005).

Inference generation is a task-oriented process that follows the principle of **cognitive economy** enforced by a limited-resource cognitive system.

However, the results of this coherence-driven selection mechanism can easily exceed the limited working memory capacity of the human cognitive system. Therefore, coherence on a more global level is achieved through **higher-level integration** processes that operate to create macro-propositions that generalize or subsume a number of text-encountered concepts and the inferences that connected them. In the process, previously selected information that maintains few connections to other information is dropped from the mental model. This results in a more consolidated network of propositions that serves as the new anchor for processing subsequent text information (Kintsch 1998).

Comprehension also requires an iterative general **revision mechanism** of the mental model that readers construct. The feelings of suspense and surprise that stories aim to create are achieved through discontinuities or changes (in settings, motivations, actions, or consequences) that are not predictable or are wrongly predictable solely on the basis of the mental model created so far. Knowledge about the structure and the function of stories leads readers to expect discontinuities and to use them as triggers to revise their mental model (Zwaan 1994). Therefore, a change in time or setting in the text may serve as a clue for revising parts of the mental model while other parts remain and integrated with subsequent text information.

The interaction of bottom-up and top-down processes for the purposes of coherence carries the possibility of **different** but equally legitimate or **successful comprehension** outcomes. Qualitative and quantitative differences in conceptual and mental state knowledge can give rise to differences between the mental models constructed by different readers. Nevertheless, comprehension is successful if these are primarily differences in elaboration but not in the level of coherence of the final mental model.

In this paper we will focus on the underlying lower-level task of constructing the possibly additional elements of the comprehension model and the process of revising these elements as the story unfolds with only a limited concern on the global requirements of coherence and cognitive economy. Our working hypothesis is that these higher level features of comprehension can be tackled on top of the underlying framework that we are developing in this paper, either at the level of the representational structures and language or with additional computational processes on top of the underlying computational framework defined in this paper. We are also assuming as solved the orthogonal issue of correctly parsing the natural language of the text into some information-equivalent structured (e.g., logical) form that gives us the explicit narrative of the story. This is not to say that this issue is not an important element of narrative text comprehension. Indeed, it may need to be tackled in conjunction with the problems on which we are focusing (since, for example, the problem of de-referencing pronoun and article anaphora could depend on background world knowledge and hence possibly on the higher-level whole comprehension of the text (Levesque, Davis, and Morgenstern 2012).

In the next two sections we will develop an appropriate representation framework using preference based argumentation that enables us to address well all the three major problems of *frame, ramification and qualification* and provide an associated revision process. The implementation of a system discussed after this shows how psychologically-inspired story comprehension can proceed as a sequence of elaboration and revision. The paper then presents, using the empirical methodology suggested by research in psychology, our initial efforts to evaluate how closely the inferences drawn by our framework and system match those given by humans engaged in a story comprehension task.

The following story will be used as a running example.

Story: *It was the night of Christmas Eve. After feeding the animals and cleaning the barn, Papa Joe took his shotgun from above the fireplace and sat out on the porch cleaning it. He had had this shotgun since he was young, and it had never failed him, always making a loud noise when it fired.*

Papa Joe woke up early at dawn, picked up his shotgun and went off to forest. He walked for hours, until the sight of two turkeys in the distance made him stop suddenly. A bird on a tree nearby was cheerfully chirping away, building its nest. He aimed at the first turkey, and pulled the trigger.

After a moment's thought, he opened his shotgun and saw there were no bullets in the shotgun's chamber. He loaded his shotgun, aimed at the turkey and pulled the trigger again. Undisturbed, the bird nearby continued to chirp and build its nest. Papa Joe was very confused. Would this be the first time that his shotgun had let him down?

The story above along with other stories and material used for the evaluation of our approach can be found at `http://cognition.ouc.ac.cy/narrative/`.

KRR for Story Comprehension

We will use methods and results from Argumentation Theory in AI (e.g., (Dung 1995; Modgil and Prakken 2012)) and its links to the area of Reasoning about Action and Change (RAC) with Default Reasoning on the static properties of domains (see (van Harmelen, Lifschitz, and Porter 2008) for an overview) to develop a Knowledge Representation and Reasoning (KRR) framework suitable for Story Comprehension (SC). Our central premise is that SC can be formalized in terms of argumentation accounting for the qualification and the revision of the inferences drawn as we read a story.

The psychological research and understanding of SC will guide us in the way we exploit the know how from AI. The close link between human common sense reasoning, such as that for SC, and argumentation has been recently re-enforced by new psychological evidence (Mercier and Sperber 2011) suggesting that human reasoning is in its general form inherently argumentative. In our proposed approach of KRR for SC the reasoning to construct a comprehension model and its qualification at all levels as the story unfolds will be captured through a uniform acceptability requirement on the arguments that support the conclusions in the model.

The significance of this form of representation for SC is that it makes easy the elaboration of new inferences from the explicit information in the narrative, that, as we discussed

in the introduction, is crucially necessary for the successful comprehension of stories. On the other hand, this easy form of elaboration and the extreme form of qualification that it needs can be mitigated by the requirement, again given from the psychological perspective, that elaborative inferences need to be *grounded* on the narrative and *sceptical* in nature. In other words, the psychological perspective of SC, that also suggests that story comprehension is a process of *"fast thinking"*, leads us to depart from a standard logical view of drawing conclusions based on the truth in all (preferred) models. Instead, the emphasis is turned on building one *grounded and well-founded* model from a collection of solid or sceptical properties that are grounded on the text and follow as unqualified conclusions.

We use a typical RAC language of Fluents, Actions, Times, with an extra sort of ***Actors***. An actor-action pair is an ***event***, and a fluent/event or its negation is a ***literal***. For this paper it suffices to represent times as natural numbers[1] and to assume that time-points are dense between story elements to allow for the realization of indirect effects. Arguments will be build from premises in the knowledge connected to any given story. We will have three types of such knowledge units as premises or basic units of arguments.

Definition 1. Let L be a fluent literal, X a fluent/event literal and S a set of fluent/event literals. A ***unit argument or premise*** has one of following forms:

- ***a unit property argument*** $\texttt{pro}(X, S)$ or $\texttt{prec}(X, S)$;
- ***a unit causal argument*** $\texttt{cau}(X, S)$;
- ***a unit persistence argument*** $\texttt{per}(L, \{L\})$ (which we sometimes write as $\texttt{per}(L, \cdot)$).

These three forms are called ***types*** of unit arguments. A unit argument of any type is denoted by $\texttt{arg}_i(H_i, B_i)$. The two forms of unit property arguments differ in that $\texttt{pro}(X, S)$ relates properties to each other at the same time-point, whereas $\texttt{prec}(X, S)$ aims to capture preconditions that hold at the time-point of an event, under which the event is blocked from bringing about its effects at the subsequent time-point.

With abuse of terminology we will sometimes call these units of arguments, simply as arguments.

The knowledge required for the comprehension of a story comprises of two parts: the explicit knowledge of the narrative extracted from the text of the story and the implicit background knowledge that the reader uses along with the narrative for elaborative inferences about the story.

Definition 2. A ***world knowledge theory*** $\mathcal{W}$ is a set of unit property and causal arguments together with a (partial) irreflexive priority relation on them. A ***narrative*** $\mathcal{N}$ is: a set of observations $\textsc{obs}(X, T)$ for a fluent/event literal X, and a time-point T; together with a (possibly empty) set of (story specific) property or causal unit arguments.

The priority relation in $\mathcal{W}$ would typically reflect the priority of specificity for properties, expressed by unit property arguments $\texttt{pro}(X, S)$, or the priority of precondition properties, expressed by unit property arguments $\texttt{prec}(X, S)$, over causal effects, expressed by unit causal arguments. This priority amongst these basic units of knowledge gives a form of non-monotonic reasoning (NMR) for deriving new properties that hold in the story.

To formalize this NMR we use a form of preference-based argumentation uniformly to capture the static (default) inference of properties at a single time point as well as inferences between different type points, by extending the domain specific priority relation to address the frame problem.

Definition 3. A ***story representation*** $\mathcal{SR} = \langle \mathcal{W}, \mathcal{N}, \succ \rangle$ comprises a world knowledge theory $\mathcal{W}$, a narrative $\mathcal{N}$, and a (partial) irreflexive priority relation $\succ$ extending the one in $\mathcal{W}$ so that: *(i)* $\texttt{cau}(H, B_1) \succ \texttt{per}(\neg H, B_2)$; *(ii)* $\texttt{per}(H, B_1) \succ \texttt{pro}(\neg H, B_2)$. The extended relation $\succ$ may also prioritize between arguments in $\mathcal{N}$ and those in $\mathcal{W}$ (typically the former over the latter).

The first priority condition, namely that causal arguments have priority over persistence arguments, encompasses a solution to the ***frame problem***. When we need to reason with defeasible property information, such as default rules about the normal state of the world in which a story takes place, we are also faced with a ***generalized frame problem***, where "a state of the world persists irrespective of the existence of general state laws". Hence, if we are told that the world is in fact in some exceptional state that violates a general (default) property this will continue to be the case in the future, until we learn of (or derive) some causal information that returns the world into its normal state. The solution to this generalized frame problem is captured succinctly by the second general condition on the priority relation of a story representation and its combination with the first condition.

A representation $\mathcal{SR}$ of our example story (focusing on its ending) may include the following unit arguments in $\mathcal{W}$ and $\mathcal{N}$ (where pj is short for "Papa Joe"):

$c1:\ \texttt{cau}(fired\_at(pj, X), \{aim(pj, X), pull\_trigger(pj)\})$
$c2:\ \texttt{cau}(\neg alive(X), \{fired\_at(pj, X), alive(X)\})$
$c3:\ \texttt{cau}(noise, \{fired\_at(pj, X)\})$
$c4:\ \texttt{cau}(\neg chirp(bird), \{noise, nearby(bird)\})$
$c5:\ \texttt{cau}(gun\_loaded, \{load\_gun\})$
$p1:\ \texttt{prec}(\neg fired\_at(pj, X), \{\neg gun\_loaded\})$
$p2:\ \texttt{pro}(\neg fired\_at(pj, X), \{\neg noise\})$ (story specific)

with $p1 \succ c1$, $p2 \succ c1$; and the following in $\mathcal{N}$:

$\textsc{obs}(alive(turkey), 1)$, $\textsc{obs}(aim(pj, turkey), 1)$,
$\textsc{obs}(pull\_trigger(pj), 1)$, $\textsc{obs}(\neg gun\_loaded, 4)$,
$\textsc{obs}(load\_gun, 5)$, $\textsc{obs}(pull\_trigger(pj), 6)$,
$\textsc{obs}(chirp(bird), 10)$, $\textsc{obs}(nearby(bird), 10)$,

with the exact time-point choices being inconsequential.

As we can see in this example the representation of common sense world knowledge has the form of simple associations between concepts in the language. This stems from a key observation in psychology that typically all world knowledge and irrespective of type is inherently default. It is not in the form of an elaborate formal theory of detailed definitions of concepts, but rather is better regarded as a collection of relatively loose semantic associations between concepts, reflecting typical rather than absolute information. Thus knowledge need not be fully qualified at the represen-

[1] In general, abstract time points called ***scenes*** are useful.

tation level, since it can be qualified via the reasoning process by the relative strength of other (conflicting) associations in the knowledge. In particular, as we will see below, **endogenous qualification** will be tackled by the priority relation in the theory and exogenous qualification by this priority coupled with the requirement that explicit narrative information forms, in effect, non-defeasible arguments.

Argumentation Semantics for Stories

To give the semantics of any given story representation $\mathcal{SR}$ we will formulate a corresponding preference based argumentation framework of the form $\langle Arguments, Disputes, Defences\rangle$. Arguments will be based on sets of timed unit arguments. Since we are required to reason about properties over time, it is necessary that arguments populate some connected subset of the time line.

Definition 4. Let $\mathcal{SR} = \langle \mathcal{W}, \mathcal{N}, \succ\rangle$ be a story representation. A ***(unit) argument tuple*** has the form $\big\langle \mathtt{arg}(H,B), T^h, d; (X,T)\big\rangle$, where, $\mathtt{arg}(H,B)$, is a unit argument in $\mathcal{SR}$, X is a fluent/event literal, $d \in \{\text{F}, \text{B}\}$ is an ***inference type*** of either forwards derivation or backwards derivation by contradiction, and T^h, T are time points. T^h refers to the time-point at which the head of the unit argument applies, while X and T refer to the conclusion drawn using the unit argument in the tuple. An ***interpretation*** Δ of $\mathcal{SR}$ is then defined as a a set of argument tuples. We say Δ ***supports*** a fluent/event literal, X, ***at*** T, if either $\big\langle \mathtt{arg}(H,B), T^h, d; (X,T)\big\rangle \in \Delta$ or $\text{OBS}(X,T) \in \mathcal{N}$. The notion of support is extended to hold on sets of timed literals.

The inference process of how an argument tuple supports a timed literal, and thus is allowed to belong to an interpretation, is made precise by the following definition.

Definition 5. Let Δ be an interpretation and $\big\langle \mathtt{arg}(H,B), T^h, d; (X,T)\big\rangle$ in Δ with $d = F$. Then $\mathtt{arg}(H,B)$ ***applied at*** T^h ***forward derives*** X ***at*** T ***under*** Δ iff $X = H$, $T = T^h$ and Δ supports B at T'. The set $\{\langle Y, T'\rangle \mid Y \in B\}$ is called the ***activation condition*** for the derivation; $T' = T^h$ if $\mathtt{arg}(H,B)$ is of the form $\mathtt{pro}(H,B)$. $T' = T^h - 1$ for the other argument types.
When $d = B$, $\mathtt{arg}(H,B)$ ***applied at*** T^h ***backward derives*** X ***at*** T ***under*** Δ iff $\neg X \in B$ and Δ supports $\{\neg H\}$ at T^h and $B \setminus \{\neg X\}$ at T. The set $\big\{\big\langle \neg H, T^h\big\rangle\big\} \cup \{\langle Y, T\rangle \mid Y \in B \setminus \{\neg X\}\}$ is the ***activation condition***; $T = T^h$ if $\mathtt{arg}(H,B)$ is of the form $\mathtt{pro}(H,B)$. $T = T^h - 1$ for the other argument types.

The framework thus includes ***reasoning by contradiction*** with the defeasible world knowledge. Although the psychological debate on the question to what extent humans reason by contradiction, e.g., by contraposition, (see, e.g., (Johnson-Laird and Yang 2008; Rips 1994)) is still ongoing it is natural for a formal argumentation framework to capture this mode of indirect reasoning (see, e.g., (Kakas, Toni, and Mancarella 2013; Kakas and Mancarella 2013)). One of the main consequences of this is that it gives a form of ***backwards persistence***, e.g., from an observation to support (but not necessarily conclude) that the observed property holds also at previous time points. An argument tuple of the form $\langle \mathtt{per}(L,\cdot), T+1, \text{B}; (\neg L, T)\rangle$ captures the backwards persistence of $\neg L$ from time $T+1$ to T using by contraposition the unit argument of persistence of L from T to $T+1$. We also note that the separation of the inference type (e.g., forwards and backwards) is known to be significant in preference based argumentation (Modgil and Prakken 2012). This will be exploited when we consider the attacking between arguments: their disputes and defences.

To reflect the suggestion by psychology that inferences drawn by readers are strongly tied to the story we require that the activation conditions of argument tuples must be eventually traced on the explicit information in the narrative of the story representation.

Definition 6. An interpretation Δ is ***grounded*** on $\mathcal{SR}$ iff there is a total ordering of Δ such that the activation condition of any tuple $\alpha \in \Delta$ is supported by the set of tuples that precede α in the ordering or by the narrative in $\mathcal{SR}$.

Hence in a grounded interpretation there can be no cycles in the tuples that support their activation conditions and so these will always end with tuples whose activation conditions will be supported directly by the observations in the narrative of the story.

We can now define the argumentation framework corresponding to any given story representation. The central task is to capture through the argumentation semantics the non-monotonic reasoning of linking the narrative to the defeasible information in the world knowledge. In particular, the argumentation will need to capture the ***qualification*** problem, encompassed in this synthesis of the narrative with the world knowledge, both at the level of static reasoning at one time point with default property arguments and at the level of temporal projection from one time point to another.

Definition 7. Let $\mathcal{SR}$ be a story representation. Then the ***corresponding argumentation framework,*** $\big\langle ARG^{\mathcal{SR}}, DIS^{\mathcal{SR}}, DEF^{\mathcal{SR}}\big\rangle$ is defined as follows:

- An argument, A, in $ARG^{\mathcal{SR}}$ is any grounded interpretation of $\mathcal{SR}$.
- Given an argument A then A is ***in conflict*** with $\mathcal{SR}$ iff there exists a tuple $\alpha = \big\langle \mathtt{arg}(H,B), T^h, d; (X,T)\big\rangle$ in A such that $\text{OBS}(\neg X, T) \in \mathcal{N}$ of $\mathcal{SR}$.
- Given two arguments A_1, A_2 then these are ***in (direct) conflict*** with each other iff there exists a tuple $\alpha_2 = \big\langle \mathtt{arg}_2(H_2,B_2), T_2^h, d_2; (X_2,T_2)\big\rangle$ in A_2 and a tuple $\alpha_1 = \big\langle \mathtt{arg}_1(H_1,B_1), T_1^h, d_1; (X_1,T_1)\big\rangle$ in A_1 such that $X_1 = \neg X_2$, $T_1 = T_2$. Given two arguments A_1, A_2 then these are ***in indirect conflict*** with each other iff there exists a tuple $\alpha_2 = \big\langle \mathtt{arg}_2(H_2,B_2), T_2^h, d_2; (X_2,T_2)\big\rangle$ in A_2 and a tuple $\alpha_1 = \big\langle \mathtt{arg}_1(H_1,B_1), T_1^h, d_1; (X_1,T_1)\big\rangle$ in A_1 such that ($d_1 = \text{B}$ or $d_2 = \text{B}$) and $H_1 = \neg H_2$, $T_1^h = T_2^h$.
- Given two arguments A_1, A_2 then A_2 ***disputes*** A_1 and hence $(A_2, A_1) \in DIS^{\mathcal{SR}}$ iff A_2 is in direct or indirect conflict with A_1, and in the case of indirect conflict $d_1 = \text{B}$ holds in the definition of indirect conflict above.
- Argument A_1 ***undercuts*** A_2 iff
  - A_1, A_2 are in direct or indirect conflict via α_1 and α_2,

– when in direct conflict, there exists a tuple $\alpha_1' = \left\langle \texttt{arg}_1'(H_1', B_1'), T_1^{h'}, d_1'; (X_1', T_1') \right\rangle$ in A_1 and a tuple $\alpha_2' = \left\langle \texttt{arg}_2'(H_2', B_2'), T_2^{h'}, d_2'; (X_2', T_2') \right\rangle$ in A_2 such that $\texttt{arg}_1'(H_1', B_1') \succ \texttt{arg}_2'(H_2', B_2')$ and $T_1^{'} = T_2^{'}$ or $T_1^{h'} = T_2^{h'}$.
– when in indirect conflict, then $\texttt{arg}_1(H_1, B_1) \succ \texttt{arg}_2(H_2, B_2)$ where $\texttt{arg}_1(H_1, B_1)$ and $\texttt{arg}_2(H_2, B_2)$ are the unit arguments in α_1 and α_2 respectively.

- Argument A_1 ***defends against*** A_2 and hence $(A_1, A_2) \in DEF^{\mathcal{SR}}$, iff there exists a subset $A_2^{'} \subseteq A_2$ which is in minimal conflict with A_1 (i.e., no proper subset of $A_2^{'}$ is in conflict with A_1) and A_1 undercuts $A_2^{'}$.

Several clarifying comments are in order. Arguments that are in dispute are arguments that support some contrary conclusion at the same time point and hence form counter-arguments for each other. The use of contrapositive reasoning for backwards inference also means that it is possible to have arguments that support conclusions that are not contrary to each other but whose unit arguments have conflicting conclusions. For example, in our running example we can use the causal unit argument, $c1$, to forward derive $fired\_at(pj, X)$ and the property argument $p1$ to backwards derive gun_loaded from $\neg fired\_at(pj, X)$ and despite the fact that the derived facts are not in conflict the unit arguments used concern conflicting conclusions. Hence such arguments are also considered to be in conflict but instead of a direct conflict we say we have an indirect conflict. Not all such indirect conflicts are important. A dispute that results from an indirect conflict of a unit argument used backwards on a unit argument that is used forwards does not have any effect. Such cases are excluded from giving rise to disputes.

This complication in the definitions of conflicts and disputes results from the *defeasible nature* of the world knowledge and the fact we are allowing reasoning by contradiction on such defeasible information. These complications in fact stem from the fact that we are only approximating the proof by contradiction reasoning, capturing this indirectly through contraposition. The study of this is beyond the scope of this paper and the reader is referred to the newly formulated Argumentation Logic (Kakas, Toni, and Mancarella 2013).

Undercuts between arguments require that the undercutting argument does so through a stronger unit or premise argument than some unit argument in the argument that is undercut. The defence relation is build out of undercuts by applying an undercut on minimally conflicting subsets of the argument which we are defending against. Hence these two relations between arguments are asymmetric. Note also that the stronger premise from the undercutting argument does not necessarily need to come from the subset of the unit arguments that supports the conflicting conclusion. Instead, it can come from any part of the undercutting argument to undercut at any point of the chain supporting the activation of the conflicting conclusion. This, as we shall illustrate below, is linked to how the framework addresses the ***ramification problem*** of reasoning with actions and change.

The semantics of a story representation is defined using the corresponding argumentation framework as follows.

Definition 8. Let $\mathcal{SR}$ be a story representation and $\langle ARG^{\mathcal{SR}}, DIS^{\mathcal{SR}}, DEF^{\mathcal{SR}} \rangle$ its corresponding argumentation framework. An argument Δ is ***acceptable*** in $\mathcal{SR}$ iff

- Δ is not in conflict with $\mathcal{SR}$ nor in direct conflict with Δ.
- No argument A undercuts Δ.
- For any argument A that minimally disputes Δ, Δ defends against A.

Acceptable arguments are called ***comprehension models*** of $\mathcal{SR}$. Given a comprehension model Δ, a timed fluent literal (X, T) is ***entailed by*** $\mathcal{SR}$ iff this is supported by Δ.

The above definition of comprehension model and story entailment is of a sceptical form where, apart from the fact that all conclusions must be ground on the narrative, they must also not be non-deterministic in the sense that there can not exist another comprehension model where the negative conclusion is entailed. Separating disputes and undercuts and identifying defences with undercuts facilitates this sceptical form of entailment. Undercuts (see, e.g., (Modgil and Prakken 2012) for some recent discussion) are strong counter-claims whose existence means that the attacked set is inappropriate for sceptical conclusions whereas disputes are weak counter-claims that could be defended or invalidated by extending the argument to undercut them back. Also the explicit condition that an acceptable argument should not be undercut even if it can undercut back means that this definition does not allow non-deterministic choices for arguments that can defend themselves.

To illustrate the formal framework, how arguments are constructed and how a comprehension of a story is formed through acceptable arguments let us consider our example story starting from the end of the second paragraph, corresponding to time-points 1-3 in the example narrative. Note that the empty Δ supports $aim(pj, turkey)$ and $pull\_trigger(pj)$ at 1. Hence, $c1$ on 2 forward activates $fired\_at(pj, turkey)$ at 2 under the empty argument, Δ. We can thus populate Δ with $\langle c1, 2, \text{F}; (fired\_at(pj, turkey), 2)\rangle$. Similarly, we can include $\langle \texttt{per}(alive(turkey), \cdot), 2, \text{F}; (alive(turkey), 2)\rangle$ in the new Δ. Under this latter Δ, $c2$ on 3 forward activates $\neg alive(turkey)$ at 3, allowing us to further extend Δ with $\langle c2, 3, \text{F}; (\neg alive(turkey), 3)\rangle$. The resulting Δ is a grounded interpretation that supports $\neg alive(turkey)$ at 3. It is based on this inference, that we expect readers to respond that the first turkey is dead, when asked about its status at this point, since no other argument grounded on the narrative (thus far) can support a qualification argument to this inference. Note also that we can include in Δ the tuple $\langle p1, 2, \text{B}; (gun\_loaded, 1)\rangle$ to support, using backwards (contrapositive) reasoning with $p1$, the conclusion that the gun was loaded when it was fired at time 1.

Reading the first sentence of the third paragraph, we learn that $\text{OBS}(\neg gun\_loaded, 4)$. We now expect that this new piece of evidence will lead readers to revise their inferences as now we have an argument to support the conclusion $\neg fired\_at(pj, turkey)$ based on the stronger (qualifying)

unit argument of $p1$. For this we need to support the activation condition of $p1$ at time 1, i.e., to support $\neg gun\_loaded$ at 1. To do this we can use the argument tuples:

$\langle \texttt{per}(gun\_loaded, \cdot), 4, \text{B}; (\neg gun\_loaded, 3)\rangle$
$\langle \texttt{per}(gun\_loaded, \cdot), 3, \text{B}; (\neg gun\_loaded, 2)\rangle$
$\langle \texttt{per}(gun\_loaded, \cdot), 2, \text{B}; (\neg gun\_loaded, 1)\rangle$

which support the conclusion that the gun was also unloaded before it was observed to be so. This uses $\texttt{per}(gun\_loaded, \cdot)$ contrapositively to backward activate the unit argument of persistence, e.g., had the gun been loaded at 3, it would have been so at 4 which would contradict the story. Note that this backwards inference of $\neg gun\_loaded$ would be qualified by a causal argument for $\neg gun\_loaded$ at any time earlier than 4, e.g., if the world knowledge contained the unit argument

$c: \texttt{cau}(\neg gun\_loaded, \{pull\_trigger(pj)\})$

This then supports an indirect conflict at time 2 with the forwards persistence of gun_loaded from 1 to 2 and due to the stronger nature of unit causal over persistence arguments the backwards inference of $\neg gun\_loaded$ is undercut and so cannot belong to an acceptable argument.

Assuming that c is absent, the argument, Δ_1, consisting of these three "persistence" tuples is in conflict on gun_loaded on 1 with the argument Δ above. Each argument disputes the other and in fact neither can form an acceptable argument. If we extend Δ_1 with the tuple $\langle p1, 2, \text{F}; (\neg fired\_at(pj, turkey), 2)\rangle$ then this can now undercut and thus defend against Δ using the priority of $p1$ over $c1$. Therefore the extended Δ_1 is acceptable and the conclusion $\neg fired\_at(pj, turkey)$ at 2 is drawn revising the previous conclusions drawn from Δ. The process of understanding our story may then proceed by extending Δ_1, with $\langle \texttt{per}(alive(turkey), \cdot), T, \text{F}; (alive(turkey), T)\rangle$ for $T = 2, 3, 4$, resulting in a model that supports $alive(turkey)$ at 4. It is based on this inference that we expect readers to respond that the first turkey is alive at 4.

Continuing with the story, after Papa Joe loads the gun and fires again, we can support by forward inferences that the gun fired, that noise was caused, and that the bird stopped chirping, through a chaining of the unit arguments $c1, c3, c4$. But $\text{OBS}(chirp(bird), 10)$ supports disputes on all these through the repeated backwards use of the same unit arguments grounded on this observation. We thus have an **exogenous qualification** effect where these conclusions can not be sceptical and so will not be supported by any comprehension model. But if we also consider the stronger (story specific) information in $p2$, that this gun does not fire without a noise, together with the backwards inference of $\neg noise$ an argument that contains these can undercut the firing of the gun at time 2 and thus defend against disputes that are grounded on $pull\_triger$ at 1 and the gun firing. As a result, we have the effect of blocking the ramification of the causation of $noise$ and so $\neg noise$ (as well as $\neg fired\_at(pj, turkey)$) are sceptically concluded. Readers, indeed respond in this way.

With this latter part of the example story we see how our framework addresses the *ramification problem* and its non-trivial interaction with the qualification problem (Thielscher 2001). In fact, a generalized form of this problem is addressed where the ramifications are not chained only through causal laws but through any of the forms of inference we have in the framework — causal, property or persistence — and through any of the type of inference — forwards or backwards by contradiction.

Algorithm 1 Computing a Comprehension Model

input: story $\mathcal{SR}$, partitioned in a list of k blocks, and a set of questions $Q[b]$ associated with each $\mathcal{SR}$ block b.
Set $\mathbb{G}[0]$ to be the empty graph.
for every $b = 1, 2, \ldots, k$ **do**
  Let $\mathcal{SR}[b]$ be the restriction of $\mathcal{SR}$ up to its b-th block.
  Let $\mathbb{G}[b] := graph(\mathbb{G}[b-1], \mathcal{SR}[b])$ be the new graph.
  Let $\Pi[b] := retract(\Delta[b-1], \mathbb{G}[b], \mathcal{SR}[b])$.
  Let $\Delta[b] := elaborate(\Pi[b], \mathbb{G}[b], \mathcal{SR}[b])$.
  Answer $Q[b]$ with the comprehension model $\Delta[b]$.
end for

A comprehension model can be tested, as is often done in psychology, through a series of multiple-choice questions.

Definition 9. Let M be a comprehension model of a story representation $\mathcal{SR}$. A possible answer,"X at T", to a question is ***accepted***, respectively ***rejected***, iff "X at T" (respectively "$\neg X$ at T") is supported by M. Otherwise, we say that the question is ***allowed or possible*** by M.

In some cases, we may want to extend the notion of a comprehension model to allow some non-sceptical entailments. This is needed to reflect the situation when a reader cannot find a sceptical answer to a question and chooses between two or more allowed answers. This can be captured by allowing each such answer to be supported by a more general notion of acceptability such as the admissibility criterion of argumentation semantics. For this, we can drop the condition that Δ is not undercut by any argument and allow weaker defences, through disputes, to defend back on a dispute that is not at the same time an undercut.

Finally, we note that a comprehension model need not be complete as it does not need to contain all possible sceptical conclusions that can be drawn from the narrative and the entire world knowledge. It is a *subset* of this, given by the subset of the available world knowledge that readers choose to use. This incompleteness of the comprehension model is required for important cognitive economy and coherence properties of comprehension, as trivially a "full model" is contrary to the notion of coherence.

Computing Comprehension Models

The computational procedure below constructs a comprehension model, by iteratively reading a new part of the story $\mathcal{SR}$, retracting existing inferences that are no longer appropriate, and including new inferences that are triggered as a result of the new story part. Each part of the story may include more than one observation, much in the same way that human readers may be asked to read multiple sentences in the story before being asked to answer a question. We shall call each story part of interest a ***block***, and shall assume that it is provided as input to the computational procedure.

At a high level the procedure proceeds as in Algorithm 1. The story is read one block at a time. After each block

Algorithm 2 Elaborating a Comprehension Model

input: provisional comprehension model Π, graph $\mathbb{G}$, story $\mathcal{SR}$; all inputs possibly restricted up to some block.
repeat
  Let $\mathbb{G} := retract(\Pi, \mathbb{G}, \mathcal{SR})$.
  Let E include all tuples $\langle \mathtt{arg}(H, B), T^h, d; (X, T) \rangle$ such that $\mathtt{arg}(H, B)$ activates X at T under Π.
  Let $\Pi := \Delta$.
  Let $\Pi := expand(\Delta, E, \mathbb{G})$.
until $\Delta = \Pi$
output: elaborated comprehension model Δ.

of $\mathcal{SR}$ is read, a directed acyclic graph $\mathbb{G}[b]$ is maintained which succinctly encodes all interpretations that are relevant for $\mathcal{SR}$ up to its b-th block. Starting from $\mathbb{G}[b-1]$, a new tuple is added as a vertex if it is possible to add a directed edge to each $\langle X, T \rangle$ in the tuple's condition from either an observation $\textsc{obs}(X, T)$ in the narrative of $\mathcal{SR}[b]$, or from a tuple $\langle \mathtt{arg}(H, B), T^h, d; (X, T) \rangle$ already in $\mathbb{G}[b]$. In effect, then, edges correspond to the notion of support from the preceding section, and the graph is the maximal grounded interpretation given the part of the story read.

Once graph $\mathbb{G}[b]$ is computed, it is used to revise the comprehension model $\Delta[b-1]$ so that it takes into account the observations in $\mathcal{SR}[b]$. The revision proceeds in two steps.

In the first step, the tuples in $\Delta[b-1]$ are considered in the order in which they were added, and each one is checked to see whether it should remain in the comprehension model. Any tuple in $\Delta[b-1]$ that is undercut by the tuples in $\mathbb{G}[b]$, or disputed and cannot be defended, is retracted, and is not included in the provisional set $\Pi[b]$. As a result of a retraction, any tuple $\langle \mathtt{arg}(H, B), T^h, d; (X, T) \rangle \in \Delta[b-1]$ such that $\mathtt{arg}(H, B)$ no longer activates X at T under $\Pi[b]$ is also retracted and is not included in $\Pi[b]$. This step guarantees that the argument $\Pi[b]$ is trivially acceptable.

In the second step, the provisional set $\Pi[b]$, which is itself a comprehension model (but likely a highly incomplete one), is elaborated with new inferences that follow. The elaboration process proceeds as in Algorithm 2. Since the provisional comprehension model Π effectively includes only unit arguments that are "strong" against the attacks from $\mathbb{G}$, it is used to remove (only as part of the local computation of this procedure) any weak arguments from $\mathbb{G}$ itself (i.e., arguments that are undercut), and any arguments that depend on the former to activate their inferences. This step, then, ensures that all arguments (subsets of G) that are defended are no longer part of the revised G, in effect accommodating the minimality condition for attacking sets. It then considers all arguments that activate their inferences in the provisional comprehension model. The comprehension model is expanded with a new tuple from E if the tuple is not in conflict with the story nor in direct conflict with the current model Δ, and if "attacked" by arguments in $\mathbb{G}$ then these arguments do not undercut Δ, and Δ undercuts back. Only arguments coming from the *revised* graph $\mathbb{G}$ are considered, as per the minimality criterion on considered attacks.

The elaboration process adds only "strong" arguments in the comprehension model, retaining its property as a comprehension model. The discussion above forms the basis for the proof of the following theorem:

Theorem 1. Algorithm 1 runs in time that is polynomial in the size of $\mathcal{SR}$ and the number of time-points of interest, and returns a comprehension model of the story.

Proof sketch. Correctness follows from our earlier discussion. Regarding running time: The number of iterations of the top-level algorithm is at most linear in the relevant parameters. In constructing the graph $\mathbb{G}[b]$, each pair of elements (unit arguments or observations at some time-point) in $\mathcal{SR}[b]$ is considered once, for a constant number of operations. The same is the case for the retraction process in the subsequent step of the algorithm. Finally, the loop of the elaboration process repeats at most a linear in the relevant parameters number of times, since at least one new tuple is included in Π in every loop. Within each loop, each step considers each pair of elements (unit arguments or observations at some time-point) in $\mathcal{SR}[b]$ once, for a constant number of operations. The claim follows. QED

The computational processes presented above have been implemented using Prolog, along with an accompanying high-level language for representing narratives, background knowledge, and multiple-choice questions. Without going into details, the language allows the user to specify a sequence of sessions of the form `session(s(B),Qs,Vs)`, where `B` is the next story block to read, `Qs` is the set of questions to be answered afterwards, and `Vs` is the set of fluents made visible in a comprehension model returned to the user.

The narrative itself is represented by a sequence of statements of the form `s(B) :: X at T`, where `B` is the block in which the statement belongs (with possibly multiple statements belonging in the same block), `X` is a fluent or action, and `T` is the time-point at which it is observed.

The background knowledge is represented by clauses of the form `p(N) :: A, B, ..., C implies X` or `c(N) :: A, B, ..., C causes X`, where `p` or `c` shows a property or causal clause, `N` is the name of the rule, `A, B, ..., C` is the rule's body, and `X` is the rule's head. Negations are represented by prefixing a fluent or action in the body or head with the minus symbol. Variables can be used in the fluents or actions to represent relational rules. Preferences between clauses are represented by statements of the form `p(N1) >> c(N2)` with the natural reading.

Questions are represented by clauses of the form `q(N) ?? (X1 at T1, ..., X2 at T2) ; ...`, where `N` is the name of the question, `(X1 at T1, ..., X2 at T2)` is the first possible answer as a conjunction of fluents or actions that need to hold at their respective time-points, and `;` separates the answers. The question is always the same: "Which of the following choices is the case?".

The implemented system demonstrates real modularity and elaboration tolerance, allowing as input any story narrative or background knowledge in the given syntax, always appropriately qualifying the given information to compute a comprehension model. The system is available at `http://cognition.ouc.ac.cy/narrative/`.

Evaluation through Empirical Studies

In the first part of the evaluation of our approach we carried a *psychological study* to ascertain the world knowledge that is activated to successfully comprehend example stories such as our example story on the basis of data obtained from human readers. We were interested both in the outcomes of successful comprehension and the world knowledge that contributed to the human comprehension. We developed a set of inferential questions to follow the reading of prespecified story segments. These assessed the extent to which readers connected, explained, and elaborated key story elements. Readers were instructed to answer each question and to justify their answers using a "think-aloud" method of answering questions while reading in order to reveal the world knowledge that they had used.

The qualitative data from the readers was pooled together and analysed as to the frequencies of the types of responses in conjunction with the information given in justifications and think-aloud protocols. For example, the data indicated that all readers considered Papa Joe to be living on a farm or in a village (q.01, "Where does Papa Joe live?") and that all readers attributed an intention of Papa Joe to hunt (q.06, "What was Papa Joe doing in the forest?"). An interesting example of variability occurred in the answers for the group of questions 07,08,10,11, asking about the status of the turkeys at various stages in the story. The majority of participants followed a comprehension model which was revised between the first turkey being dead and alive. However, a minority of participants consistently answered that both turkeys were alive. These readers had defeated the causal arguments that supported the inference that the first turkey was dead, perhaps based on an expectation that the desire of the protagonist for turkey would be met with complications. We believe that such expectations can be generated from standard *story knowledge* in the same way as we draw other elaborative inferences from WK.

Evaluation of the system

Using the empirical data discussed above, we tested our framework's ability to capture the majority answers and account for their variability. The parts of our example story representation relevant to questions 01 and 06 are as follows:

s(1) :: night at 0.
s(1) :: xmasEve at 0.
s(1) :: clean(pj,barn) at 0.
s(2) :: xmasDay at 1.
s(2) :: gun(pjGun) at 1.
s(2) :: longWalk(pj) at 1.
s(2) :: animal(turkey1) at 2.
s(2) :: animal(turkey2) at 2.
s(2) :: alive(turkey1) at 2.
s(2) :: alive(turkey2) at 2.
s(2) :: chirp(bird) at 2.
s(2) :: nearby(bird) at 2.
s(2) :: aim(pjGun,turkey1) at 2.
s(2) :: pulltrigger(pjGun) at 2.

The two questions are answered after reading, respectively, the first and second blocks of the story above:

session(s(1),[q(01)],_). session(s(2),[q(06)],_).

with their corresponding multiple-choice answers being:

q(01) ??
lives(pj,city) at 0; lives(pj,hotel) at 0;
lives(pj,farm) at 0; lives(pj,village) at 0.

q(06) ??
motive(in(pj,forest),practiceShoot) at 3;
motive(in(pj,forest),huntFor(food)) at 3;
(motive(in(pj,forest),catch(turkey1)) at 3,
motive(in(pj,forest),catch(turkey2)) at 3);
motive(in(pj,forest),hearBirdsChirp) at 3.

To answer the first question, the system uses the following background knowledge:

p(11) :: has(home(pj),barn) implies lives(pj,countrySide).
p(12) :: true implies -lives(pj,hotel).
p(13) :: true implies lives(pj,city).
p(14) :: has(home(pj),barn) implies -lives(pj,city).
p(15) :: clean(pj,barn) implies at(pj,barn).
p(16) :: at(pj,home), at(pj,barn) implies has(home(pj),barn).
p(17) :: xmasEve, night implies at(pj,home).
p(18) :: working(pj) implies -at(pj,home).
p(111) :: lives(pj,countrySide) implies lives(pj,village).
p(112) :: lives(pj,countrySide) implies lives(pj,farm).
p(113) :: lives(pj,village) implies -lives(pj,farm).
p(114) :: lives(pj,farm) implies -lives(pj,village).
p(14) >> p(13). p(18) >> p(17).

By the story information, p(17) implies at(pj,home), without being attacked by p(18), since nothing is said in the story about Papa Joe working. Also by the story information, p(15) implies at(pj,barn). Combining the inferences from above, p(16) implies has(home(pj),barn), and p(11) implies lives(pj,countrySide). p(12) immediately dismisses the case of living in a hotel (as people usually do not), whereas p(14) overrides p(13) and dismisses the case of living in the city. Yet, the background knowledge cannot unambiguously derive one of the remaining two answers. In fact, p(111), p(112), p(113), p(114) give arguments for either of the two choices. This is in line with the variability in the empirical data in terms of human answers to the first question.

To answer the second question, the system uses the following background knowledge:

p(21) :: want(pj,foodFor(dinner)) implies
motive(in(pj,forest),huntFor(food)).
p(22) :: hunter(pj) implies
motive(in(pj,forest),huntFor(food)).
p(23) :: firedat(pjGun,X), animal(X) implies
-motive(in(pj,forest),catch(X)).
p(24) :: firedat(pjGun,X), animal(X) implies
-motive(in(pj,forest),hearBirdsChirp).
p(25) :: xmasDay implies
want(pj,foodFor(dinner)).
p(26) :: longWalk(pj) implies
-motive(in(pj,forest),practiceShooting).
p(27) :: xmasDay implies
-motive(in(pj,forest),practiceShooting).

By the story information and parts of the background knowledge not shown above, we can derive that Papa Joe is a hunter, and that he has fired at a turkey. From the first inference, p(22) already implies that the motivation is to hunt for food. The same inference can be derived by p(25) and p(21), although for a different reason. At the same time, p(23) and p(24) dismiss the possibility of the motivation being to catch the two turkeys or to hear birds chirp, whereas story information along with either p(26) or p(27) dismiss also the possibility of the motivation being to practice shooting.

The background knowledge above follows evidence from the participant responses in our psychological study that the motives in the answers of the second question can be "derived" from higher-level desires or goals of the actor. Such high-level desires and intentions are examples of *generalizations* that contribute to the coherence of comprehension, and to the creation of *expectations* in readers about the course of action that the story might follow in relation to fulfilling desires and achieving intentions of the protagonists.

Related Work

Automated story understanding has been an ongoing field of AI research for the last forty years, starting with the planning and goal-oriented approaches of Schank, Abelson, Dyer and others (Schank and Abelson 1977; Dyer 1983); for a good overview see (Mueller 2002) and the website (Mueller 2013). Logic-related approaches have largely been concerned with the development of appropriate representations, translations or annotations of narratives, with the implicit or explicit assumption that standard deduction or logical reasoning techniques can subsequently be applied to these. For example, the work of Mueller (Mueller 2003), which in terms of story representation is most closely related to our approach, equates various modes of story understanding with the solving of satisfiability problems. (Niehaus and Young 2009) models understanding as partial order planning, and is also of interest here because of a methodology that includes a controlled comparison with human readers.

To our knowledge there has been very little work relating story comprehension with computational argumentation, an exception being (Bex and Verheij 2013), in which a case is made for combining narrative and argumentation techniques in the context of legal reasoning, and with which our argumentation framework shares important similarities. Argumentation for reasoning about actions and change, on which our formal framework builds, has been studied in (Vo and Foo 2005; Michael and Kakas 2009).

Many other authors have emphasized the importance of commonsense knowledge and reasoning in story comprehension (Silva and Montgomery 1977; Dahlgren, McDowell, and Stabler 1989; Riloff 1999; Mueller 2004; 2009; Verheij 2009; Elson and McKeown 2009; Michael 2010), and indeed how it can offer a basis for story comprehension tasks beyond question answering (Michael 2013b).

Conclusions and Future Work

We have set up a conceptual framework for story comprehension by fusing together knowhow from the psychology of text comprehension with established AI techniques and theory in the areas of Reasoning about Actions and Change and Argumentation. We have developed a proof of concept automated system to evaluate the applicability of our framework through a similar empirical process of evaluating human readers. We are currently, carrying out psychological experiments with other stories to harness world knowledge and test our system against the human readers.

There are still several problems that we need to address to complete a fully automated approach to SC, over and above the problem of extracting through Natural Language Processing techniques the narrative from the free format text. Two major such problems for our immediate future work are (a) to address further the computational aspects of the challenges of cognitive economy and coherence and (b) the systematic extraction or acquisition of common sense world knowledge. For the first of these we will investigate how this can be addressed by applying "computational heuristics" on top of (and without the need to reexamine) the solid semantic framework that we have developed thus far, drawing again from psychology to formulate such heuristics. In particular, we expect that the psychological studies will guide us in modularly introducing computational operators such as *selection, dropping and generalization operators* so that we can improve the coherence of the computed models.

For the problem of the systematic acquisition of world knowledge we aim to source this (semi)-automatically from the Web. For this we could build on lexical databases such as WordNet (Miller 1995), FrameNet (Baker, Fillmore, and Lowe 1998), and PropBank (Palmer, Gildea, and Kingsbury 2005), exploring the possibility of populating the world knowledge theories using archives for common sense knowledge (e.g., Cyc (Lenat 1995)) or through the automated extraction of commonsense knowledge from text using natural language processing (Michael and Valiant 2008), and appealing to textual entailment for the semantics of the extracted knowledge (Michael 2009; 2013a).

We envisage that the strong inter-disciplinary nature of our work can provide a concrete and important test bed for evaluating the development of NMR frameworks in AI while at the same time offering valuable feedback for Psychology.

References

Baker, C. F.; Fillmore, C. J.; and Lowe, J. B. 1998. The Berkeley FrameNet Project. In *Proc. of 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, 86–90.

Bex, F., and Verheij, B. 2013. Legal Stories and the Process of Proof. *Artif. Intell. Law* 21(3):253–278.

Brewer, W., and Lichtenstein, E. 1982. Stories are to Entertain: A Structural-Affect Theory of Stories. *Journal of Pragmatics* 6:473–486.

Dahlgren, K.; McDowell, J.; and Stabler, E. 1989. Knowledge Representation for Commonsense Reasoning with Text. *Computational Linguistics* 15(3):149–170.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artif. Intell.* 77(2):321–358.

Dyer, M. G. 1983. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension.* MIT Press, Cambridge, MA.

Elson, D., and McKeown, K. 2009. Extending and Evaluating a Platform for Story Understanding. In *Proc. of AAAI Symposium on Intelligent Narrative Technologies II*.

Johnson-Laird, P. N., and Yang, Y. 2008. Mental Logic, Mental Models, and Simulations of Human Deductive Reasoning. In Sun, R., ed., *The Cambridge Handbook of Computational Psychology*, 339–358.

Kakas, A., and Mancarella, P. 2013. On the Semantics of Abstract Argumentation. *Logic Computation* 23:991–1015.

Kakas, A.; Toni, F.; and Mancarella, P. 2013. Argumentation for Propositional Logic and Nonmonotonic Reasoning. In *Proc. of 11th International Symposium on Logical Formalizations of Commonsense Reasoning*.

Kintsch, W. 1988. The Role of Knowledge in Discourse Comprehension: A Construction-Integration Model. *Psychological Review* 95:163–182.

Kintsch, W. 1998. *Comprehension: A Paradigm of Cognition*. NY: Cambridge University Press.

Lenat, D. B. 1995. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Commun. ACM* 38(11):32–38.

Levesque, H. J.; Davis, E.; and Morgenstern, L. 2012. The Winograd Schema Challenge. In *Proc. of 13th International Conference on Principles of Knowledge Representation and Reasoning*, 552–561.

McNamara, D. S., and Magliano, J. 2009. Toward a Comprehensive Model of Comprehension. *The Psychology of Learning and Motivation* 51:297–384.

Mercier, H., and Sperber, D. 2011. Why Do Humans Reason? Arguments for an Argumentative Theory. *Behavioral and Brain Sciences* 34(2):57–74.

Michael, L., and Kakas, A. C. 2009. Knowledge Qualification through Argumentation. In *Proc. of 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, 209–222.

Michael, L., and Valiant, L. G. 2008. A First Experimental Demonstration of Massive Knowledge Infusion. In *Proc. of 11th International Conference on Principles of Knowledge Representation and Reasoning*, 378–389.

Michael, L. 2009. Reading Between the Lines. In *Proc. of 21st International Joint Conference on Artificial Intelligence*, 1525–1530.

Michael, L. 2010. Computability of Narrative. In *Proc. of AAAI Symposium on Computational Models of Narrative*.

Michael, L. 2013a. Machines with Websense. In *Proc. of 11th International Symposium on Logical Formalizations of Commonsense Reasoning*.

Michael, L. 2013b. Story Understanding... Calculemus! In *Proc. of 11th International Symposium on Logical Formalizations of Commonsense Reasoning*.

Miller, G. A. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38(11):39–41.

Modgil, S., and Prakken, H. 2012. A General Account of Argumentation with Preferences. *Artif. Intell.* 195:361–397.

Mueller, E. T. 2002. Story Understanding. In Nadel, L., ed., *Encyclopedia of Cognitive Science*, volume 4, 238–246. London: Macmillan Reference.

Mueller, E. 2003. Story Understanding through Multi-Representation Model Construction. In Hirst, G., and Nirenburg, S., eds., *Proc. of the HLT-NAACL 2003 Workshop on Text Meaning*, 46–53.

Mueller, E. 2004. Understanding Script-Based Stories Using Commonsense Reasoning. *Cognitive Systems Research* 5(4):307–340.

Mueller, E. 2009. Story Understanding through Model Finding. In *Proc. of Workshop on Advancing Computational Models of Narrative*.

Mueller, E. 2013. Story Understanding Resources. http://xenia.media.mit.edu/ mueller/storyund/storyres.html. Accessed February 28, 2013.

Niehaus, J., and Young, R. M. 2009. A Computational Model of Inferencing in Narrative. In *Proc. of AAAI Symposium on Intelligent Narrative Technologies II*.

Palmer, M.; Gildea, D.; and Kingsbury, P. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics* 31(1):71–106.

Rapp, D., and den Broek, P. V. 2005. Dynamic Text Comprehension: An Integrative View of Reading. *Current Directions in Psychological Science* 14:297–384.

Riloff, E. 1999. Information Extraction as a Stepping Stone Toward Story Understanding. In Ram, A., and Moorman, K., eds., *Understanding Language Understanding: Computational Models of Reading*, 435–460. The MIT Press.

Rips, L. 1994. *The Psychology of Proof*. MIT Press.

Schank, R. C., and Abelson, R. P. 1977. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum, Hillsdale, NJ.

Silva, G., and Montgomery, C. A. 1977. Knowledge Representation for Automated Understanding of Natural Language Discourse. *Computers and the Humanities* 11(4):223–234.

Thielscher, M. 2001. The Qualification Problem: A Solution to the Problem of Anomalous Models. *Artif. Intell.* 131(1–2):1–37.

van Harmelen, F.; Lifschitz, V.; and Porter, B. 2008. *Handbook of Knowledge Representation*. Elsevier Science.

Verheij, B. 2009. Argumentation Schemes, Stories and Legal Evidence. In *Proc. of Workshop on Advancing Computational Models of Narrative*.

Vo, Q. B., and Foo, N. Y. 2005. Reasoning about Action: An Argumentation-Theoretic Approach. *J. Artif. Intell. Res.* 24:465–518.

Zwaan, R. A. 1994. Effect of Genre Expectations on Text Comprehension. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 20:920–933.

Tableau vs. Sequent Calculi for Minimal Entailment

Olaf Beyersdorff[*] and **Leroy Chew**[†]
School of Computing, University of Leeds, UK

Abstract

In this paper we compare two proof systems for minimal entailment: a tableau system $OTAB$ and a sequent calculus MLK, both developed by Olivetti (1992). Our main result shows that $OTAB$-proofs can be efficiently translated into MLK-proofs, *i.e.*, MLK p-simulates $OTAB$. The simulation is technically very involved and answers an open question posed by Olivetti (1992) on the relation between the two calculi. We also show that the two systems are exponentially separated, *i.e.*, there are formulas which have polynomial-size MLK-proofs, but require exponential-size $OTAB$-proofs.

Introduction

Minimal entailment is the most important special case of circumscription, which in turn is one of the main formalisms for non-monotonic reasoning (McCarthy 1980). The key intuition behind minimal entailment is the notion of minimal models, providing as few exceptions as possible. Apart from its foundational relation to human reasoning, minimal entailment has widespread applications, e.g. in AI, description logics (Bonatti, Lutz, and Wolter 2009; Grimm and Hitzler 2009; Giordano et al. 2013) and SAT solving (Janota and Marques-Silva 2011).

While the complexity of non-monotonic logics has been thoroughly studied — cf. e.g. the recent papers (Durand, Hermann, and Nordh 2012; Thomas 2012; Bonatti, Lutz, and Wolter 2009) or the survey (Thomas and Vollmer 2010) — considerably less is known about the complexity of theorem proving in these logics. This is despite the fact that a number of quite different formalisms have been introduced for circumscription and minimal entailment (Olivetti 1992; Niemelä 1996; Bonatti and Olivetti 2002; Grimm and Hitzler 2009; Giordano et al. 2013). While proof complexity has traditionally focused on proof systems for classical propositional logic, there has been remarkable interest in proof complexity of non-classical logics during the last decade. A number of exciting results have been obtained — in particular for modal and intuitionistic logics (Hrubeš 2009; Jeřábek 2009) — and interesting phenomena have been observed that show a quite different picture from classical proof complexity, cf. (Beyersdorff and Kutz 2012) for a survey.

In this paper we focus our attention at two very different formalisms for minimal entailment: a sequent calculus MLK and a tableau system $OTAB$, both developed by Olivetti (1992).[1] These systems are very natural and elegant, and in fact they were both inspired by their classical propositional counterparts: Gentzen's LK (1935) and Smullyan's analytic tableau (1968).

Our main contribution is to show a p-simulation of $OTAB$ by MLK, *i.e.*, proofs in $OTAB$ can be efficiently transformed into MLK-derivations. This answers an open question by Olivetti (1992) on the relationship between these two calculi. At first sight, our result might not appear unexpected as sequent calculi are usually stronger than tableau systems, cf. e.g. (Urquhart 1995). However, the situation is more complicated here, and even Olivetti himself did not seem to have a clear conjecture as to whether such a simulation should be expected, cf. the remark after Theorem 8 in (Olivetti 1992).

The reason for the complication lies in the nature of the tableau: while rules in MLK are 'local', *i.e.*, they refer to only two previous sequents in the proof, the conditions to close branches in $OTAB$ are 'global' as they refer to other branches in the tableau, and this reference is even recursive. The trick we use to overcome this difficulty is to annotate nodes in the tableau with additional information that 'localises' the global information. This annotation is possible in polynomial time. The annotated nodes are then translated into minimal entailment sequents that form the skeleton of the MLK derivation for the p-simulation.

In addition to the p-simulation of $OTAB$ by MLK, we obtain an exponential separation between the two

[*]Supported by a grant from the John Templeton Foundation.

[†]Supported by a Doctoral Training Grant from EPSRC.

[1]While the name MLK is Olivetti's original notation (Olivetti 1992), we introduce the name $OTAB$ here as shorthand for Olivetti's tableau. By $NTAB$ we denote another tableau for minimal entailment suggested by Niemelä (1996), cf. the conclusion of this paper.

systems, *i.e.*, there are formulas which have polynomial-size proofs in *MLK*, but require exponential-size *OTAB* tableaux. In proof complexity, lower bounds and separations are usually much harder to show than simulations, and indeed there are famous examples where simulations have been known for a long time, but separations are currently out of reach, cf. (Krajíček 1995). In contrast, the situation is opposite here: while the separation carries over rather straightforwardly from the comparison between classical tableau and *LK*, the proof of the simulation result is technically very involved.

This paper is organised as follows. We start by recalling basic definitions from minimal entailment and proof complexity, and explaining Olivetti's systems *MLK* and *OTAB* for minimal entailment (Olivetti 1992). This is followed by two sections containing the p-simulation and the separation of *OTAB* and *MLK*. In the last section, we conclude by placing our results into the global picture of proof complexity research on circumscription and non-monotonic logics.

Preliminaries

Our propositional language contains the logical symbols $\bot, \top, \neg, \vee, \wedge, \rightarrow$. For a set of formulae Σ, $\mathrm{VAR}(\Sigma)$ is the set of all atoms that occur in Σ. For a set P of atoms we set $\neg P = \{\neg p \mid p \in P\}$. Disjoint union of two sets A and B is denoted by $A \sqcup B$.

Minimal Entailment. Minimal entailment is a form of non-monotonic reasoning developed as a special case of McCarthy's circumscription (McCarthy 1980). Minimal entailment comes both in a propositional and a first-order variant. Here we consider only the version of minimal entailment for propositional logic. We identify models with sets of positive atoms and use the partial ordering $\subseteq$ based on inclusion. This gives rise to a natural notion of minimal model for a set of formulae, in which the number of positive atoms is minimised with respect to inclusion. For a set of propositional formulae Γ we say that Γ minimally entails a formula ϕ if all minimal models of Γ also satisfies ϕ. We denote this entailment by $\Gamma \models_M \phi$.

Proof Complexity. A *proof system* (Cook and Reckhow 1979) for a language L over alphabet Γ is a polynomial-time computable partial function $f : \Gamma^\star \rightharpoonup \Gamma^\star$ with $rng(f) = L$. An *f-proof* of string y is a string x such that $f(x) = y$.

Proof systems are compared by simulations. We say that a proof system f *simulates* g ($g \leq f$) if there exists a polynomial p such that for every g-proof π_g there is an f-proof π_f with $f(\pi_f) = g(\pi_g)$ and $|\pi_f| \leq p(|\pi_g|)$. If π_f can even be constructed from π_g in polynomial time, then we say that f *p-simulates* g ($g \leq_p f$). Two proof systems f and g are *(p-)equivalent* ($g \equiv_{(p)} f$) if they mutually (p-)simulate each other.

The sequent calculus of *Gentzen's system LK* is one of the historically first and best studied proof systems (Gentzen 1935). In *LK* a sequent is usually written in the form $\Gamma \vdash \Delta$. Formally, a *sequent* is a pair (Γ,Δ) with Γ and Δ finite sets of formulae. In classical logic $\Gamma \vdash \Delta$ is true if every model for $\bigwedge \Gamma$ is also a model of $\bigvee \Delta$, where the disjunction of the empty set is taken as $\bot$ and the conjunction as $\top$. The system can be used both for propositional and first-order logic; the propositional rules are displayed in Fig. 1. Notice that the rules here do not contain structural rules for contraction or exchange. These come for free as we chose to operate with sets of formulae rather than sequences. Note the soundness of rule $(\bullet \vdash)$, which gives us monotonicity of classical propositional logic.

$$\frac{}{A \vdash A}\,(\vdash) \qquad \frac{}{\bot \vdash}\,(\bot \vdash) \qquad \frac{}{\vdash \top}\,(\vdash \top)$$

$$\frac{\Gamma \vdash \Sigma}{\Delta, \Gamma \vdash \Sigma}\,(\bullet \vdash) \qquad \frac{\Gamma \vdash \Sigma}{\Gamma \vdash \Sigma, \Delta}\,(\vdash \bullet)$$

$$\frac{\Gamma \vdash \Sigma, A}{\neg A, \Gamma \vdash \Sigma}\,(\neg \vdash) \qquad \frac{A, \Gamma \vdash \Sigma}{\Gamma \vdash \Sigma, \neg A}\,(\vdash \neg)$$

$$\frac{A, \Gamma \vdash \Sigma}{B \wedge A, \Gamma \vdash \Sigma}\,(\bullet\wedge \vdash) \qquad \frac{A, \Gamma \vdash \Sigma}{A \wedge B, \Gamma \vdash \Sigma}\,(\wedge\bullet \vdash)$$

$$\frac{\Gamma \vdash \Sigma, A \qquad \Gamma \vdash \Sigma, B}{\Gamma \vdash \Sigma, A \wedge B}\,(\vdash \wedge)$$

$$\frac{A, \Gamma \vdash \Sigma \qquad B, \Gamma \vdash \Sigma}{A \vee B, \Gamma \vdash \Sigma}\,(\vee \vdash)$$

$$\frac{\Gamma \vdash \Sigma, A}{\Gamma \vdash \Sigma, B \vee A}\,(\vdash \bullet\vee) \qquad \frac{\Gamma \vdash \Sigma, A}{\Gamma \vdash \Sigma, A \vee B}\,(\vdash \vee\bullet)$$

$$\frac{A, \Gamma \vdash \Sigma, B}{\Gamma \vdash \Sigma, A \rightarrow B}\,(\vdash\rightarrow)$$

$$\frac{\Gamma \vdash \Sigma, A \qquad B, \Delta \vdash \Lambda}{A \rightarrow B, \Gamma, \Delta \vdash \Sigma, \Lambda}\,(\rightarrow\vdash)$$

$$\frac{\Gamma \vdash \Sigma, A \qquad A, \Gamma \vdash \Sigma}{\Gamma \vdash \Sigma}\,(\mathrm{cut})$$

Figure 1: Rules of the sequent calculus *LK* (Gentzen 1935)

Olivetti's sequent calculus and tableau system for minimal entailment

In this section we review two proof systems for minimal entailment, which were developed by Olivetti (1992). We start with the sequent calculus *MLK*. Semantically, a minimal entailment sequent $\Gamma \vdash_M \Delta$ is true if and only if in all minimals models of $\bigwedge \Gamma$ the formula $\bigvee \Delta$ is satisfied. In addition to all axioms and rules from *LK*, the calculus *MLK* comprises the axioms and rules detailed in Figure 2. In the *MLK* axiom, the notion of a *positive* atom p in a formula ϕ is defined inductively by counting the number of negations and implications in ϕ

on top of p (cf. (Olivetti 1992) for the precise definition).

$$\frac{}{\Gamma \vdash_M \neg p}\ (\vdash_M)$$

where p is an atom that does not occur positively in any formula in Γ

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash_M \Delta}\ (\vdash\vdash_M)$$

$$\frac{\Gamma \vdash_M \Sigma, A \qquad A, \Gamma \vdash_M \Lambda}{\Gamma \vdash_M \Sigma, \Lambda}\ (\text{M-cut})$$

$$\frac{\Gamma \vdash_M \Sigma \qquad \Gamma \vdash_M \Delta}{\Gamma, \Sigma \vdash_M \Delta}\ (\bullet \vdash_M)$$

$$\frac{\Gamma \vdash_M \Sigma, A \qquad \Gamma \vdash_M \Sigma, B}{\Gamma \vdash_M \Sigma, A \wedge B}\ (\vdash_M \wedge)$$

$$\frac{A, \Gamma \vdash_M \Sigma \qquad B, \Gamma \vdash_M \Sigma}{A \vee B, \Gamma \vdash_M \Sigma}\ (\vee \vdash_M)$$

$$\frac{\Gamma \vdash_M \Sigma, A}{\Gamma \vdash_M \Sigma, B \vee A}\ (\vdash_M \bullet\vee)$$

$$\frac{\Gamma \vdash_M \Sigma, A}{\Gamma \vdash_M \Sigma, A \vee B}\ (\vdash_M \vee\bullet)$$

$$\frac{A, \Gamma \vdash_M \Sigma}{\Gamma \vdash_M \Sigma, \neg A}\ (\vdash_M \neg)$$

$$\frac{A, \Gamma \vdash_M \Sigma, B}{\Gamma \vdash_M \Sigma, A \to B}\ (\vdash_M \to)$$

Figure 2: Rules of the sequent calculus *MLK* for minimal entailment (Olivetti 1992)

Theorem 1 ***(Theorem 8 in (Olivetti 1992))*** *A sequent $\Gamma \vdash_M \Delta$ is true iff it is derivable in MLK.*

In addition to the sequent calculus *MLK*, Olivetti developed a tableau calculus for minimal entailment (Olivetti 1992). Here we will refer to this calculus as *OTAB*. A tableau is a rooted tree where nodes are labelled with formulae. In *OTAB*, the nodes are labelled with formulae that are signed with the symbol T or F. The combination of the sign and the top-most connective allows us to classify signed formulas into α or β-type formulae as detailed in Figure 3. Intuitively, for an α-type formula, a branch in the tableau is augmented by α_1, α_2, whereas for a β-type formula it splits according to β_1, β_2. Nodes in the tableau can be either marked or unmarked. For a sequent $\Gamma \vdash_M \Delta$, an *OTAB* tableau is constructed by the following process. We start from an initial tableau consisting of a single branch of unmarked formulae, which are exactly all formulae $\gamma \in \Gamma$, signed as $T\gamma$, and all formulae $\delta \in \Delta$, signed as $F\delta$. For a tableau and a branch $\mathcal{B}$ in this tableau we can extend the tableau by two rules:

α	α_1	α_2
$T(A \wedge B)$	TA	TB
$F\neg(A \wedge B)$	$F\neg A$	$F\neg B$
$T\neg(A \vee B)$	$T\neg A$	$T\neg B$
$F(A \vee B)$	FA	FB
$T\neg(A \to B)$	TA	$T\neg B$
$F(A \to B)$	$F\neg A$	FB
$T\neg\neg A$	TA	TA
$F\neg\neg A$	FA	FA

β	β_1	β_2
$T(A \vee B)$	TA	TB
$F\neg(A \vee B)$	$F\neg A$	$F\neg B$
$T\neg(A \wedge B)$	$T\neg A$	$T\neg B$
$F(A \wedge B)$	FA	FB
$T(A \to B)$	$T\neg A$	TB
$F\neg(A \to B)$	FA	$F\neg B$

Figure 3: Classification of signed formulae into α and β-type by sign and top-most connective

(A) If formula ϕ is an unmarked node in $\mathcal{B}$ of type α, then mark ϕ and add the two unmarked nodes α_1 and α_2 to the branch.

(B) If formula ϕ is an unmarked node in $\mathcal{B}$ of type β, then mark ϕ and split $\mathcal{B}$ into two branches $\mathcal{B}_1, \mathcal{B}_2$ with unmarked $\beta_1 \in \mathcal{B}_1$ and unmarked $\beta_2 \in \mathcal{B}_2$.

A branch $\mathcal{B}$ is *completed* if and only if all unmarked formulae on the branch are literals. A branch $\mathcal{B}$ is *closed* if and only if it satisfies at least one of the following conditions:

1. For some formula A, TA and $T\neg A$ are nodes of $\mathcal{B}$ (T-closed).
2. For some formula A, FA and $F\neg A$ are nodes of $\mathcal{B}$ (F-closed).
3. For some formula A, TA and FA are nodes of $\mathcal{B}$ (TF-closed).

For branch $\mathcal{B}$ let $\mathrm{At}(\mathcal{B}) = \{p : p \text{ is an atom and } Tp \text{ is a node in } \mathcal{B}\}$. We define two types of *ignorable branches:*

1. $\mathcal{B}$ is an *ignorable type-1 branch* if $\mathcal{B}$ is completed and there is an atom a such that $F\neg a$ is a node in $\mathcal{B}$, but Ta does not appear in $\mathcal{B}$.
2. $\mathcal{B}$ is an *ignorable type-2 branch* if there is another branch $\mathcal{B}'$ in the tableau that is completed but not T-closed, such that $\mathrm{At}(\mathcal{B}') \subset \mathrm{At}(\mathcal{B})$.

Theorem 2 ***(Theorem 2 in (Olivetti 1992))*** *The sequent $\Gamma \vdash_M \Delta$ is true if and only if there is an OTAB tableau in which every branch is closed or ignorable.*

Simulating *OTAB* by *MLK*

We will work towards a simulation of the tableau system *OTAB* by the sequent system *MLK*. In preparation for this a few lemmas are needed. We also add more information to the nodes (this can all be done in polynomial time). We start with a fact about *LK* (for a proof see (Beyersdorff and Chew 2014)).

Lemma 3 *For sets of formulae Γ, Δ and disjoints sets of atoms Σ^+, Σ^- with $\mathrm{VAR}(\Gamma \cup \Delta) = \Sigma^+ \sqcup \Sigma^-$ we can efficiently construct polynomial-size LK-proofs of $\Sigma^+, \neg\Sigma^-, \Gamma \vdash \Delta$ when the sequent is true.*

We also need to derive a way of weakening in *MLK*, and we show this in the next lemma.

Lemma 4 *From a sequent $\Gamma \vdash_M \Delta$ with non-empty Δ we can derive $\Gamma \vdash_M \Delta, \Sigma$ in a polynomial-size MLK-proof for any set of formulae Σ.*

Proof. We take $\delta \in \Delta$, and from the LK-axiom we get $\delta \vdash \delta$. From weakening in LK we obtain $\Gamma, \delta \vdash \Delta, \Sigma$. Using rule $(\vdash\vdash_M)$ we obtain $\Gamma, \delta \vdash_M \Delta, \Sigma$. We then derive $\Gamma \vdash_M \Delta, \Sigma$ using the (M-cut) rule. □

The proof makes essential use of the (M-cut) rule. As a result MLK is not complete without (M-cut); e.g. the sequent $\emptyset \vdash_M \neg a, \neg b$ cannot be derived. A discussion on cut elimination in MLK is given in (Olivetti 1992).

Lemma 5 *Let $T\tau$ be an α-type formula with $\alpha_1 = T\tau_1$, $\alpha_2 = T\tau_2$, and let $F\psi$ be an α-type formula with $\alpha_1 = F\psi_1$, $\alpha_2 = F\psi_2$. Similarly, let $T\phi$ be a β-type formula with $\beta_1 = T\phi_1$, $\beta_2 = T\phi_2$, and let $F\chi$ be an β-type formula with $\beta_1 = F\chi_1$, $\beta_2 = F\chi_2$.*

The following sequents all can be proved with polynomial-size LK-proofs: $\tau \vdash \tau_1 \wedge \tau_2$, $\tau_1 \wedge \tau_2 \vdash \tau$, $\psi \vdash \psi_1 \vee \psi_2$, $\psi_1 \vee \psi_2 \vdash \psi$, $\phi \vdash \phi_1 \vee \phi_2$, $\phi_1 \vee \phi_2 \vdash \phi$, $\chi \vdash \chi_1 \wedge \chi_2$, and $\chi_1 \wedge \chi_2 \vdash \chi$.

The straightforward proof of this involves checking all cases, which we omit here.

We now annotate the nodes u in an $OTAB$ tableau with three sets of formulae A_u, B_u, C_u and a set of branches D_u. This information will later be used to construct sequents $A_u \vdash_M B_u, C_u$, which will form the skeleton of the eventual MLK proof that simulates the $OTAB$ tableau. Intuitively, if we imagine following a branch when constructing the tableau, A_u corresponds to the current unmarked T-formulae on the branch, while B_u corresponds to the current unmarked F-formulae. C_u contains global information on all the branches that minimise the ignorable type-2 branches in the subtree with root u. The formal definition follows. We start with the definition of the formulae A_u and B_u, which proceeds by induction on the construction of the tableau.

Definition 6 *Nodes u in the OTAB tableau from the initial tableau are annotated with $A_u = \Gamma$ and $B_u = \Delta$.*

For the inductive step, consider the case that the extension rule (A) was used on node u for the α-type signed formula ϕ. If $\phi = T\chi$ has $\alpha_1 = T\chi_1$, $\alpha_2 = T\chi_2$ then for the node v labelled α_1 and the node w labelled α_2, $A_v = A_w = (\{\chi_1, \chi_2\} \cup A_u) \setminus \{\chi\}$ and $B_u = B_v = B_w$. If $\phi = F\chi$ has $\alpha_1 = F\chi_1$, $\alpha_2 = F\chi_2$ then for the node v labelled α_1 and the node w labelled α_2, $A_u = A_v = A_w$ and $B_v = B_w = (\{\chi_1, \chi_2\} \cup B_u) \setminus \{\chi\}$.

Consider now the case that the branching rule (B) was used on node u for the β-type signed formula ϕ. If $\phi = T\chi$ has $\beta_1 = T\chi_1$, $\beta_2 = T\chi_2$ then for the node v labelled β_1 and the node w labelled β_2, $A_v = (\{\chi_1\} \cup A_u) \setminus \{\chi\}$, $A_w = (\{\chi_2\} \cup A_u) \setminus \{\chi\}$ and $B_v = B_w = B_u$. If $\phi = F\chi$ has $\beta_1 = F\chi_1$, $\beta_2 = F\chi_2$ then for the node v labelled β_1 and the node w labelled β_2, $B_v = (\{\chi_1\} \cup B_u) \setminus \{\chi\}$, $B_w = (\{\chi_2\} \cup B_u) \setminus \{\chi\}$ and $A_v = A_w = A_u$.

For each ignorable type-2 branch $\mathcal{B}$ we can find another branch $\mathcal{B}'$, which is not ignorable type-2 and such that $\mathrm{At}(\mathcal{B}') \subset \mathrm{At}(\mathcal{B})$. The definition of ignorable type-2 might just refer to another ignorable type-2 branch, but eventually — since the tableau is finite — we reach a branch $\mathcal{B}'$, which is not ignorable type-2. There could be several such branches, and we will denote the left-most such branch $\mathcal{B}'$ as $\theta(\mathcal{B})$.

We are now going to construct sets C_u and D_u. The set D_u contains some information on type-2 ignorable branches. Let u be a node, which is the root of a subtableau T, and consider the set I of all type-2 ignorable branches that go through T. Now intuitively, D_u is defined as the set of all branches from $\theta(I)$ that are outside of T. The set C_u is then defined from D_u as $C_u = \{\bigwedge_{p \in \mathrm{At}(\theta(\mathcal{B}))} p \mid \mathcal{B} \in D_u\}$. The formal constructions of C_u and D_u are below. Unlike A_u and B_u, which are constructed inductively from the root of the tableau, the sets C_u and D_u are constructed inductively from the leaves to the root, by reversing the branching procedure.

Definition 7 *For an ignorable type-2 branch $\mathcal{B}$ the end node u is annotated by the singleton sets $C_u = \{\bigwedge_{p \in \mathrm{At}(\theta(\mathcal{B}))} p\}$ and $D_u = \{\theta(\mathcal{B})\}$; for other leaves $C_u = D_u = \emptyset$.*

Inductively, we define:

- *For a node u with only one child v, we set $D_u = D_v$ and $C_u = C_v$.*
- *For a node u with two children v and w, we set $D_u = (D_v \setminus \{\mathcal{B} \mid w \in \mathcal{B}\}) \cup (D_w \setminus \{\mathcal{B} \mid v \in \mathcal{B}\})$ and $C_u = \{\bigwedge_{p \in \mathrm{At}(\theta(\mathcal{B}))} p \mid \mathcal{B} \in D_u\}$.*

For each binary node u with children v, w we specify two extra sets. We set $E_u = (D_v \cup D_w) \setminus D_u$, and from this we can construct the set of formulae $F_u = \{\bigwedge_{p \in \mathrm{At}(\mathcal{B})} p \mid \mathcal{B} \in E_u\}$. We let $\omega = \bigvee F_u$.

We now prepare the simulation result with a couple of lemmas.

Lemma 8 *Let $\mathcal{B}$ be a branch in an OTAB tableau ending in leaf u. Then A_u is the set of all unmarked T-formulae on $\mathcal{B}$ (with the sign T removed). Likewise B_u is the set of all unmarked F-formulae on $\mathcal{B}$ (with the sign F removed).*

Proof. We will verify this for T-formulae, the argument is the same for F-formulae. If $T\phi$ at node v is an unmarked formula on branch $\mathcal{B}$ then ϕ has been added to A_v, regardless of which extension rule is used and cannot be removed at any node unless it is marked. Therefore, if u is the leaf of the branch, we have $\phi \in A_u$. If $T\phi$ is marked then it is removed (in the inductive step in the construction in Definition 6) and is not present in A_u. F-formulae do not appear in A_u. □

Lemma 9 *Let $\mathcal{B}$ be a branch in an OTAB tableau.*

1. *Assume that $T\phi$ appears on the branch $\mathcal{B}$, and let $A(\mathcal{B})$ be the set of unmarked T-formulae on $\mathcal{B}$ (with the sign T removed). Then $A(\mathcal{B}) \vdash \phi$ can be derived in a polynomial-size LK-proof.*

2. *Assume that $F(\phi)$ appears on the branch $\mathcal{B}$, and let $B(\mathcal{B})$ be the set of unmarked F-formulae on $\mathcal{B}$ (with the sign F removed). Then $\phi \vdash B(\mathcal{B})$ can be derived in a polynomial-size LK-proof.*

Proof. We prove the two claims by induction on the number of branching rules (A) and extension rules (B) that have been applied on the path to the node. We start with the proof of the first item.

Induction Hypothesis (on the number of applications of rules (A) and (B) on the node labelled $T\phi$): For a node labelled $T\phi$ on branch $\mathcal{B}$, we can derive $A(\mathcal{B}) \vdash \phi$ in a polynomial-size LK-proof (in the size of the tableau).

Base Case ($T\phi$ is unmarked): The LK axiom $\phi \vdash \phi$ can be used and then weakening to obtain $A(\mathcal{B}) \vdash \phi$.

Inductive Step: If $T\phi$ is a marked α-type formula, then both $\alpha_1 = T\phi_1$ and $\alpha_2 = T\phi_2$ appear on the branch. By the induction hypothesis we derive $A(\mathcal{B}) \vdash \phi_1$, $A(\mathcal{B}) \vdash \phi_2$ in polynomial-size proofs, hence we can derive $A(\mathcal{B}) \vdash \phi_1 \wedge \phi_2$ in a polynomial-size proof (we are bounded in total number of proof subtrees by the numbers of nodes in our branch). We then have $\phi_1 \wedge \phi_2 \vdash \phi$ using Lemma 5. Using the cut rule we can derive $A(\mathcal{B}) \vdash \phi$.

If $T\phi$ is a β-type formula and is marked, then the branch must contain $\beta_1 = T\phi_1$ or $\beta_2 = T\phi_2$. Without loss of generality we can assume that $\beta_1 = T\phi_1$ appears on the branch. By the induction hypothesis $A(\mathcal{B}) \vdash \phi_1$, therefore we can derive $A(\mathcal{B}) \vdash \phi_1 \vee \phi_2$ since it is a β-type formula and derive $\phi_1 \vee \phi_2 \vdash \phi$ with Lemma 5. Then using the cut rule we derive $A(\mathcal{B}) \vdash \phi$.

The second item is again shown by induction.

Induction Hypothesis (on the number of applications of rules (A) and (B) on the node labelled $F\phi$): For a node labelled $F\phi$ on branch $\mathcal{B}$, we can derive $\phi \vdash B(\mathcal{B})$ in a polynomial-size LK-proof (in the size of the tableau).

Base Case ($F\phi$ is unmarked): The LK axiom $\phi \vdash \phi$ can be used and then weakened to $\phi \vdash B(\mathcal{B})$.

Inductive Step: If $F\phi$ is a marked α-type formula, then both $\alpha_1 = F\phi_1$ and $\alpha_2 = F\phi_2$ appear on the branch. Since by the inductive hypothesis $\phi_1 \vdash B(\mathcal{B})$ and $\phi_2 \vdash B(\mathcal{B})$, we can derive $\phi_1 \vee \phi_2 \vdash B(\mathcal{B})$ in a polynomial-size proof. We then have $\phi \vdash \phi_1 \vee \phi_2$ using Lemma 5. Using the cut rule we can derive $\phi \vdash B(\mathcal{B})$.

If $F\phi$ is a β-type formula and is marked, then the branch must contain $\beta_1 = F\phi_1$ or $\beta_2 = F\phi_2$. Without loss of generality we can assume $\beta_1 = F\phi_1$ appears on the branch. By the induction hypothesis $\phi_1 \vdash B(\mathcal{B})$, therefore we can derive $\phi_1 \wedge \phi_2 \vdash B(\mathcal{B})$ since it is a β-type formula and derive $\phi \vdash \phi_1 \wedge \phi_2$ with Lemma 5. Using the cut rule we derive $\phi \vdash B(\mathcal{B})$. □

Lemma 10 *Let $\mathcal{B}$ be a branch, which is completed but not T-closed. For any node u on $\mathcal{B}$, the model* $\mathrm{At}(\mathcal{B})$ *satisfies A_u.*

Proof. We prove the lemma by induction on the height of the subtree with root u.

Base Case (u is a leaf): By Lemma 8, A_u is the set of all unmarked T-formulae on $\mathcal{B}$. But these are all literals as $\mathcal{B}$ is completed, and hence the subset of positive atoms is equal to $\mathrm{At}(\mathcal{B})$.

Inductive step: If u is of extension type (A) with child node v then the models of A_u are exactly the same as the models of A_v. This is true for all α-type formulae. For example, if the extension process (A) was used on formula $T(\psi \wedge \chi)$ and the node v was labelled $T\psi$ then $A_v = \{\psi, \chi\} \cup A_u \setminus \{\psi \wedge \chi\}$ and this has the same models as A_u. By the induction hypothesis, $\mathrm{At}(\mathcal{B}) \models A_v$ and hence $\mathrm{At}(\mathcal{B}) \models A_u$.

If u is of branch type (B) with children v and w then $\mathrm{At}(\mathcal{B}) \models A_v$ and $\mathrm{At}(\mathcal{B}) \models A_w$. The argument works similarly for all β-type formulae; for example, if the extension process was using formula $T(\psi \vee \chi)$ and v is labelled $T\psi$ and w is labelled $T\chi$, then $A_u = (\{\psi \vee \chi\} \cup A_v) \setminus \{\psi\}$. Hence $\mathrm{At}(\mathcal{B}) \models A_v$ implies $\mathrm{At}(\mathcal{B}) \models A_u$. □

We now approach the simulation result (Theorem 13) and start to construct MLK proofs. For the next two lemmas, we fix an $OTAB$ tableau of size k and use the notation from Definitions 6 and 7 (recall in particular the definition of ω at the end of Definition 7).

Lemma 11 *There is a polynomial q such that for every binary node u, every proper subset $A' \subset A_u$ and every $\gamma \in A_u \setminus A'$ we can construct an MLK-proof of $A', \omega \vdash_M \gamma$ of size at most $q(k)$.*

Proof. **Induction Hypothesis** (on the number of formulae of A_u used in the antecedent: $|A'|$): We can find a $q(k)$-size MLK proof containing all sequents $A', \omega \vdash_M \gamma$ for every $\gamma \in A_u \setminus A'$.

Base Case (when A' is empty): For the base case we aim to prove $\omega \vdash_M \gamma$, and repeat this for every γ. We use two ingredients. Firstly, we need the sequent $\omega \vdash_M F_u, \gamma$ which is easy to prove using weakening and $(\vee \vdash)$, since ω is a disjunction of the elements in F_u. Our second ingredient is a scheme of $\omega, \bigwedge_{p \in M} p \vdash_M \gamma$ for all the $\bigwedge_{p \in M} p$ in F_u, *i.e.*, $M = \mathrm{At}(\mathcal{B})$ for some $\mathcal{B} \in E_u$. With these we can repeatedly use (M-cut) on the first sequent for every element in F_u. We now show how to efficiently prove the sequents of the form $\omega, \bigwedge_{p \in M} p \vdash_M \gamma$.

For branch $\mathcal{B} \in E_u$, as $\mathrm{At}(\mathcal{B})$ is a model M for A_u by Lemma 10, $M \models \gamma$. Since no atom a in $\mathrm{VAR}(\gamma) \setminus M$ appears positive in the set M we can infer $M \vdash_M \neg a$ directly via $(\vdash_M)$. With rule $(\vdash_M \wedge)$ we can derive $\bigwedge_{p \in M} p \vdash_M \bigwedge_{p \in \mathrm{VAR}(\gamma) \setminus M} \neg p$ in a polynomial-size proof. Using $(\vdash)$, $(\vdash \vee\bullet)$, and $(\vdash \bullet\vee)$ we can derive $\bigwedge_{p \in M} p \vdash \omega$. We then use these sequents in the proof below, denoting $\bigwedge_{p \in \mathrm{VAR}(\gamma) \setminus M} \neg p$ as $n(M)$:

$$\dfrac{\dfrac{\bigwedge_{p \in M} p \vdash \omega}{\bigwedge_{p \in M} p \vdash_M \omega}\,(\vdash\vdash_M) \qquad \bigwedge_{p \in M} p \vdash_M n(M)}{\omega, \bigwedge_{p \in M} p \vdash_M n(M)}\,(\bullet \vdash_M)$$

From Lemma 3, $M, \neg\mathrm{VAR}(\gamma) \setminus M \vdash \gamma$ can be derived in a polynomial-size proof. We use simple syntactic manipulation to change the antecedent into an equivalent conjunction and then weaken to derive $\omega, \bigwedge_{p\in M} p, \bigwedge_{p\in \mathrm{VAR}(\gamma)\setminus M} \neg p \vdash_M \gamma$ in a polynomial-size proof. Then we use:

$$\frac{\omega, \bigwedge_{p\in M} p, n(M) \vdash_M \gamma \qquad \omega, \bigwedge_{p\in M} p \vdash_M n(M)}{\omega, \bigwedge_{p\in M} p \vdash_M \gamma}\ (\text{M-cut})$$

Inductive Step: We look at proving $A', \gamma', \omega \vdash_M \gamma$, for every other $\gamma \in A_u \setminus A'$. For each γ we use two instances of the inductive hypothesis: $A', \omega \vdash_M \gamma$ and $A', \omega \vdash_M \gamma'$.

$$\frac{A', \omega \vdash_M \gamma' \qquad A', \omega \vdash_M \gamma}{A', \gamma', \omega \vdash_M \gamma}\ (\bullet \vdash_M)$$

Since we repeat this for every γ we only add $|(A_u \setminus A') \setminus \{\gamma\}|$ many lines in each inductive step and retain a polynomial bound. □

The previous lemma was an essential preparation for our next Lemma 12, which in turn will be the crucial ingredient for the p-simulation in Theorem 13.

Lemma 12 *There is a polynomial q such for every binary node u there is an MLK-proof of $A_u, \omega \vdash B_u$ of size at most $q(k)$.*

Proof. **Induction Hypothesis** (on the number of formulae of A_u used in the antecedent: $|A'|$): Let $A' \subseteq A_u$. There is a fixed polynomial q such that $A', \omega \vdash B_u$ has an *MLK*-proof of size at most $q(|\omega|)$.

Base Case (when A' is empty): We approach this very similarly as in the previous lemma. Using weakening and $(\vee \vdash)$, the sequent $\omega \vdash_M F_u, B_u$ can be derived in a polynomial-size proof. By repeated use of the cut rule on sequents of the form $\omega, \bigwedge_{p\in \mathrm{At}(\mathcal{B})} p \vdash_M B_u$ for $\mathcal{B} \in E_u$ the sequent $\omega \vdash_M B_u$ is derived. Now we only need to show that we can efficiently obtain $\omega, \bigwedge_{p\in M} p \vdash_M B_u$.

Consider branch $\mathcal{B} \in E_u$. As $\mathrm{At}(\mathcal{B})$ is a minimal model M for Γ by Lemma 10, this model M must satisfy Δ and given the limitations of the branching processes of F-labelled formulae, B_u as well.

Similarly as in the base case of Lemma 11 we can derive $\bigwedge_{p\in M} p \vdash_M \bigwedge_{p\in \mathrm{VAR}(B_u)\setminus M} \neg p$ and $\bigwedge_{p\in M} p \vdash \omega$ in a polynomial-size proof. We then use these sequents in the proof below once again, denoting $\bigwedge_{p\in \mathrm{VAR}(\gamma)\setminus M} \neg p$ as $n(M)$.

$$\frac{\dfrac{\bigwedge_{p\in M} p \vdash \omega}{\bigwedge_{p\in M} p \vdash_M \omega}\ (\vdash\vdash_M) \qquad \bigwedge_{p\in M} p \vdash_M n(M)}{\omega, \bigwedge_{p\in M} p \vdash_M n(M)}\ (\bullet \vdash_M)$$

We can use M satisfying B_u to derive $\omega, \bigwedge_{p\in M} p, n(M) \vdash B_u$ in the same way as we derive $\omega, \bigwedge_{p\in M} p, \bigwedge_{p\in \mathrm{VAR}(\gamma)\setminus M} \neg p \vdash \gamma$ in Lemma 11.

$$\frac{\omega, \bigwedge_{p\in M} p, n(M) \vdash_M B_u \qquad \omega, \bigwedge_{p\in M} p \vdash_M n(M)}{\omega, \bigwedge_{p\in M} p \vdash_M B_u}\ (\text{M-cut})$$

Inductive Step: Assume that $A', \omega \vdash_M B_u$ has already been derived. Let $\gamma \in A_u \setminus A'$. We use Lemma 11 to get a short proof of $A', \omega \vdash_M \gamma$. One application of rule $(\bullet \vdash_M)$

$$\frac{A', \omega \vdash_M B_u \qquad A', \omega \vdash_M \gamma}{A', \gamma, \omega \vdash_M B_u}\ (\bullet \vdash_M)$$

finishes the proof. □

Theorem 13 *MLK p-simulates OTAB.*

Proof. **Induction Hypothesis** (on the height of the subtree with root u): For node u, we can derive $A_u \vdash_M B_u, C_u$ in *MLK* in polynomial size (in the full tableau).

Base Case (u is a leaf): If the branch is T-closed, then by Lemma 9, for some formula ϕ we can derive $A_u \vdash \phi$ and $A_u \vdash \neg\phi$. Hence $A_u \vdash \phi \wedge \neg\phi$ can be derived and with $\phi \wedge \neg\phi \vdash$ and the cut rule we can derive $A_u \vdash$ in a polynomial-size proof. By weakening and using $(\vdash\vdash_M)$ we can derive $A_u \vdash_M B_u$ in polynomial size as required.

If the branch is F-closed, then by Lemma 9, for some formula ϕ we can derive $\phi \vdash B_u$ and $\neg\phi \vdash B_u$. Hence $\phi \vee \neg\phi \vdash B_u$ can be derived and with $\vdash \phi \vee \neg\phi$ and the cut rule we can derive $\vdash B_u$ in a polynomial-size proof. By weakening and using $(\vdash\vdash_M)$ we can derive $A_u \vdash_M B_u$ in polynomial size.

If the branch is TF-closed, then by Lemma 9, for some formula ϕ we can derive $A_u \vdash \phi$ and $\phi \vdash B_u$. Hence via the cut rule and using $(\vdash\vdash_M)$ we can derive $A_u \vdash_M B_u$ in polynomial size as required.

If the branch is ignorable type-1 then the branch is completed. Therefore A_u is a set of atoms and there is some atom $a \notin A_u$ such that $\neg a \in B_u$. It therefore follows that $A_u \vdash_M \neg a$ can be derived as an axiom using the $(\vdash_M)$ rule. We then use Lemma 4 to derive $A_u \vdash_M B_u$ in polynomial size.

If the branch is ignorable type-2 then $p \in \mathrm{At}(\theta(\mathcal{B}))$ implies $p \in A_u$. Since $C_u = \{\bigwedge_{p\in \mathrm{At}(\theta(\mathcal{B}))} p\}$ we can find a short proof of $A_u \vdash C_u$ using $(\vdash \wedge)$.

Inductive Step: The inductive step splits into four cases according to which extension or branching rule is used on node u.

Case 1. Extension rule (A) is used on node u for formula $T\phi$ with resulting nodes v and w labelled $T\phi_1$, $T\phi_2$, respectively.

$$\frac{\dfrac{\overline{\phi_1 \vdash \phi_1}}{\phi_1, \phi_2 \vdash \phi_1}\ (\bullet \vdash) \qquad \dfrac{\overline{\phi_2 \vdash \phi_2}}{\phi_1, \phi_2 \vdash \phi_2}\ (\bullet \vdash)}{\phi_1, \phi_2 \vdash \phi_1 \wedge \phi_2}\ (\vdash \wedge)$$

Since we are extending the branch on an α-type formula signed with T, we can find a short proof of $\phi_1 \wedge \phi_2 \vdash \phi$ using Lemma 5. Together with $\phi_1, \phi_2 \vdash \phi_1 \wedge \phi_2$ shown above we derive:

$$\frac{\phi_1, \phi_2 \vdash \phi_1 \wedge \phi_2 \qquad \phi_1 \wedge \phi_2 \vdash \phi}{\phi_1, \phi_2 \vdash \phi}\ (\text{cut})$$

By definition we have $\phi_1, \phi_2 \in A_v$, and then by weakening $\phi_1, \phi_2 \vdash \phi$ we obtain $A_v \vdash \phi$. By Definitions 6 and 7, $B_v = B_u$ and likewise $C_u = C_v$. Hence $A_v \vdash_M B_u, C_u$ is available by the induction hypothesis. From this we get:

$$\dfrac{\dfrac{A_v \vdash \phi}{A_v \vdash_M \phi}\,(\vdash\vdash_M) \qquad A_v \vdash_M B_u, C_u}{A_v, \phi \vdash_M B_u, C_u}\,(\bullet \vdash_M)$$

$A_u \vdash \phi_1$ and $A_u \vdash \phi_2$ also have short proofs from weakening axioms. These can be used to cut out ϕ_1, ϕ_2 from the antecedent of $A_v, \phi \vdash_M B_u, C_u$ resulting in $A_u \vdash_M B_u, C_u$ as required.

Case 2. Extension rule (A) is used on node u for formula $F\phi$ with resulting nodes v and w labelled $F\phi_1$, $F\phi_2$, respectively. We can find short proofs of $A_u, \phi_1 \vdash \phi_1 \vee \phi_2$, $A_u, \phi_2 \vdash \phi_1 \vee \phi_2$ using axioms, weakening and the rules $(\vdash \bullet\vee)$, $(\vdash \vee\bullet)$. Similarly as in the last case, we have $A_v = A_u$ and likewise $C_u = C_v$. Therefore, by induction hypothesis $A_u \vdash_M B_v, C_u$ is available with a short proof.

$$\dfrac{A_u \vdash_M B_v, C_u \qquad \dfrac{A_u, \phi_1 \vdash \phi_1 \vee \phi_2}{A_u, \phi_1 \vdash_M \phi_1 \vee \phi_2}\,(\vdash\vdash_M)}{A_u \vdash_M B_v \setminus \{\phi_1\}, \phi_1 \vee \phi_2, C_u}\,(\text{M-cut})$$

We can do the same trick with ϕ_2:

$$\dfrac{A_u \vdash_M B_v \setminus \{\phi_1\}, \phi_1 \vee \phi_2, C_u \qquad \dfrac{A_u, \phi_2 \vdash \phi_1 \vee \phi_2}{A_u, \phi_2 \vdash_M \phi_1 \vee \phi_2}\,(\vdash\vdash_M)}{A_u \vdash_M B_u \setminus \{\phi\}, \phi_1 \vee \phi_2, C_u}\,(\text{M-cut})$$

Since $F\phi$ is an α-type formula, then $\phi_1 \vee \phi_2 \vdash \phi$ by Lemma 5, and by weakening $A_u, \phi_1 \vee \phi_2 \vdash \phi$. The derivation is the finished by:

$$\dfrac{A_u \vdash_M B_u \setminus \{\phi\}, \phi_1 \vee \phi_2, C_u \qquad \dfrac{A_u, \phi_1 \vee \phi_2 \vdash \phi}{A_u, \phi_1 \vee \phi_2 \vdash_M \phi}\,(\vdash\vdash_M)}{A_u \vdash_M B_u, C_u}\,(\text{M-cut})$$

Case 3. Branching rule (B) is used on node u for formula $T\phi$ with children v and w labelled $T\phi_1$, $T\phi_2$, respectively. The sequents $A_v \vdash_M B_u, C_v$ and $A_w \vdash_M B_u, C_w$ are available from the induction hypothesis.

$A_v \vdash_M B_u, C_u, F_u$ and $A_w \vdash_M B_u, C_u, F_u$ can be derived via weakening by Lemma 4. From these sequents, simple manipulation through classical logic and the cut rule gives us $A_v \vdash_M B_u, C_u, \omega$ and $A_w \vdash_M B_u, C_u, \omega$. Using the rule $(\vee \vdash_M)$ we obtain $A_u \setminus \{\phi\}, \phi_1 \vee \phi_2 \vdash_M B_u, C_u, \omega$. Since $\phi \in A_u$, from Lemma 5 we derive $\phi \vdash \phi_1 \vee \phi_2$ and $\phi_1 \vee \phi_2 \vdash \phi$ in polynomial size. Weakening derives $A_u \vdash \phi_1 \vee \phi_2$ and $A_u \setminus \{\phi\}, \phi_1 \vee \phi_2 \vdash \phi$. From these we derive:

$$\dfrac{\dfrac{A_u \setminus \{\phi\}, \phi_1 \vee \phi_2 \vdash \phi}{A_u \setminus \{\phi\}, \phi_1 \vee \phi_2 \vdash_M \phi}\,(\vdash\vdash_M) \qquad A_u \setminus \{\phi\}, \phi_1 \vee \phi_2 \vdash_M B_u, C_u, \omega}{A_u, \phi_1 \vee \phi_2 \vdash_M B_u, C_u, \omega}\,(\bullet \vdash_M)$$

$$\dfrac{A_u, \phi_1 \vee \phi_2 \vdash_M B_u, C_u, \omega \qquad \dfrac{A_u \vdash \phi_1 \vee \phi_2}{A_u \vdash_M \phi_1 \vee \phi_2}\,(\vdash\vdash_M)}{A_u \vdash_M B_u, C_u, \omega}\,(\text{M-cut})$$

From Lemma 12, $A_u, \omega \vdash_M B_u, C_u$ has a polynomial size proof. We can then finish the derivation with a cut:

$$\dfrac{A_u, \omega \vdash_M B_u \qquad A_u \vdash_M B_u, C_u, \omega}{A_u \vdash_M B_u, C_u}\,(\text{M-cut})$$

Case 4. Branching rule (B) is used on node u for formula $F\phi$ with children v and w labelled $F\phi_1$, $F\phi_2$, respectively. The sequents $A_u \vdash_M B_v, C_v$ and $A_u \vdash_M B_w, C_w$ are available from the induction hypothesis.

From these two sequents we obtain via weakening $A_u \vdash_M B_v, C_u, F_u$ and $A_u \vdash_M B_w, C_u, F_u$. We can turn F_u into the disjunction of its elements by simple manipulation through classical logic and the cut rule and derive $A_u \vdash_M B_v, C_u, \omega$ and $A_u \vdash_M B_w, C_u, \omega$. Using the rule $(\vdash_M \wedge)$ we obtain $A_u \vdash_M B_u \setminus \{\phi\}, \phi_1 \wedge \phi_2, C_u, \omega$. Since $\phi_1 \wedge \phi_2 \vdash \phi$ by Lemma 5, we derive by weakening $A_u, \phi_1 \wedge \phi_2 \vdash \phi$. We then continue:

$$\dfrac{A_u \vdash_M B_u \setminus \{\phi\}, \phi_1 \wedge \phi_2, C_u, \omega \qquad \dfrac{A_u, \phi_1 \wedge \phi_2 \vdash \phi}{A_u, \phi_1 \wedge \phi_2 \vdash_M \phi}\,(\vdash\vdash_M)}{A_u \vdash_M B_u, C_u, \omega}\,(\text{M-cut})$$

From Lemma 12, $A_u, \omega \vdash_M B_u, C_u$ has a polynomial-size proof.

$$\dfrac{A_u, \omega \vdash_M B_u \qquad A_u \vdash_M B_u, C_u, \omega}{A_u \vdash_M B_u, C_u}\,(\text{M-cut})$$

This completes the proof of the induction.

From this induction, the theorem can be derived as follows. The induction hypothesis applied to the root u of the tableau gives polynomial-size *MLK* proofs of $A_u \vdash_M B_u, C_u$. By definition $A_u = \Gamma$ and $B_u = \Delta$. Finally, $C_u = D_u = \emptyset$, because for every ignorable type-2 branch $\mathcal{B}$, the branch $\theta(\mathcal{B})$ is inside the tableau.

Since all our steps are constructive we prove a p-simulation. □

Separating *OTAB* and *MLK*

In the previous section we showed that *MLK* p-simulates *OTAB*. Here we prove that the two systems are in fact exponentially separated.

Lemma 14 *In every OTAB tableau for $\Gamma \vdash_M \Delta$ with inconsistent Γ, any completed branch is T-closed.*

Proof. If a branch $\mathcal{B}$ is completed but not T-closed, then via Lemma 10, $\mathrm{At}(\mathcal{B})$ is a model for all initial T-formulae. But these form an inconsistent set. □

Theorem 15 *OTAB does not simulate MLK.*

Proof. We consider Smullyan's *analytic tableaux* (Smullyan 1968), and use the hard sets of inconsistent formulae in (D'Agostino 1992).

For each natural number $n > 0$ we use variables $p_1, \ldots, p_n$. Let H_n be the set of all 2^n clauses of length n over these variables (we exclude tautological clauses) and define $\phi_n = \bigwedge H_n$. Since every model must contradict one of these clauses, ϕ_n is inconsistent. We now consider the sequents $\phi_n \vdash_M$.

Since classical entailment is included in minimal entailment there must also be an $OTAB$ tableau for these formulae. Every type-1 ignorable branch in the $OTAB$ tableau is completed and therefore also T-closed by Lemma 14. The tableau cannot contain any type-2 ignorable branches as every completed branch is T-closed. Hence the $OTAB$ tableaux for $\phi_n \vdash_M$ are in fact analytic tableaux and have $n!$ many branches by Proposition 1 from (D'Agostino 1992).

Since the examples are easy for truth tables (D'Agostino 1992), they are also easy for LK and the rule $(\vdash\vdash_M)$ completes a polynomial-size proof for them in MLK. □

Conclusion

In this paper we have clarified the relationship between the proof systems $OTAB$ and MLK for minimal entailment. While cut-free sequent calculi typically have the same proof complexity as tableau systems, MLK is not complete without M-cut (Olivetti 1992), and also our translation uses M-cut in an essential way (however, we can eliminate LK-cut).

We conclude by mentioning that there are further proof systems for minimal entailment and circumscription, which have been recently analysed from a proof-complexity perspective (Beyersdorff and Chew 2014). In particular, Niemelä (1996) introduced a tableau system $NTAB$ for minimal entailment for clausal formulas, and Bonatti and Olivetti (2002) defined an analytic sequent calculus $CIRC$ for circumscription. Building on initial results from (Bonatti and Olivetti 2002) we prove in (Beyersdorff and Chew 2014) that $NTAB \leq_p CIRC \leq_p MLK$ is a chain of proof systems of strictly increasing strength, *i.e.*, in addition to the p-simulations we obtain separations between the proof systems.

Combining the results of (Beyersdorff and Chew 2014) and the present paper, the full picture of the simulation order of proof systems for minimal entailment emerges. In terms of proof size, MLK is the best proof system as it p-simulates all other known proof systems. However, for a complete understanding of the simulation order some problems are still open. While the separation between $OTAB$ and MLK from Theorem 15 can be straightforwardly adapted to show that $OTAB$ also does not simulate $CIRC$, we leave open whether the reverse simulation holds. Likewise, the relationship between the two tableau systems $OTAB$ and $NTAB$ is not clear.

It is also interesting to compare our results to the complexity of theorem proving procedures in other non-monotonic logics as default logic (Beyersdorff et al. 2011) and autoepistemic logic (Beyersdorff 2013); cf. also (Egly and Tompits 2001) for results on proof complexity in the first-order versions of some of these systems. In particular, (Beyersdorff et al. 2011) and (Beyersdorff 2013) show very close connections between proof lengths in some sequent systems for default and autoepistemic logic and proof lengths of classical LK, for which any non-trivial lower bounds are a major outstanding problem. It would be interesting to know if a similar relation also holds between MLK and LK.

References

Beyersdorff, O., and Chew, L. 2014. The complexity of theorem proving in circumscription and minimal entailment. To appear in Proc. *IJCAR'14*. Available as Technical Report TR14-014, Electronic Colloquium on Computational Complexity.

Beyersdorff, O., and Kutz, O. 2012. Proof complexity of non-classical logics. In Bezhanishvili, N., and Goranko, V., eds., *Lectures on Logic and Computation - ESSLLI 2010/11, Selected Lecture Notes*. Springer, Berlin Heidelberg. 1–54.

Beyersdorff, O.; Meier, A.; Müller, S.; Thomas, M.; and Vollmer, H. 2011. Proof complexity of propositional default logic. *Archive for Mathematical Logic* 50(7):727–742.

Beyersdorff, O. 2013. The complexity of theorem proving in autoepistemic logic. In *SAT*, 365–376.

Bonatti, P. A., and Olivetti, N. 2002. Sequent calculi for propositional nonmonotonic logics. *ACM Transactions on Computational Logic* 3(2):226–278.

Bonatti, P. A.; Lutz, C.; and Wolter, F. 2009. The complexity of circumscription in DLs. *J. Artif. Intell. Res. (JAIR)* 35:717–773.

Cook, S. A., and Reckhow, R. A. 1979. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic* 44(1):36–50.

D'Agostino, M. 1992. Are tableaux an improvement on truth-tables? *Journal of Logic, Language and Information* 1(3):235–252.

Durand, A.; Hermann, M.; and Nordh, G. 2012. Trichotomies in the complexity of minimal inference. *Theory Comput. Syst.* 50(3):446–491.

Egly, U., and Tompits, H. 2001. Proof-complexity results for nonmonotonic reasoning. *ACM Transactions on Computational Logic* 2(3):340–387.

Gentzen, G. 1935. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* 39:68–131.

Giordano, L.; Gliozzi, V.; Olivetti, N.; and Pozzato, G. L. 2013. A non-monotonic description logic for reasoning about typicality. *Artif. Intell.* 195:165–202.

Grimm, S., and Hitzler, P. 2009. A preferential tableaux calculus for circumscriptive ALCO. In Polleres, A., and Swift, T., eds., *Proc. Web Reasoning and Rule Systems*, volume 5837 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 40–54.

Hrubeš, P. 2009. On lengths of proofs in non-classical logics. *Annals of Pure and Applied Logic* 157(2–3):194–205.

Janota, M., and Marques-Silva, J. 2011. cmMUS: A tool for circumscription-based MUS membership testing. In *LPNMR*, 266–271.

Jeřábek, E. 2009. Substitution Frege and extended Frege proof systems in non-classical logics. *Annals of Pure and Applied Logic* 159(1–2):1–48.

Krajíček, J. 1995. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge: Cambridge University Press.

McCarthy, J. 1980. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.

Niemelä, I. 1996. A tableau calculus for minimal model reasoning. In *TABLEAUX*, 278–294.

Olivetti, N. 1992. Tableaux and sequent calculus for minimal entailment. *J. Autom. Reasoning* 9(1):99–139.

Smullyan, R. 1968. *First Order Logic*. Berlin: Springer-Verlag.

Thomas, M., and Vollmer, H. 2010. Complexity of non-monotonic logics. *Bulletin of the EATCS* 102:53–82.

Thomas, M. 2012. The complexity of circumscriptive inference in Post's lattice. *Theory of Computing Systems* 50(3):401–419.

Urquhart, A. 1995. The complexity of propositional proofs. *Bulletin of Symbolic Logic* 1:425–467.

Revisiting Chase Termination for Existential Rules and their Extension to Nonmonotonic Negation

Jean-François Baget
INRIA

Fabien Garreau
University of Angers

Marie-Laure Mugnier
University of Montpellier

Swan Rocher
University of Montpellier

Abstract

Existential rules have been proposed for representing ontological knowledge, specifically in the context of Ontology-Based Data Access. Entailment with existential rules is undecidable. We focus in this paper on conditions that ensure the termination of a breadth-first forward chaining algorithm known as the chase. Several variants of the chase have been proposed. In the first part of this paper, we propose a new tool that allows to extend existing acyclicity conditions ensuring chase termination, while keeping good complexity properties. In the second part, we study the extension to existential rules with nonmonotonic negation under stable model semantics, discuss the relevancy of the chase variants for these rules and further extend acyclicity results obtained in the positive case.

Introduction

Existential rules (also called Datalog+/-) have been proposed for representing ontological knowledge, specifically in the context of Ontology-Based Data Access, a new paradigm in data management that aims to exploit ontological knowledge when accessing data (Calì, Gottlob, and Lukasiewicz 2009a; Baget et al. 2009). These rules allow to assert the existence of unknown individuals, a feature recognized as crucial for representing knowledge in an open domain perspective. Existential rules generalize lightweight description logics, such as DL-Lite and $\mathcal{EL}$ (Calvanese et al. 2007; Baader, Brandt, and Lutz 2005) and overcome some of their limitations by allowing any predicate arity as well as cyclic structures.

Entailment with existential rules is known to be undecidable ((Beeri and Vardi 1981) (Chandra, Lewis, and Makowsky 1981) on tuple-generating dependencies). Many sufficient conditions for decidability, obtained by syntactic restrictions on sets of rules, have been exhibited in knowledge representation and database theory (see e.g., the overview in (Mugnier 2011)). We focus in this paper on conditions that ensure the termination of a breadth-first forward chaining algorithm, known as the *chase* in the database literature. Given a knowledge base composed of data and existential rules, the chase saturates the data by application of the rules. When it is ensured to terminate, inferences enabled by the rules can be materialized in the data, which can then be queried like a classical database, thus allowing to benefit from database optimizations techniques implemented in current data management systems. Several variants of the chase have been proposed, which differ in the way they deal with redundant information (Fagin et al. 2005; Deutsch, Nash, and Remmel 2008; Marnette 2009). It follows that they do not behave in the same way with respect to termination. In the following, when we write *the* chase, we mean one of these variants. Various acyclicity notions have been proposed to ensure the halting of some chase variants.

Nonmonotonic extensions to existential rules were recently considered in (Calı, Gottlob, and Lukasiewicz 2009b) with stratified negation, (Gottlob et al. 2012) with well-founded semantics and (Magka, Krötzsch, and Horrocks 2013) with stable model semantics. This latter work studies skolemized existential rules (which can then be seen as specific logic programs) and focuses on cases where a finite unique model exists.

In this paper, we tackle the following issues : Can we still extend known acyclicity notions ? Would any chase variant be applicable to existential rules provided with nonmonotonic negation, a useful feature for ontological modeling ?

1. Extending acyclicity notions. Acyclicity conditions can be classified into two main families : the first one constrains the way existential variables are propagated during the chase (e.g. (Fagin et al. 2003; 2005; Marnette 2009; Krötzsch and Rudolph 2011)) and the second one encodes dependencies between rules, i.e., the fact that a rule may lead to trigger another rule (e.g. (Baget 2004; Deutsch, Nash, and Remmel 2008; Baget et al. 2011)). These conditions are based on different graphs, but all of them can be seen as can as forbidding "dangerous" cycles in the considered graph. We define a new family of graphs that allows to extend these acyclicity notions, while keeping good complexity properties.

2. Processing rules with nonmonotonic negation. We define a notion of stable models directly on nonmonotonic existential rules and provide a derivation algorithm inspired by Answer Set Programming solvers that instantiate rules "on the fly" (Lefèvre and Nicolas 2009; Dao-Tran et al. 2012). This algorithm is parametrized by a chase variant. We point out that, differently to the positive case, not all variants of the chase lead to sound procedures in presence of nonmonotonic negation ; furthermore, skolemizing existential variables or not makes a semantic difference, even when both computations terminate. Finally, we further extend acy-

clicity results obtained on positive rules by exploiting negative information as well.

A technical report with the proofs omitted for space restriction reasons is available `http://www2.lirmm.fr/~baget/publications/nmr2014-long.pdf`.

Preliminaries

Atomsets We consider first-order vocabularies with constants but no other function symbols. An *atom* is of the form $p(t_1, \ldots, t_k)$ where p is a predicate of arity k and the t_i are terms, i.e., variables or constants (in the paper we denote constants by $a, b, c, ...$ and variables by $x, y, z, ...$). An *atomset* is a set of atoms. Unless indicated otherwise, we will always consider *finite* atomsets. If F is an atom or an atomset, we write $terms(F)$ (resp. $vars(F)$, resp. $csts(F)$) the set of terms (resp. variables, resp. constants) that occur in F. If F is an atomset, we write $\phi(F)$ the formula obtained by the conjunction of all atoms in F, and $\Phi(F)$ the existential closure of $\phi(F)$. We say that an atomset F *entails* an atomset Q (notation $F \models Q$) if $\Phi(F) \models \Phi(Q)$. It is well-known that $F \models Q$ iff there exists a *homomorphism* from Q to F, i.e., a *substitution* $\sigma : vars(F) \to terms(Q)$ such that $\sigma(Q) \subseteq F$. Two atomsets F and F' are said to be *equivalent* if $F \models F'$ and $F' \models F$. If there is a homomorphism σ from an atomset F to itself (i.e., an *endomorphism* of F) then F and $\sigma(F)$ are equivalent. An atomset F is a *core* if there is no homomorphism from F to one of its strict subsets. Among all atomsets equivalent to an atomset F, there exists a unique core (up to isomorphism). We call this atomset *the* core of F.

Existential Rules An *existential rule* (and simply a rule hereafter) is of the form $B \to H$, where B and H are atomsets, respectively called the *body* and the *head* of the rule. To an existential rule $R : B \to H$ we assign a formula $\Phi(R) = \forall \vec{x} \forall \vec{y} (\phi(B) \to \exists \vec{z} \phi(H))$, where $vars(B) = \vec{x} \cup \vec{y}$, and $vars(H) = \vec{x} \cup \vec{z}$. Variables $\vec{x}$, which appear in both B and H, are called *frontier variables*, while variables $\vec{z}$, which appear only in H are called *existential variables*. E.g., $\Phi(b(x, y) \to h(x, z)) = \forall x \forall y (b(x, y) \to \exists z h(x, z))$. The presence of existential variables in rule heads is the distinguishing feature of existential rules.

A *knowledge base* is a pair $K = (F, \mathcal{R})$ where F is an atomset (the set of facts) and $\mathcal{R}$ is a finite set of existential rules. We say that $K = (F, \{R_1, \ldots, R_k\})$ *entails* an atomset Q (notation $K \models Q$) if $\Phi(F), \Phi(R_1), \ldots, \Phi(R_k) \models \Phi(Q)$. The fundamental problem we consider, denoted by ENTAILMENT, is the following : given a knowledge base K and an atomset Q, is it true that $K \models Q$? When $\Phi(Q)$ is seen as a Boolean conjunctive query, this problem is exactly the problem of determining if K yields a positive answer to this query.

A rule $R : B \to H$ is *applicable* to an atomset F if there is a homomorphism π from B to F. Then the *application of R to F according to π* produces an atomset $\alpha(F, R, \pi) = F \cup \pi(safe(H))$, where $safe(H)$ is obtained from H by replacing existential variables with fresh ones. An $\mathcal{R}$-derivation from F is a (possibly infinite) sequence $F_0 = \sigma_0(F), \ldots, \sigma_k(F_k), \ldots$ of atomsets such that $\forall 0 \leq i$, σ_i is an endomorphism of F_i (that will be used to remove redundancy in F_i) and $\forall 0 < i$, there is a rule $(R : B \to H) \in \mathcal{R}$ and a homomorphism π_i from B to $\sigma_i(F_{i-1})$ such that $F_i = \alpha(\sigma_i(F_{i-1}), R, \pi_i)$.

Example 1 *Consider the existential rule*

$$R = human(x) \to hasParent(x, y), human(y)\,;$$

and the atomset $F = \{human(a)\}$. The application of R to F produces an atomset $F' = F \cup \{hasParent(x, y_0), human(y_0)\}$ where y_0 is a fresh variable denoting an unknown individual. Note that R could be applied again to F' (mapping x to y_0), which would create another existential variable and so on.

A finite $\mathcal{R}$-derivation $F_0, \ldots, F_k$ from F is said to be *from F to F_k*. Given a knowledge base $K = (F, \mathcal{R})$, $K \models Q$ iff there exists a finite $\mathcal{R}$-derivation from F to F' such that $F' \models Q$ (Baget et al. 2011).

Let R_i and R_j be rules, and F be an atomset such that R_i is applicable to F by a homomorphism π ; a homomorphism π' from B_j to $F' = \alpha(F, R_i, \pi)$ is said to be *new* if $\pi'(B_j) \not\subseteq F$. Given a rule $R = B \to H$, a homomorphism π from B to F is said to be *useful* if it cannot be extended to a homomorphism from $B \cup H$ to F ; if π is not useful then $\alpha(F, R, \pi)$ is equivalent to F, but this is not a necessary condition for $\alpha(F, R, \pi)$ to be equivalent to F.

Chase Termination

An algorithm that computes an $\mathcal{R}$-derivation by exploring all possible rule applications in a breadth-first manner is called a *chase*. In the following, we will also call chase the derivation it computes. Different kinds of chase can be defined by using different properties to compute $F'_i = \sigma_i(F_i)$ in the derivation (hereafter we write F'_i for $\sigma_i(F_i)$ when there is no ambiguity). All these algorithms are sound and complete w.r.t. the ENTAILMENT problem in the sense that $(F, \mathcal{R}) \models Q$ iff they provide in finite (but unbounded) time a finite $\mathcal{R}$-derivation from F to F_k such that $F_k \models Q$.

Different kinds of chase In the *oblivious chase* (also called naive chase), e.g., (Calì, Gottlob, and Kifer 2008), a rule R is applied according to a homomorphism π only if it has not already been applied according to the same homomorphism. Let $F_i = \alpha(F'_{i-1}, R, \pi)$, then $F'_i = F'_{i-1}$ if R was previously applied according to π, otherwise $F'_i = F_i$. This can be slightly improved. Two applications π and π' of the same rule add the same atoms if they map frontier variables identically (for any frontier variable x of R, $\pi(x) = \pi'(x)$) ; we say that they are frontier-equal. In the *frontier chase*, let $F_i = \alpha(F'_{i-1}, R, \pi)$, we take $F'_i = F'_{i-1}$ if R was previously applied according to some π' frontier-equal to π, otherwise $F'_i = F_i$. The *skolem chase* (Marnette 2009) relies on a skolemisation of the rules : a rule R is transformed into a rule $skolem(R)$ by replacing each occurrence of an existential variable y with a functional term $f^R_y(\vec{x})$, where $\vec{x}$ are the frontier variables of R. Then the oblivious chase is run on skolemized rules. It can easily be checked that frontier chase and skolem chase yield isomorphic results, in the sense that they generate exactly the same atomsets, up to a bijective renaming of variables by skolem terms.

The *restricted chase* (also called standard chase) (Fagin et al. 2005) detects a kind of local redundancy. Let $F_i = \alpha(F'_{i-1}, R, \pi)$, then $F'_i = F_i$ if π is useful, otherwise $F'_i =$

F'_{i-1}. The *core chase* (Deutsch, Nash, and Remmel 2008) considers the strongest possible form of redundancy : for any F_i, F'_i is the core of F_i.

A chase is said to be *local* if $\forall i \leq j$, $F'_i \subseteq F'_j$. All chase variants presented above are local, *except for the core chase*. This property will be critical for nonmonotonic existential rules.

Chase termination Since ENTAILMENT is undecidable, the chase may not halt. We call *C-chase* a chase relying on some criterion C to generate $\sigma(F_i) = F'_i$. So C can be oblivious, skolem, restricted, core or any other criterion that ensures the equivalence between F_i and F'_i. A C-chase generates a possibly infinite $\mathcal{R}$-derivation $\sigma_0(F), \sigma_1(F_1), \ldots, \sigma_k(F_k), \ldots$

We say that this derivation *produces* the (possibly infinite) atomset $(F, \mathcal{R})^C = \cup_{0 \leq i \leq \infty} \sigma_i(F_i) \setminus \cup_{0 \leq i \leq \infty} \overline{(\sigma_i(F_i))}$, where $\overline{(\sigma_i(F_i))} = F_i \setminus \sigma(F_i)$. Note that this produced atomset is usually defined as the infinite union of the $\sigma_i(F_i)$. Both definitions are equivalent when the criterion C is *local*. But the usual definition would produce too big an atomset with a non-local chase such as the core chase : an atom generated at step i and removed at step j would still be present in the infinite union. We say that a (possibly infinite) derivation obtained by the C-chase is *complete* when any further rule application on that derivation would produce the same atomset. A complete derivation obtained by any C-chase produces a *universal model* (i.e., most general) of $(F, \mathcal{R})$: for any atomset Q, we have $F, \mathcal{R} \models Q$ iff $(F, \mathcal{R})^C \models Q$.

We say that the C-chase *halts* on $(F, \mathcal{R})$ when the C-chase generates a finite complete $\mathcal{R}$-derivation from F to F_k. Then $(F, \mathcal{R})^C = \sigma_k(F_k)$ is a finite universal model. We say that $\mathcal{R}$ is *universally C-terminating* when the C-chase halts on $(F, \mathcal{R})$ for any atomset F. We call *C-finite* the class of universally C-terminating sets of rules. It is well known that the chase variants do not behave in the same way w.r.t. termination. The following examples highlight these different behaviors.

Example 2 (Oblivious / Skolem chase) *Let $R = p(x, y) \rightarrow p(x, z)$ and $F = \{p(a, b)\}$. The oblivious chase does not halt : it adds $p(a, z_0)$, $p(a, z_1)$, etc. The skolem chase considers the rule $p(x, y) \rightarrow p(x, f^R_z(x))$; it adds $p(a, f^R_y(a))$ then halts.*

Example 3 (Skolem / Restricted chase) *Let $R : p(x) \rightarrow r(x, y), r(y, y), p(y)$ and $F = \{p(a)\}$. The skolem chase does not halt : at Step 1, it maps x to a and adds $r(a, f^R_y(a))$, $r(f^R_y(a), f^R_y(a))$ and $p(f^R_y(a))$; at step 2, it maps x to $f^R_y(a)$ and adds $r(f^R_y(a), f^R_y(f^R_y(a)))$, etc. The restricted chase performs a single rule application, which adds $r(a, y_0)$, $r(y_0, y_0)$ and $p(y_0)$; indeed, the rule application that maps x to y_0 yields only redundant atoms w.r.t. $r(y_0, y_0)$ and $p(y_0)$.*

Example 4 (Restricted / Core chase) *Let $F = s(a)$, $R_1 = s(x) \rightarrow p(x, x_1), p(x, x_2), r(x_2, x_2)$, $R_2 = p(x, y) \rightarrow q(y)$ and $R_3 = q(x) \rightarrow r(x, y), q(y)$. Note that R_1 creates redundancy and R_3 could be applied indefinitely if it were the only rule. R_1 is the first applied rule, which creates new variables, called x_1 and x_2 for simplicity. The restricted chase does not halt : R_3 is not applied on x_2 because it is already satisfied at this point, but it is applied on x_1, which creates an infinite chain. The core chase applies R_1, computes the core of the result, which removes $p(a, x_1)$, then halts.*

It is natural to consider the oblivious chase as the weakest form of chase and necessary to consider the core chase as the strongest form of chase (since the core is the minimal representative of its equivalence class). We say that a criterion C is *stronger* than C' and write $C \geq C'$ when C'-finite $\subseteq$ C-finite. We say that C is *strictly stronger* than C' (and write $C > C'$) when $C \geq C'$ and $C' \not\geq C$.

It is well-known that core > restricted > skolem > oblivious. An immediate remark is that core-finite corresponds to *finite expansion sets (fes)* defined in (Baget and Mugnier 2002). To sum up, the following inclusions hold between C-finite classes : oblivious-finite $\subset$ skolem-finite = frontier-finite $\subset$ restricted-finite $\subset$ core-finite = fes.

Known Acyclicity Notions

We can only give a brief overview of known acylicity notions, which should however allow to place our contribution within the existing landscape. A comprehensive taxonomy can be found in (Cuenca Grau et al. 2013).

Acyclicity notions ensuring that some chase variant terminates can be divided into two main families, each of them relying on a different graph : a "position-based" approach which basically relies on a graph encoding variable sharing between positions in predicates and a "rule dependency approach" which relies on a graph encoding dependencies between rules, i.e., the fact that a rule may lead to trigger another rule (or itself).

Position-based approach In the position-based approach, cycles identified as dangerous are those passing through positions that may contain existential variables ; intuitively, such a cycle means that the creation of an existential variable in a given position may lead to create another existential variable in the same position, hence an infinite number of existential variables. Acyclicity is then defined by the absence of dangerous cycles. The simplest notion of acyclicity in this family is that of *weak acyclicity (wa)* (Fagin et al. 2003) (Fagin et al. 2005), which has been widely used in databases. It relies on a directed graph whose nodes are the positions in predicates (we denote by (p, i) the position i in predicate p). Then, for each rule $R : B \rightarrow H$ and each variable x in B occurring in position (p, i), edges with origin (p, i) are built as follows : if x is a frontier variable, there is an edge from (p, i) to each position of x in H ; furthermore, for each existential variable y in H occurring in position (q, j), there is a special edge from (p, i) to (q, j). A set of rules is weakly acyclic if its associated graph has no cycle passing through a special edge.

Example 5 (Weak-acyclicity) *Let $R_1 = h(x) \rightarrow p(x, y)$, where y is an existential variable, and $R_2 = p(u, v), q(v) \rightarrow h(v)$. The position graph of $\{R_1, R_2\}$ contains a special edge from $(h, 1)$ to $(p, 2)$ due to R_1 and an edge from $(p, 2)$ to $(h, 1)$ due to R_2, thus $\{R_1, R_2\}$ is not wa.*

Weak-acyclicity has been generalized, mainly by shifting the focus from positions to existential variables (*joint-acyclicity (ja)*(Krötzsch and Rudolph 2011)) or to positions in atoms instead of predicates (*super-weak-acyclicity (swa)* (Marnette 2009)). Other related notions can be imported

from logic programming, e.g., *finite domain (fd)* (Calimeri et al. 2008) and *argument-restricted (ar)* (Lierler and Lifschitz 2009). See the first column in Figure 1, which shows the inclusions between the corresponding classes of rules (all these inclusions are known to be strict).

Rule Dependency In the second approach, the aim is to avoid cyclic triggering of rules (Baget 2004; Baget et al. 2009; Deutsch, Nash, and Remmel 2008; Cuenca Grau et al. 2012). We say that a rule R_2 *depends* on a rule R_1 if there exists an atomset F such that R_1 is applicable to F according to a homomorphism π and R_2 is applicable to $F' = \alpha(F, R_1, \pi)$ according to a new useful homomorphism. This abstract dependency relation can be effectively computed with a unification operation known as piece-unifier (Baget et al. 2009). Piece-unification takes existential variables into account, hence is more complex than the usual unification between atoms. A *piece-unifier* of a rule body B_2 with a rule head H_1 is a substitution μ of $vars(B'_2) \cup vars(H'_1)$, where $B'_2 \subseteq B_2$ and $H'_1 \subseteq H_1$, such that *(1)* $\mu(B'_2) = \mu(H'_1)$ and *(2)* existential variables in H'_1 are not unified with separating variables of B'_2, i.e., variables that occur both in B'_2 and in $(B_2 \setminus B'_2)$; in other words, if a variable x occuring in B'_2 is unified with an existential variable y in H'_1, then all atoms in which x occurs also belong to B'_2. It holds that R_2 depends on R_1 iff there is a piece-unifier of B_2 with H_1 satisfying easy to check additional conditions (atom erasing (Baget et al. 2011) and usefulness (Cuenca Grau et al. 2013)).

Example 6 (Rule dependency) *Consider the rules from Example 5. There is no piece-unifier of B_2 with H_1. The substitution $\mu = \{(u, x), (v, y)\}$, with $B'_2 = p(u, v)$ and $H'_1 = H_1$, is not a piece-unifier because v is unified with an existential variable, whereas it is a separating variable of B'_2 (thus, $q(v)$ should be included in B'_2, which is impossible). Thus R_2 does not depend on R_1.*

The *graph of rule dependencies* of a set of rules $\mathcal{R}$, denoted by GRD($\mathcal{R}$), encodes the dependencies between rules in $\mathcal{R}$. It is a directed graph with set of nodes $\mathcal{R}$ and an edge (R_i, R_j) if R_j depends on R_i (intuition : "R_i may lead to trigger R_j in a new way"). E.g., considering the rules in Example 6, the only edge is (R_2, R_1).

When the GRD is acyclic (*aGRD*, (Baget 2004)), any derivation sequence is necessarily finite. This notion is incomparable with those based on positions.

We point out here that the *oblivious* chase may not stop on *wa* rules. Thus, the only acyclicity notion in Figure 1 that ensures the termination of the oblivious chase is *aGRD* since all other notions generalize *wa*.

Combining both approches Both approaches have their weaknesses : there may be a dangerous cycle on positions but no cycle w.r.t. rule dependencies (see the preceeding examples), and there may be a cycle w.r.t. rule dependencies whereas rules contain no existential variables (e.g. $p(x, y) \rightarrow p(y, x), q(x)$). Attempts to combine both notions only succeded to combine them in a "modular way" : if the rules in each strongly connected component (s.c.c.) of the GRD belong to a *fes* class, then the set of rules is *fes* (Baget 2004; Deutsch, Nash, and Remmel 2008). More specifically, it is easy to check that if for a given C-chase, each s.c.c. is C-finite, then the C-chase stops.

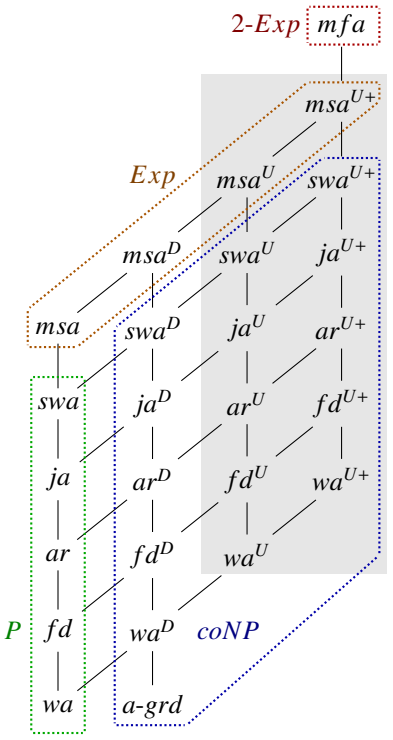


FIGURE 1 – Relations between recognizable acyclicity properties. All inclusions are strict and complete (i.e., if there is no path between two properties then they are incomparable).

In this paper, we propose an "integrated" way of combining both approaches, which relies on a single graph. This allows to unify preceding results and to generalize them without complexity increasing (the new acyclicity notions are those with a gray background in Figure 1).

Finally, let us mention *model-faithful acyclicity (mfa)* (Cuenca Grau et al. 2012), which generalizes the previous acyclicity notions and cannot be captured by our approach. Briefly, mfa involves running the skolem chase until termination or a cyclic functional term is found. The price to pay for the generality of this property is high complexity : checking if a set of rules is universally mfa (i.e., for any set of facts) is 2EXPTIME-complete. Checking *model-summarizing acyclicity (msa)*, which approximates mfa, remains EXPTIME-complete. In contrast, checking position-based properties is in PTIME and checking agrd is also co-NP-complete. Sets of rules satisfying mfa are skolem-finite (Cuenca Grau et al. 2012), thus all properties studied in this paper ensure C-finiteness, when $C \geq$ skolem.

Extending Acyclicity Notions

In this section, we combine rule dependency and propagation of existential variables into a single graph. W.l.o.g. we assume that distinct rules do not share any variable. Gi-

ven an atom $a = p(t_1, \ldots, t_k)$, the i^{th} position in a is denoted by $\langle a, i\rangle$, with $pred(\langle a, i\rangle) = p$ and $term(\langle a, i\rangle) = t_i$. If $a \in A$, we say that $\langle a, i\rangle$ is in A. If $term(\langle a, i\rangle)$ is an existential (resp. frontier) variable, $\langle a, i\rangle$ is called an *existential* (resp. *frontier*) position. In the following, we use "position graph" as a generic name to denote a graph whose nodes are positions in *atoms*. We define several position graphs of increasing expressivity, i.e., allowing to check termination for increasingly larger classes of rules.

Definition 1 ((Basic) Position Graph (PG)) *The* position graph *of a rule $R : B \to H$ is the directed graph $PG(R)$ defined as follows :*

- *there is a node for each $\langle a, i\rangle$ in B or in H ;*
- *for all frontier position $\langle b, i\rangle \in B$ and all $\langle h, j\rangle \in H$, there is an edge from $\langle b, i\rangle$ to $\langle h, j\rangle$ if $term(\langle b, i\rangle) = term(\langle h, j\rangle)$ or if $\langle h, j\rangle$ is existential.*

Given a set of rules $\mathcal{R}$, the basic position graph *of $\mathcal{R}$, denoted by $PG(\mathcal{R})$, is the disjoint union of $PG(R_i)$, for all $R_i \in \mathcal{R}$.*

An existential position $\langle a, i\rangle$ is said to be *C-infinite* if there is an atomset F such that running the C-chase on F produces an unbounded number of instantiations of $term(\langle a, i\rangle)$. In what follows, we consider any chase that is stronger than the skolem chase, and will simply denote such a position by an infinite position, without further reference to the chase used. To detect infinite positions, we encode how variables may be "propagated" among rules by adding edges to $PG(\mathcal{R})$, called *transition edges*, which go from positions in rule heads to positions in rule bodies. The set of transition edges has to be *correct* : if an existential position $\langle a, i\rangle$ is infinite, there must be a cycle going through $\langle a, i\rangle$ in the graph.

Definition 2 (PG^X) *Let $\mathcal{R}$ be a set of rules. The three following position graphs are obtained from $PG(\mathcal{R})$ by adding a (transition) edge from each position p_h in a rule head H_i to each position p_b in a rule body B_j, with the* same predicate*, provided that some condition is satisfied :*

- full PG*, denoted by $PG^F(\mathcal{R})$: no additional condition,*
- dependency PG *($PG^D(\mathcal{R})$) : if R_j depends on R_i,*
- PG with unifiers *($PG^U(\mathcal{R})$) : if there is a piece-unifier μ of B_j with H_i such that $\mu(term(p_h)) = \mu(term(p_b))$.*

All three position graphs are correct for chases stronger than the skolem one. Intuitively, $PG^F(\mathcal{R})$ corresponds to the case where all rules are supposed to depend on all rules ; its set of cycles is in bijection with the set of cycles in the predicate position graph defining weak-acyclicity. $PG^D(\mathcal{R})$ encodes actual rule dependencies. Finally, $PG^U(\mathcal{R})$ adds information about the piece-unifiers themselves. This provides an accurate encoding of variable propagation from an atom position to another.

Proposition 1 (Inclusions between PG^X) *Let $\mathcal{R}$ be a set of rules. $PG^U(\mathcal{R}) \subseteq PG^D(\mathcal{R}) \subseteq PG^F(\mathcal{R})$. Furthermore, $PG^D(\mathcal{R}) = PG^F(\mathcal{R})$ if $GRD(\mathcal{R})$ is a complete graph.*

Example 7 (PG^F and PG^D) *Let $\mathcal{R} = \{R_1, R_2\}$ from Ex. 5. Figure 2 pictures $PG^F(\mathcal{R})$ and $PG^D(\mathcal{R})$. The dashed edges belong to $PG^F(\mathcal{R})$ but not to $PG^D(\mathcal{R})$. Indeed, R_2 does not depend on R_1. $PG^F(\mathcal{R})$ has a cycle while $PG^D(\mathcal{R})$ has not.*

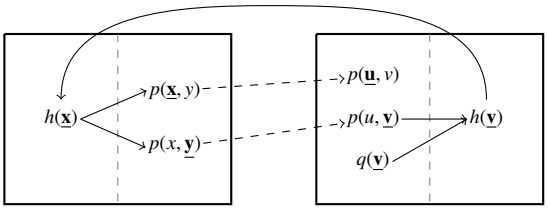


FIGURE 2 – $PG^F(\mathcal{R})$ and $PG^D(\mathcal{R})$ from Ex. 7. Position $\langle a, i\rangle$ is represented by underlining the i-th term in a.

Example 8 (PG^D and PG^U) *Let $\mathcal{R} = \{R_1, R_2\}$, with $R_1 = t(x, y) \to p(z, y), q(y)$ and $R_2 = p(u, v), q(u) \to t(v, w)$. In Figure 3, the dashed edges belong to $PG^D(\mathcal{R})$ but not to $PG^U(\mathcal{R})$. Indeed, the only piece-unifier of B_2 with H_1 unifies u and y. Hence, the cycle in $PG^D(\mathcal{R})$ disappears in $PG^U(\mathcal{R})$.*

We now study how acyclicity properties can be expressed on position graphs. The idea is to associate, with an acyclicity property, a function that assigns to each position a subset of positions reachable from this position, according to some propagation constraints ; then, the property is fulfilled if no existential position can be reached from itself. More precisely, a *marking function Y* assigns to each node $\langle a, i\rangle$ in a position graph PG^X, a subset of its (direct or indirect) successors, called its *marking*. A *marked cycle* for $\langle a, i\rangle$ (w.r.t. X and Y) is a cycle C in PG^X such that $\langle a, i\rangle \in C$ and for all $\langle a', i'\rangle \in C$, $\langle a', i'\rangle$ belongs to the marking of $\langle a, i\rangle$. Obviously, the less situations there are in which the marking may "propagate" in a position graph, the stronger the acyclicity property is.

Definition 3 (Acyclicity property) *Let Y be a marking function and PG^X be a position graph. The* acyclicity property *associated with Y in PG^X, denoted by Y^X, is satisfied if there is no marked cycle for an existential position in PG^X. If Y^X is satisfied, we also say that $PG^X(\mathcal{R})$* satisfies *Y.*

For instance, the marking function associated with weak-acyclicity assigns to each node the set of its successors in $PG^F(\mathcal{R})$, without any additional constraint. The next proposition states that such marking functions can be defined for each class of rules between *wa* and *swa* (first column in Figure 1), in such a way that the associated acyclicity property in PG^F characterizes this class.

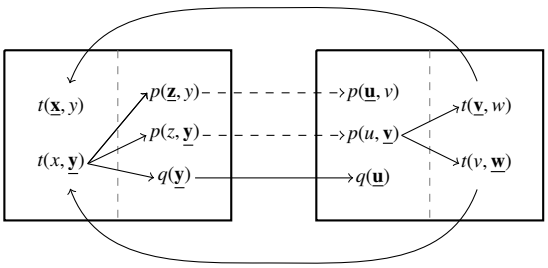


FIGURE 3 – $PG^D(\mathcal{R})$ and $PG^U(\mathcal{R})$ from Ex. 8.

Proposition 2 *A set of rules $\mathcal{R}$ is wa (resp. fd, ar, ja, swa) iff $PG^F(\mathcal{R})$ satisfies the acyclicity property associated with wa- (resp. fd-, ar-, ja-, swa-) marking.*

As already mentioned, all these classes can be safely extended by combining them with the GRD. To formalize this, we recall the notion $Y^<$ from (Cuenca Grau et al. 2013) : given an acyclicity property Y, a set of rules $\mathcal{R}$ is said to satisfy $Y^<$ if each s.c.c. of $GRD(\mathcal{R})$ satisfies Y, except for those composed of a single rule with no loop. [1] Whether $\mathcal{R}$ satisfies $Y^<$ can be checked on $PG^D(\mathcal{R})$:

Proposition 3 *Let $\mathcal{R}$ be a set of rules, and Y be an acyclicity property. $\mathcal{R}$ satisfies $Y^<$ iff $PG^D(\mathcal{R})$ satisfies Y, i.e., $Y^< = Y^D$.*

For the sake of brevity, if Y_1 and Y_2 are two acyclicity properties, we write $Y_1 \subseteq Y_2$ if any set of rules satisfying Y_1 also satisfies Y_2. The following results are straightforward.

Proposition 4 *Let Y_1, Y_2 be two acyclicity properties. If $Y_1 \subseteq Y_2$, then $Y_1^D \subseteq Y_2^D$.*

Proposition 5 *Let Y be an acyclicity property. If a-grd $\not\subseteq Y$ then $Y \subset Y^D$.*

Hence, any class of rules satisfying a property Y^D strictly includes both *a-grd* and the class characterized by Y ; (e.g., Figure 1, from Col. 1 to Col. 2). More generally, strict inclusion in the first column leads to strict inclusion in the second one :

Proposition 6 *Let Y_1, Y_2 be two acyclicity properties such that $Y_1 \subset Y_2$, wa $\subseteq Y_1$ and $Y_2 \not\subseteq Y_1^D$. Then $Y_1^D \subset Y_2^D$.*

The next theorem states that PG^U is strictly more powerful than PG^D ; moreover, the "jump" from Y^D to Y^U is at least as large as from Y to Y^D.

Theorem 1 *Let Y be an acyclicity property. If $Y \subset Y^D$ then $Y^D \subset Y^U$. Furthermore, there is an injective mapping from the sets of rules satisfying Y^D but not Y, to the sets of rules satisfying Y^U but not Y^D.*

Proof: Assume $Y \subset Y^D$ and $\mathcal{R}$ satisfies Y^D but not Y. $\mathcal{R}$ can be rewritten into $\mathcal{R}'$ by applying the following steps. First, for each rule $R_i = B_i[\vec{X}, \vec{Y}] \rightarrow H_i[\vec{Y}, \vec{Z}] \in \mathcal{R}$, let $R_{i,1} = B_i[\vec{X}, \vec{Y}] \rightarrow p_i(\vec{X}, \vec{Y})$ where p_i is a fresh predicate ; and $R_{i,2} = p_i(\vec{X}, \vec{Y}) \rightarrow H_i[\vec{Y}, \vec{Z}]$. Then, for each rule $R_{i,1}$, let $R'_{i,1}$ be the rule $(B'_{i,1} \rightarrow H_{i,1})$ with $B'_{i,1} = B_{i,1} \cup \{p'_{j,i}(x_{j,i}) : \forall R_j \in \mathcal{R}\}$, where $p'_{j,i}$ are fresh predicates and $x_{j,i}$ fresh variables. Now, for each rule $R_{i,2}$, let $R'_{i,2}$ be the rule $(B_{i,2} \rightarrow H'_{i,2})$ with $H'_{i,2} = H_{i,2} \cup \{p'_{i,j}(z_{i,j}) : \forall R_j \in \mathcal{R}\}$, where $z_{i,j}$ are fresh existential variables. Let $\mathcal{R}' = \bigcup_{R_i \in \mathcal{R}} \{R'_{i,1}, R'_{i,2}\}$. This construction ensures that each $R'_{i,2}$ depends on $R'_{i,1}$, and each $R'_{i,1}$ depends on each $R'_{j,2}$, thus, there is a *transition* edge from each $R'_{i,1}$ to $R'_{i,2}$ and from each $R'_{j,2}$ to each $R'_{i,1}$. Hence, $PG^D(\mathcal{R}')$ contains exactly one cycle for each cycle in $PG^F(\mathcal{R})$. Furthermore, $PG^D(\mathcal{R}')$ contains at least one marked cycle wrt Y, and then $\mathcal{R}'$ is not Y^D. Now, each cycle in $PG^U(\mathcal{R}')$ is also a cycle in $PG^D(\mathcal{R})$, and, since $PG^D(\mathcal{R})$ satisfies Y, $PG^U(\mathcal{R}')$ also does. Hence, $\mathcal{R}'$ does not belong to Y^D but to Y^U. □

We also check that strict inclusions in the second column in Figure 1 lead to strict inclusions in the third column.

Theorem 2 *Let Y_1 and Y_2 be two acyclicity properties. If $Y_1^D \subset Y_2^D$ then $Y_1^U \subset Y_2^U$.*

Proof: Let $\mathcal{R}$ be a set of rules such that $\mathcal{R}$ satisfies Y_2^D but does not satisfy Y_1^D. We rewrite $\mathcal{R}$ into $\mathcal{R}'$ by applying the following steps. For each pair of rules $R_i, R_j \in \mathcal{R}$ such that R_j depends on R_i, for each variable x in the frontier of R_j and each variable y in the head of R_i, if x and y occur both in a given predicate position, we add to the body of R_j a new atom $p_{i,j,x,y}(x)$ and to the head of R_i a new atom $p_{i,j,x,y}(y)$, where $p_{i,j,x,y}$ denotes a fresh predicate. This construction allows each term from the head of R_i to propagate to each term from the body of R_j, if they shared some predicate position in $\mathcal{R}$. Thus, any cycle in $PG^D(\mathcal{R})$ is also in $PG^U(\mathcal{R}')$, without any change in the behavior w.r.t. the acyclicity properties. Hence $\mathcal{R}'$ satisfies Y_2^U but does not satisfy Y_1^U. □

The next result states that Y^U is a sufficient condition for chase termination :

Theorem 3 *Let Y be an acyclicity property ensuring the halting of some chase variant C. Then, the C-chase halts for any set of rules $\mathcal{R}$ that satisfies Y^U (hence Y^D).*

Finally, we remind that classes from *wa* to *swa* can be recognized in PTIME, and checking *a-grd* is coNP-complete. The next result shows that checking the more expressive Y^U instead of Y^D is made at no additional complexity cost.

Theorem 4 (Complexity) *Let Y be an acyclicity property, and $\mathcal{R}$ be a set of rules. If checking that $\mathcal{R}$ is Y is in coNP, then checking that $\mathcal{R}$ is Y^D or Y^U is coNP-complete.*

Further Refinements

Still without complexity increasing, we can further extend Y^U into Y^{U^+} by a finer analysis of marked cycles and unifiers. We define the notion of *incompatible* sequence of unifiers, which ensures that a given sequence of rule applications is impossible. Briefly, a marked cycle for which all sequences of unifiers are incompatible can be ignored. Beside the gain for positive rules, this refinement will allow one to take better advantage of negation.

We first point out that the notion of piece-unifier is not appropriate to our purpose. We have to relax it, as illustrated by the next example. We call *unifier*, of a rule body B_2 with a rule head H_1, a substitution μ of $vars(B'_2) \cup vars(H'_1)$, where $B'_2 \subseteq B_2$ and $H'_1 \subseteq H_1$, such that $\mu(B'_2) = \mu(H'_1)$ (thus, it satisfies Condition (1) of a piece-unifier).

Example 9 *Let $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$ with :*
$R_1 : p(x_1, y_1) \rightarrow q(y_1, z_1)$
$R_2 : q(x_2, y_2) \rightarrow r(x_2, y_2)$
$R_3 : r(x_3, y_3) \wedge s(x_3, y_3) \rightarrow p(x_3, y_3)$
$R_4 : q(x_4, y_4) \rightarrow s(x_4, y_4)$
There is a dependency cycle (R_1, R_2, R_3, R_1) and a corresponding cycle in PG^U. We want to know if such a sequence of rule applications is possible. We build the following new

1. This particular case is to cover *aGRD*, in which each s.c.c. is an isolated node.

rule, which is a composition of R_1 and R_2 (formally defined later) : $R_1 \diamond_\mu R_2 : p(x_1, y_1) \rightarrow q(y_1, z_1) \wedge r(y_1, z_1)$
There is no piece-unifier of R_3 with $R_1 \diamond_\mu R_2$, since y_3 would be a separating variable mapped to the existential variable z_1. This actually means that R_3 is not applicable right after *$R_1 \diamond_\mu R_2$. However, the atom needed to apply $s(x_3, y_3)$ can be brought by a sequence of rule applications (R_1, R_4). We thus relax the notion of piece-unifier to take into account arbitrary long sequences of rule applications.*

Definition 4 (Compatible unifier) *Let R_1 and R_2 be rules. A unifier μ of B_2 with H_1 is* compatible *if, for each position $\langle a, i\rangle$ in B'_2, such that $\mu(\text{term}(\langle a, i\rangle))$ is an existential variable z in H'_1, $PG^U(\mathcal{R})$ contains a path, from a position in which z occurs, to $\langle a, i\rangle$, that does not go through another existential position. Otherwise, μ is* incompatible.

Note that a piece-unifier is necessarily compatible.

Proposition 7 *Let R_1 and R_2 be rules, and let μ be a unifier of B_2 with H_1. If μ is incompatible, then no application of R_2 can use an atom in $\mu(H_1)$.*

We define the rule corresponding to the composition of R_1 and R_2 according to a compatible unifier, then use this notion to define a compatible sequence of unifiers.

Definition 5 (Unified rule, Compatible sequence of unifiers)

• *Let R_1 and R_2 be rules such that there is a compatible unifier μ of B_2 with H_1. The associated* unified rule *$R_\mu = R_1 \diamond_\mu R_2$ is defined by $H_\mu = \mu(H_1) \cup \mu(H_2)$, and $B_\mu = \mu(B_1) \cup (\mu(B_2) \setminus \mu(H_1))$.*
• *Let $(R_1, \ldots, R_{k+1})$ be a sequence of rules. A sequence $s = (R_1\ \mu_1\ R_2 \ldots\ \mu_k\ R_{k+1})$, where, for $1 \leq i \leq k$, μ_i is a unifier of B_{i+1} with H_i, is a* compatible sequence *of unifiers if :* (1) *μ_1 is a compatible unifier of B_2 with H_1, and* (2) *if $k > 0$, the sequence obtained from s by replacing $(R_1\ \mu_1\ R_2)$ with $R_1 \diamond_{\mu_1} R_2$ is a* compatible *sequence of unifiers.*

E.g., in Example 9, the sequence $(R_1\ \mu_1\ R_2\ \mu_2\ R_3\ \mu_3\ R_1)$, with the obvious μ_i, is compatible. We can now improve all previous acyclicity properties (see the fourth column in Figure 1).

Definition 6 (Compatible cycles) *Let Y be an acyclicity property, and PG^U be a position graph with unifiers. The* compatible cycles for $\langle a, i\rangle$ *in PG^U are all marked cycles C for $\langle a, i\rangle$ wrt Y, such that there is a compatible sequence of unifiers induced by C. Property $Y^{U+}$is satisfied if, for each existential position $\langle a, i\rangle$, there is no compatible cycle for $\langle a, i\rangle$ in PG^U.*

Results similar to Theorem 1 and Theorem 2 are obtained for Y^{U^+} w.r.t. Y^U, namely :

- For any acyclicity property Y, $Y^U \subset Y^{U+}$.
- For any acyclicity properties Y_1 and Y_2, if $Y_1^U \subset Y_2^U$, then $Y_1^{U+} \subset Y_2^{U+}$.

Moreover, Theorem 3 can be extended to Y^{U^+} : let Y be an acyclicity property ensuring the halting of some chase variant C ; then the C-chase halts for any set of rules $\mathcal{R}$ that satisfies Y^{U^+} (hence Y^U). Finally, the complexity result from Theorem 4 still holds for this improvement.

Handling Nonmonotonic Negation

We now add nonmonotonic negation, which we denote by **not**. A *nonmonotonic existential rule* (NME rule) R is of the form $(B^+, \mathbf{not}B_1^-, \ldots, \mathbf{not}B_k^- \rightarrow H)$, where B^+, B_i^- and H are atomsets, respectively called the *positive* body, the *negative* bodies and the head of R. Note that we generalize the usual notion of negative body by allowing to negate conjunctions of atoms. Moreover, the rule head may contain several atoms. However, we impose a safeness condition : $\forall 1 \leq i \leq k$, $vars(B_i^-) \subseteq vars(B^+)$. The formula assigned to R is $\Phi^{not}(R) = \forall \vec{x} \forall \vec{y} (\phi(B^+) \wedge \mathbf{not}\phi(B_1^-), \ldots, \mathbf{not}\phi(B_k^-) \rightarrow \exists \vec{z} \phi(H)$. We write $pos(R)$ the existential rule obtained from R by removing its negative bodies, and $pos(\mathcal{R})$ the set of all $pos(R)$ rules, for $R \in \mathcal{R}$.

About our Stable Model Semantics Answer Set Programming (Gelfond 2007) introduced stable model semantics for propositional logic, and was naturally extended to grounded programs (*i.e.,* sets of NME rules without variables). In this framework, the semantics can be provided through the Gelfond-Lifschitz reduct operator that allows to compute a saturation (*i.e.,* a chase) using only grounded NME rules. This semantics can be easily extended to rules with no existential variable in the head, or to skolemized NME rules, as done, for instance, in (Magka, Krötzsch, and Horrocks 2013). The choice of the chase/saturation mechanism is here irrelevant, since no such mechanism can produce any redundancy.

The problem comes when considering existential variables in the head of rules. Several semantics have been proposed in that case, for instance circumscription in (Ferraris, Lee, and Lifschitz 2011), or justified stable models in (You, Zhang, and Zhang 2013). We have chosen not to adopt circumscription since it translates NME rules to second-order expressions, and thus would not have allowed to build upon results obtained in the existential rule formalism. In the same way, we have not considered justified stable models, whose semantics does not correspond to stable models on grounded rules, as shown by the following example :

Example 10 *Let $\Pi_1 = \{\emptyset \rightarrow p(a); p(a), \mathbf{not}\, q(a) \rightarrow t(a).\}$ be a set of ground NME rules. Then $\{p(a); q(a)\}$ is a justified stable model, but not a stable model. Let $\Pi_2 = \{\emptyset \rightarrow p(a); p(a), \mathbf{not}\, q(b) \rightarrow t(a)\}$. Then $\{p(a); t(a)\}$ is a stable model but not a justified stable model.*

Let us now recast the Gelfond-Lifschitz reduct-based semantics in terms of the skolem-chase. Essentially (we will be more precise in the next section), a stable model M is a possibly infinite atomset produced by a skolem-chase that respects some particular conditions :

- all rule applications are sound, *i.e.,* none of its negative bodies can be found in the stable model produced (the rule is not blocked) ;
- the derivation is complete, *i.e.,* any rule applicable and not blocked is applied in the derivation.

In the next subsection, we formally define the notion of a stable model, while replacing the skolem-chase with any C-chase. We thus obtain a family of semantics parameterized by the considered chase, and define different notions of C-stable models.

On the Chase and Stable Models We define a notion of stable model directly on nonmonotonic existential rules and provide a derivation algorithm inspired from the notion of computation in (Liu et al. 2010) and Answer Set Programming solvers that instantiate rules on the fly (Lefèvre and Nicolas 2009; Dao-Tran et al. 2012) instead of grounding rules before applying them. The difference with our framework is that they consider normal logic programs, which are a generalization of skolemized NME rules.

A natural question is then to understand if the choice of a chase mechanism has an impact, not only on the termination, but also on the semantics. Thus, we consider the chase as a parameter. Intuitively, a C-stable set A is produced by a C-chase that, according to (Gelfond 2007), must satisfy the NME rules (we say that it is *sound*, *i.e.,* that no negative body appearing in the chase is in A) and the *rationality principle* (the sound chase does not generate anything that cannot be believed, and it must be complete : any rule application not present in the chase would be unsound).

To define *C-stable sets*, we first need to introduce additional notions. A *NME $\mathcal{R}$-derivation from F* is a $pos(\mathcal{R})$-derivation from $\mathcal{R}$. This derivation $D = (F_0 = \sigma_0(F), \ldots, \sigma_k(F_k), \ldots)$ produces a possibly infinite atomset A. Let R be a NME rule such that $pos(R)$ was applied at some step i in D, *i.e.,* $F_{i+1} = \alpha(\sigma_i(F_i), pos(R), \pi_i)$. We say that this application is *blocked* if one of the $\pi_i(B_q^-)$ (for any negative body B_q^- in R) can be found in A. This can happen in two ways. Either $\pi_i(B_q^-)$ can already be found in $\sigma_i(F_i)$ or it appears later in the derivation. In both cases, there is a $\sigma_j(F_j)$ (with $j \geq i$) that contains the atomset $\pi_i(B_q^-)$, *as transformed by the sequence of simplifications from F_i to F_j, i.e.,* there exists F_j with $j \geq i$ s.t. the atomset $\sigma_{i\to j}(\pi_i(B_q^-)) = \sigma_j(\ldots(\sigma_{i+1}(\pi_i(B_q^-)))\ldots)$ is included in $\sigma_j(F_j)$. We say that a derivation D is *sound* when no rule application is blocked in A. A sound derivation is said to be *complete* when adding any other rule application to the derivation would either make it unsound, or would not change the produced atomset. The derivation is a C-chase when the σ_i used at each step is determined by the criterion C.

Definition 7 (C-stable sets) *Let F be a finite atomset, and $\mathcal{R}$ be a set of NME rules. We say that a (possibly infinite) atomset A is C-stable for $(F, \mathcal{R})$ if there is a complete sound nonmonotonic C-chase from F that produces A.*

Proposition 8 *If $\mathcal{R}$ is a set of existential rules, then there is a unique C-stable set, which is equivalent to the universal model $(F, \mathcal{R})^C$. If $\{F\} \cup \mathcal{R}$ is a set of skolemized NME rules (with F being seen as a rule with empty body), then its skolem-stable sets are in bijection with its stable models.*

Sketch of proof : First part of the claim stems from the fact that existential rules generate a unique branch that corresponds to a derivation. When that branch is complete, it corresponds to a chase. Second part of the claim comes from the fact that our definitions mimic the behavior of the sound and complete algorithm implemented in (Lefèvre and Nicolas 2009). □

C-chase Tree The problem with the fixpoint Definition 7 is that it does not provide an effective algorithm : at each step of the derivation, we need to know the set produced by that derivation. The algorithm used in the solver ASPéRIX (Lefèvre and Nicolas 2009) is here generalized to a procedure that generates the (possibly infinite) *C-derivation tree* of $(F, \mathcal{R})$. All nodes of that tree are labeled by three fields. The field IN contains the atomset that was inferred in the current branch. The field OUT contains the set of forbidden atomsets, *i.e.,* that must not be inferred. Finally, the field MBT ("must be true") contains the atomset that has yet to be proven. A node is called *unsound* when a forbidden atomset has been inferred, or has to be proven, *i.e.,* when $\text{OUT} \cap (\text{IN} \cup \text{MBT}) \neq \emptyset$. At the initial step, the root of the C-derivation tree is a positive node labeled $(\sigma_0(F), \emptyset, \emptyset)$. Then, let us chose a node N that is not unsound and has no child. Assume there is a rule $R = B^+, \textbf{not}B_1^-, \ldots, \textbf{not}B_k^- \to H$ in $\mathcal{R}$ such that there is a homomorphism π from B^+ to $\text{IN}(N)$. Then we will (possibly) add $k+1$ children under N, namely $N^+, N_1^-, \ldots, N_k^-$. These children are added if the rule application is not blocked, and produces new atoms. Intuitively, the positive child N^+ encodes the effective application of the rule, while the k negative children N_i^- encode the k different possibilities of blocking the rule (with each of the negative bodies). Let us consider the sequence of positive nodes from the root of the tree to N^+. It encodes a $pos(\mathcal{R})$-derivation from F. On that derivation, the C-chase generates a sequence $\sigma_0(F), \ldots, \sigma_p(F_p), S = \sigma(\alpha(\sigma_p(F_p), pos(R), \pi))$. S produces something new when $S \not\subseteq \sigma_p(F_p)$. We now have to fill the fields of the obtained children : let (IN, OUT, MBT) be the label of a node N. Then $label(N^+) = (S, \text{OUT} \cup \{\pi_i(B_1^-), \ldots, \pi_i(B_k^-)\}, \text{MBT})$ and $label(N_i^-) = (\text{IN}, \text{OUT}, \text{MBT} \cup \pi_i(B_i^-))$.

We say that a (possibly infinite) branch in the C-derivation tree is *unsound* when it contains an unsound node. A sound branch is said to be *complete* when its associated derivation is complete. Finally, a sound and complete branch is *stable* when for every node N in the branch such that $B^- \in \text{MBT}(N)$, there exists a descendant N' of N such that $B^- \in \text{IN}(N')$. We say that a branch is *unprovable* if there exists a node N in the branch and an atomset $B^- \in \text{MBT}(N)$ such that no complete branch containing N is stable. We call a *C-chase tree* any C-derivation tree for which all branches are either unsound, unprovable or complete.

Proposition 9 *An atomset A is a C-stable set for $(F, \mathcal{R})$ iff a C-chase tree of $(F, \mathcal{R})$ contains a stable branch whose associated derivation produces A.*

On the applicability of the chase variants In the positive case, all chase variants produce equivalent universal models (up to skolemization). Moreover, running a chase on equivalent knowledge bases produce equivalent results. Do these semantic properties still hold with nonmonotonic existential rules ? The answer is no in general.

The next example shows that the chase variants presented in this paper, core chase excepted, may produce non-equivalent results from equivalent knowledge bases.

Example 11 *Let $F = \{p(a, y), t(y)\}$ and $F' = \{p(a, y'), p(a, y), t(y)\}$ be two equivalent atomsets. Let $R : p(u, v), \textbf{not}\, t(v) \to r(u)$. For any C-chase other than core chase, there is a single C-stable set for $(F, \{R\})$ which is F*

(or sk(F)*) and a single C-stable set for* $(F', \{R\})$ *which is* $F' \cup \{r(a)\}$ *(or* $sk(F') \cup \{r(a)\}$*). These sets are not equivalent.*

Of course, if we consider that the initial knowledge base is already skolemized (including F seen as a rule), this trouble does not occur with the skolem-chase since there are no redundancies in facts and no redundancy can be created by a rule application. This problem does not arise with core chase either. Thus the only two candidates for processing NME rules are the core chase and the skolem chase (if we assume *a priori* skolemisation, which is already a semantic shift).

The choice between both mechanisms is important since, as shown by the next example, they may produce different results even when they both produce a *unique* C-stable set. It follows that skolemizing existential rules is not an innocuous transformation in presence of nonmontonic negation.

Example 12 *We consider* $F = i(a)$, $R_1 = i(x) \rightarrow p(x, y)$, $R_2 = i(x) \rightarrow q(x, y)$, $R_3 = q(x, y) \rightarrow p(x, y), t(y)$ *and* $R_4 = p(u, v), \mathbf{not}\, t(v) \rightarrow r(u)$*. The core chase produces at first step* $p(a, y_0)$ *and* $q(a, y_1)$*, then* $p(a, y_1)$ *and* $t(y_1)$ *and removes the redundant atom* $p(a, y_0)$*, hence* R_4 *is not applicable. The unique core-stable set is* $\{i(a), q(a, y_1), p(a, y_1), t(y_1)\}$*. With the skolem chase, the produced atoms are* $p(a, f^{R_1}(a))$ *and* $q(a, f^{R_2}(a))$*, then* $p(a, f^{R_2}(a))$ *and* $t(f^{R_2}(a))$*.* R_4 *is applied with* $p(u, v)$ *mapped to* $p(a, f^{R_1}(a))$*, which produces* $r(a)$*. These atoms yield a unique skolem-stable set. These stable sets are not equivalent.*

Termination of the Chase Tree

On the finiteness of C-chase trees We say that the *C-chase-tree halts* on $(F, \mathcal{R})$ when there exists a finite C-chase tree of $(F, \mathcal{R})$ (in that case, a breadth-first strategy for the rule applications will generate it). We can thus define *C-stable-finite* as the class of sets of nonmonotonic existential rules $\mathcal{R}$ for which the C-chase-tree halts on any $(F, \mathcal{R})$. Our first intuition was to assert "if $pos(\mathcal{R}) \in C$-finite, then $\mathcal{R} \in C$-stable-finite". However, this property is not true in general, as shown by the following example :

Example 13 *Let* $\mathcal{R} = \{R_1, R_2\}$ *where* $R_1 = h(x) \rightarrow p(x, y), h(y)$ *and* $R_2 = p(x, y), \mathbf{not}\, h(x) \rightarrow p(x, x)$*. See that* $pos(\mathcal{R}) \in$ *core-finite (as soon as* R_1 *is applied,* R_2 *is also applied and the loop* $p(x, x)$ *makes any other rule application redundant) ; however the only core-stable set of* $(\{h(a)\}, \mathcal{R})$ *is infinite (because all applications of* R_2 *are blocked).*

The following property shows that the desired property is true for *local* chases.

Proposition 10 *Let* $\mathcal{R}$ *be a set of NME rules and* C *be a local chase. If* $pos(\mathcal{R}) \in C$*-finite, then* $\mathcal{R} \in C$*-stable-finite.*

We have previously argued that the only two interesting chase variants w.r.t. the desired semantic properties are skolem and core. However, the core-finiteness of the positive part of a set of NME rules does not ensure the core-stable-finiteness of these rules. We should point out now that if $C \geq C'$, then C'-stable-finiteness implies C-stable-finiteness. We can thus ensure core-stable-finiteness when C-finiteness of the positive part of rules is ensured for a local C-chase.

Proposition 11 *Let* $\mathcal{R}$ *be a set of NME rules and* C *be a local chase. If* $pos(\mathcal{R}) \in C$*-finite, then* $\mathcal{R} \in$ *core-stable-finite.*

We can rely upon all acyclicity results in this paper to ensure that the core-chase tree halts.

Improving finiteness results with negative bodies We now explain how negation can be exploited to enhance preceding acyclicity notions. We first define the notion of *self-blocking rule*, which is a rule that will never be applied in any derivation. A rule $B^+, \mathbf{not}\, B^-_1, \ldots, \mathbf{not}\, B^-_k$ is self-blocking if there is a negative body B^-_i such that $B^-_i \subseteq (B^+ \cup H)$. Such a rule will never be applied in a sound way, so will never produce any atom. It follows that :

Proposition 12 *Let* $\mathcal{R}'$ *be the non-self-blocking rules of* $\mathcal{R}$*. If* $pos(\mathcal{R}') \in C$*-finite and* C *is local, then* $\mathcal{R} \in C$*-stable-finite.*

This idea can be further extended. We have seen for existential rules that if R' depends on R, then there is a unifier μ of $body(R')$ with $head(R)$, and we can build a rule $R'' = R \diamond_\mu R'$ that captures the sequence of applications encoded by the unifier. We extend Def. 5 to take into account negative bodies : if B^- is a negative body of R or R', then $\mu(B^-)$ is a negative body of R''. We also extend the notion of dependency in a natural way, and say that a unifier μ of $head(R)$ with $body(R')$ is self-blocking when $R \diamond_\mu R'$ is self-blocking, and R' *depends* on R when there exists a unifier of $head(R)$ with $body(R')$ that is not self-blocking. This extended notion of dependency exactly corresponds to the *positive reliance* in (Magka, Krötzsch, and Horrocks 2013).

Example 14 *Let* $R = q(x), \mathbf{not}\, p(x) \rightarrow r(x, y)$ *and* $R' = r(x, y) \rightarrow p(x), q(y)$*. Their associated positive rules are not core-finite. There is a single unifier* μ *of* R' *with* R*, and* $R \diamond_\mu R' : q(x), \mathbf{not}\, p(x) \rightarrow r(x, y), p(x), q(y)$ *is self-blocking. Then the skolem-chase-tree halts on* $(F, \{R, R'\})$ *for any* F*.*

Results obtained from positive rules can thus be generalized by considering this extended notion of dependency (for PG^U we only encode non self-blocking unifiers). Note that it does not change the complexity of the acyclicity tests.

We can further generalize this and check if a unifier sequence is self-blocking, thus extend the Y^{U+} classes to take into account negative bodies. Let us consider a compatible cycle C going through $\langle a, i \rangle$ that has not been proven safe. Let C_μ be the set of all compatible unifier sequences induced by C. We say that a sequence $\mu_1 \ldots \mu_k \in C_\mu$ is self-blocking when the rule $R_1 \diamond_{\mu_1} R_2 \ldots R_k \diamond_{\mu_k} R_{k+1}$ obtained by combining these unifiers is self-blocking. When all sequences in C_μ are self-blocking, we say that C is also self-blocking. This test comes again at no additional computational cost.

Example 15 *Let* $R_1 = q(x_1), \mathbf{not} p(x_1) \rightarrow r(x_1, y_1)$, $R_2 = r(x_2, y_2) \rightarrow s(x_2, y_2)$, $R_3 = s(x_3, y_3) \rightarrow p(x_3), q(y_3)$*.* $PG^{U+}(\{R_1, R_2, R_3\})$ *has a unique cycle, with a unique induced compatible unifier sequence. The rule* $R_1 \diamond R_2 \diamond R_3 = q(x_1), \mathbf{not} p(x_1) \rightarrow r(x_1, y_1), s(x_1, y_1), p(x_1), q(y_1)$ *is self-blocking, hence* $R_1 \diamond R_2 \diamond R_3 \diamond R_1$ *also is. Thus, there is no "dangerous" cycle.*

Proposition 13 *If, for each existential position* $\langle a, i \rangle$*, all compatible cycles for* $\langle a, i \rangle$ *in* PG^U *are self-blocking, then the stable computation based on the skolem chase halts.*

Conclusion

We have revisited chase termination with several results. First, a new tool that allows to unify and extend most existing acyclicity conditions, while keeping good computational properties. Second, a chase-like mechanism for nonmonotonic existential rules under stable model semantics, as well the extension of acyclicity conditions to take negation into account. This latter contribution extends the notion of negative reliance of (Magka, Krötzsch, and Horrocks 2013) ; and does not rely upon stratification (and thus does not enforce the existence of a single stable model).

This work will be pursued on the theoretical side by a complexity study of ENTAILMENT for the new acyclic classes and by a deeper study of logical foundations for NME rules, since it remains to relate our core-stable sets to an existing first-order semantics for general NME rules.

Acknowledgements

We thank the reviewers for their comments. This work is part of the ASPIQ and Pagoda projects and was partly funded by the french *Agence Nationale de la Recherche* (ANR) grants ANR-12-BS02-0003 and ANR-12-JS02-0007.

References

Baader, F. ; Brandt, S. ; and Lutz, C. 2005. Pushing the el envelope. In *IJCAI'05*, 364–369.

Baget, J.-F., and Mugnier, M.-L. 2002. The Complexity of Rules and Constraints. *J. Artif. Intell. Res. (JAIR)* 16 :425–465.

Baget, J.-F. ; Leclère, M. ; Mugnier, M.-L. ; and Salvat, E. 2009. Extending decidable cases for rules with existential variables. In *IJCAI'09*, 677–682.

Baget, J.-F. ; Leclère, M. ; Mugnier, M.-L. ; and Salvat, E. 2011. On rules with existential variables : Walking the decidability line. *Artificial Intelligence* 175(9-10) :1620–1654.

Baget, J.-F. 2004. Improving the forward chaining algorithm for conceptual graphs rules. In *KR'04*, 407–414. AAAI Press.

Beeri, C., and Vardi, M. 1981. The implication problem for data dependencies. In *ICALP'81*, volume 115 of *LNCS*, 73–85.

Calì, A. ; Gottlob, G. ; and Kifer, M. 2008. Taming the infinite chase : Query answering under expressive relational constraints. In *KR'08*, 70–80.

Calì, A. ; Gottlob, G. ; and Lukasiewicz, T. 2009a. A general datalog-based framework for tractable query answering over ontologies. In *PODS'09*, 77–86.

Calı, A. ; Gottlob, G. ; and Lukasiewicz, T. 2009b. Tractable query answering over ontologies with datalog±. In *Proceedings of the DL Home 22nd International Workshop on Description Logics (DL 2009)*.

Calimeri, F. ; Cozza, S. ; Ianni, G. ; and Leone, N. 2008. Computable functions in asp : Theory and implementation. In *Logic Programming*. Springer. 407–424.

Calvanese, D. ; Giacomo, G. D. ; Lembo, D. ; Lenzerini, M. ; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics : The DL-Lite family. *J. Autom. Reasoning* 39(3) :385–429.

Chandra, A. K. ; Lewis, H. R. ; and Makowsky, J. A. 1981. Embedded implicational dependencies and their inference problem. In *STOC'81*, 342–354. ACM.

Cuenca Grau, B. ; Horrocks, I. ; Krötzsch, M. ; Kupke, C. ; Magka, D. ; Motik, B. ; and Wang, Z. 2012. Acyclicity conditions and their application to query answering in description logics. In *KR*.

Cuenca Grau, B. ; Horrocks, I. ; Krötzsch, M. ; Kupke, C. ; Magka, D. ; Motik, B. ; and Wang, Z. 2013. Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research* 47 :741–808.

Dao-Tran, M. ; Eiter, T. ; Fink, M. ; Weidinger, G. ; and Weinzierl, A. 2012. Omiga : an open minded grounding on-the-fly answer set solver. In *Logics in Artificial Intelligence*. Springer. 480–483.

Deutsch, A. ; Nash, A. ; and Remmel, J. 2008. The chase revisited. In *PODS'08*, 149–158.

Fagin, R. ; Kolaitis, P. G. ; Miller, R. J. ; and Popa, L. 2003. Data exchange : Semantics and query answering. In *ICDT'03*, 207–224.

Fagin, R. ; Kolaitis, P. G. ; Miller, R. J. ; and Popa, L. 2005. Data exchange : semantics and query answering. *Theor. Comput. Sci.* 336(1) :89–124.

Ferraris, P. ; Lee, J. ; and Lifschitz, V. 2011. Stable models and circumscription. *Artif. Intell.* 175(1) :236–263.

Gelfond, M. 2007. *In Handbook of Knowledge Representation*. Elsevier Science. chapter Answer Sets.

Gottlob, G. ; Hernich, A. ; Kupke, C. ; and Lukasiewicz, T. 2012. Equality-friendly well-founded semantics and applications to description logics. In *Description Logics*.

Krötzsch, M., and Rudolph, S. 2011. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI'11*, 963–968.

Lefèvre, C., and Nicolas, P. 2009. A first order forward chaining approach for answer set computing. In *Logic Programming and Nonmonotonic Reasoning*. Springer. 196–208.

Lierler, Y., and Lifschitz, V. 2009. One more decidable class of finitely ground programs. In *Logic Programming*. Springer. 489–493.

Liu, L. ; Pontelli, E. ; Son, T. C. ; and Truszczyński, M. 2010. Logic programs with abstract constraint atoms : The role of computations. *Artificial Intelligence* 174(3–4) :295 – 315.

Magka, D. ; Krötzsch, M. ; and Horrocks, I. 2013. Computing stable models for nonmonotonic existential rules. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013). AAAI Press*.

Marnette, B. 2009. Generalized schema-mappings : from termination to tractability. In *PODS*, 13–22.

Mugnier, M.-L. 2011. Ontological Query Answering with Existential Rules. In *RR'11*, 2–23.

You, J.-H. ; Zhang, H. ; and Zhang, Y. 2013. Disjunctive logic programs with existential quantification in rule heads. *Theory and Practice of Logic Programming* 13 :563–578.

Causality in Databases: The Diagnosis and Repair Connections

Babak Salimi and Leopoldo Bertossi
Carleton University, School of Computer Science
Ottawa, Canada
{bsalimi, bertossi}@scs.carleton.ca

Abstract

In this work we establish and investigate the connections between causality for query answers in databases, database repairs wrt. denial constraints, and consistency-based diagnosis. The first two are relatively new problems in databases, and the third one is an established subject in knowledge representation. We show how to obtain database repairs from causes and the other way around. The vast body of research on database repairs can be applied to the newer problem of determining actual causes for query answers. By formulating a causality problem as a diagnosis problem, we manage to characterize causes in terms of a system's diagnoses.

1 Introduction

When querying a database, a user may not always obtain the expected results, and the system could provide some explanations. They could be useful to further understand the data or check if the query is the intended one. Actually, the notion of explanation for a query result was introduced in (Meliou et al. 2010a), on the basis of the deeper concept of *actual causation*.

Intuitively, a tuple t is a *cause* for an answer $\bar{a}$ to a conjunctive query $\mathcal{Q}$ from a relational database instance D if there is a "contingent" set of tuples Γ, such that, after removing Γ from D, removing/inserting t from/into D causes $\bar{a}$ to switch from being an answer to being a non-answer. Actual causes and contingent tuples are restricted to be among a pre-specified set of *endogenous tuples*, which are admissible, possible candidates for causes, as opposed to *exogenous tuples*.

Some causes may be stronger than others. In order to capture this observation, (Meliou et al. 2010a) also introduces and investigates a quantitative metric, called *responsibility*, which reflects the relative degree of causality of a tuple for a query result. In applications involving large data sets, it is crucial to rank potential causes by their responsibility (Meliou et al. 2010b; Meliou et al. 2010a).

Actual causation, as used in (Meliou et al. 2010a), can be traced back to (Halpern, and Pearl 2001; Halpern, and Pearl 2005), which provides a model-based account of causation on the basis of the *counterfactual dependence*. Responsibility was also introduced in (Chockler, and Halpern 2004), to capture the *degree of causation*.

Apart from the explicit use of causality, research on explanations for query results has focused mainly, and rather implicitly, on provenance (Buneman, Khanna, and Tan 2001; Buneman, and Tan 2007; Cheney, Chiticariu, and Tan 2009; Cui, Widom, and Wiener 2000; Karvounarakis, Ives, and Tannen 2010; Karvounarakis, and Green 2012; Tannen 2013), and more recently, on provenance for non-answers (Chapman, and Jagadish 2009; Huang et al. 2008).[1] A close connection between causality and provenance has been established (Meliou et al. 2010a). However, causality is a more refined notion that identifies causes for query results on the basis of user-defined criteria, and ranks causes according to their responsibility (Meliou et al. 2010b). For a formalization of non-causality-based explanations for query answers in DL ontologies, see (Borgida, Calvanese, and Rodriguez-Muro 2008).

Consistency-based diagnosis (Reiter 1987), a form of model-based diagnosis (Struss 2008, sec. 10.3), is an area of knowledge representation. The main task here is, given the *specification* of a system in some logical formalism and a usually unexpected *observation* about the system, to obtain *explanations* for the observation, in the form of a diagnosis for the unintended behavior.

In a different direction, a database instance, D, that is expected to satisfy certain integrity constraints (ICs) may fail to do so. In this case, a *repair* of D is a database D' that does satisfy the ICs and *minimally departs* from D. Different forms of minimality can be applied and investigated. A *consistent answer* to a query from D and wrt. the ICs is a query answer that is obtained from all possible repairs, i.e. is invariant or certain under the class of repairs. These notions were introduced in (Arenas, Bertossi, and Chomicki 1999) (see (Bertossi 2011) for a recent survey). We should mention that, although not in the framework of database repairs, consistency-based diagnosis techniques have been applied to restoring consistency of a database wrt. a set of ICs (Gertz 1997)

These three forms of reasoning, namely inferring causality in databases, consistency-based diagnosis, and consistent query answers (and repairs) are all *non-monotonic*. For ex-

[1] That is, tracing back, sometimes through the interplay of database tuple annotations, the reasons for *not* obtaining a possibly expected answer to a query.

ample, a (most responsible) cause for a query result may not be such anymore after the database is updated. In this work we establish natural, precise, useful, and deeper connections between these three reasoning tasks.

We show that inferring and computing actual causes and responsibility in a database setting become, in different forms, consistency-based diagnosis reasoning problems and tasks. Informally, a causal explanation for a conjunctive query answer can be viewed as a diagnosis, where in essence the first-order logical reconstruction of the relational database provides the system description (Reiter 1982), and the observation is the query answer. Furthermore, we unveil a strong connection between computing causes and their responsibilities for conjunctive queries, on the one hand, and computing *repairs* in databases (Bertossi 2011) wrt. denial constraints, on the other hand. These computational problems can be reduced to each other.

More precisely, our results are as follows:

1. For a boolean conjunctive query and its associated denial constraint (which is violated iff the query is true), we establish a precise connection between actual causes for the query (being true) and the subset-repairs of the instance wrt. the constraint. Namely, we obtain causes from repairs.
2. In particular, we establish the connection between an actual cause's responsibility and cardinality repairs wrt. the associated constraint.
3. We characterize and obtain subset- and cardinality- repairs for a database under a denial constraint in terms of the causes for the associated query being true.
4. We consider *a set* of denials constraints and a database that may be inconsistent wrt. them. We obtain the database repairs by means of an algorithm that takes as input the actual causes for constraint violations and their contingency sets.
5. We establish a precise connection between consistency-based diagnosis for a boolean conjunctive query being unexpectedly true according to a system description, and causes for the query being true. In particular, we can compute actual causes, their contingency sets, and responsibilities from minimal diagnosis.
6. Being this a report on ongoing work, we discuss several extensions and open issues that are under investigation.

2 Preliminaries

We will consider relational database schemas of the form $\mathcal{S} = (U, \mathcal{P})$, where U is the possibly infinite database domain and $\mathcal{P}$ is a finite set of database predicates of fixed arities. A database instance D compatible with $\mathcal{S}$ can be seen as a finite set of ground atomic formulas (in databases aka. atoms or tuples), of the form $P(c_1, ..., c_n)$, where $P \in \mathcal{P}$ has arity n, and $c_1, \ldots, c_n \in U$. A conjunctive query is a formula $\mathcal{Q}(\bar{x})$ of the first-order (FO) logic language, $\mathcal{L}(\mathcal{S})$, associated to $\mathcal{S}$ of the form $\exists \bar{y}(P_1(\bar{t}_1) \wedge \cdots \wedge P_m(\bar{t}_m))$, where the $P_i(\bar{t}_i)$ are atomic formulas, i.e. $P_i \in \mathcal{P}$, and the $\bar{t}_i$ are sequences of terms, i.e. variables or constants of U. The $\bar{x}$ in $\mathcal{Q}(\bar{x})$ shows all the free variables in the formula, i.e. those not appearing in $\bar{y}$. The query is boolean, if $\bar{x}$ is empty, i.e. the query is a sentence, in which case, it is true or false in a database, denoted by $D \models \mathcal{Q}$ and $D \not\models \mathcal{Q}$, respectively. A sequence $\bar{c}$ of constants is an answer to an open query $\mathcal{Q}(\bar{x})$ if $D \models \mathcal{Q}[\bar{c}]$, i.e. the query becomes true in D when the variables are replaced by the corresponding constants in $\bar{c}$.

An integrity constraint is a sentence of language $\mathcal{L}(\mathcal{S})$, and then, may be true or false in an instance for schema $\mathcal{S}$. Given a set *IC* of ICs, a database instance D is *consistent* if $D \models IC$; otherwise it is said to be *inconsistent*. In this work we assume that sets of ICs are always finite and logically consistent. A particular class of integrity constraints (ICs) is formed by *denial constraints* (DCs), which are sentences κ of the from: $\forall \bar{x} \neg(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n))$, where $\bar{x} = \bigcup \bar{x}_i$ and each $A_i(\bar{x}_i)$ is a database atom, i.e. predicate $A \in \mathcal{P}$. DCs will receive special attention in this work. They are common and natural in database applications since they disallow combinations of database atoms.

Causality and Responsibility. Assume that the database instance is split in two, i.e. $D = D^n \cup D^x$, where D^n and D^x denote the sets of *endogenous* and *exogenous* tuples, respectively. A tuple $t \in D^n$ is called a *counterfactual cause* for a boolean conjunctive $\mathcal{Q}$, if $D \models \mathcal{Q}$ and $D \setminus \{t\} \not\models \mathcal{Q}$. A tuple $t \in D^n$ is an *actual cause* for $\mathcal{Q}$ if there exists $\Gamma \subseteq D^n$, called a *contingency set*, such that t is a counterfactual cause for $\mathcal{Q}$ in $D \setminus \Gamma$ (Meliou et al. 2010a).

The *responsibility* of an actual cause t for $\mathcal{Q}$, denoted by $\rho(t)$, is the numerical value $\frac{1}{(|\Gamma|+1)}$, where $|\Gamma|$ is the size of the smallest contingency set for t. We can extend responsibility to all the other tuples in D^n by setting their value to 0. Those tuples are not actual causes for $\mathcal{Q}$.

In (Meliou et al. 2010a), causality for non-query answers is defined on basis of sets of *potentially missing tuples* that account for the missing answer. Computing actual causes and their responsibilities for non-answers becomes a rather simple variation of causes for answers. In this work we focus on causality for query answers.

Example 1. Consider a database D with relations R and S as below, and the query $\mathcal{Q} : \exists x \exists y(S(x) \wedge R(x, y) \wedge S(y))$. $D \models \mathcal{Q}$ and we want to find causes for $\mathcal{Q}$ being true in D under the assumption that all tuples are endogenous.

R	X	Y
	a_4	a_3
	a_2	a_1
	a_3	a_3

S	X
	a_4
	a_2
	a_3

Tuple $S(a_3)$ is a counterfactual cause for $\mathcal{Q}$. If $S(a_3)$ is removed from D, we reach a state where $\mathcal{Q}$ is no longer an answer. Therefore, the responsibility of $S(a_3)$ is 1. Besides, $R(a_4, a_3)$ is an actual cause for $\mathcal{Q}$ with contingency set $\{R(a_3, a_3)\}$. If $R(a_3, a_3)$ is removed from D, we reach a state where $\mathcal{Q}$ is still an answer, but further removing $R(a_4, a_3)$ makes $\mathcal{Q}$ a non-answer. The responsibility of $R(a_4, a_3)$ is $\frac{1}{2}$, because its smallest contingency sets have size 1. Likewise, $R(a_3, a_3)$ and $S(a_4)$ are actual causes for

$\mathcal{Q}$ with responsibility $\frac{1}{2}$. □

Now we can show that counterfactual causality for query answers is a non-monotonic notion.

Example 2. (ex. 1 cont.) Consider the same query $\mathcal{Q}$, but now the database instance $D = \{S(a_3), S(a_4), R(a_4, a_3)\}$, with the partition $D^n = \{S(a_4), S(a_3)\}$ and $D^x = \{R(a_4, a_3)\}$. Both $S(a_3)$ and $S(a_4)$ are counterfactual causes for $\mathcal{Q}$.

Now assume $R(a_3, a_3)$ is added to D as an exogenous tuple, i.e. $(D^x)' = \{R(a_4, a_3),\ R(a_3, a_3)\}$. Then, $S(a_4)$ is no longer a counterfactual cause for $\mathcal{Q}$ in $D' = D^n \cup (D^x)'$: If $S(a_4)$ is removed from the database, $\mathcal{Q}$ is still true in D'. Moreover, $S(a_4)$ not an actual cause anymore, because there is no contingency set that makes $S(a_4)$ a counterfactual cause.

Notice that, if $R(a_3, a_3)$ is instead inserted as an endogenous tuple, i.e. $(D^n)' = \{S(a_4), S(a_3), R(a_3, a_3)\}$, then, $S(a_4)$ is still an actual cause for $\mathcal{Q}$, with contingency set $\{R(a_3, a_3)\}$. □

The following proposition shows that the notion of actual causation is non-monotone in general.

Notation: $\mathcal{CS}(D^n, D^x, \mathcal{Q})$ denotes the set of actual causes for BCQ $\mathcal{Q}$ (being true) from instance $D = D^n \cup D^x$. When $D^n = D$ and $D^x = \emptyset$, we sometimes simply write: $\mathcal{CS}(D, \mathcal{Q})$.

Proposition 1. Let $(D^n)', (D^x)'$ denote updates of instances D^n, D^x by insertion of tuple t, resp. It holds: (a) $\mathcal{CS}(D^n, D^x,\ \mathcal{Q}) \subseteq \mathcal{CS}((D^n)', D^x, \mathcal{Q})$. (b) $\mathcal{CS}(D^n, (D^x)',\ \mathcal{Q}) \subseteq \mathcal{CS}(D^n, D^x, \mathcal{Q})$. □

Example 2 shows that the inclusion in (b) may be strict. It is easy to show that it can also be strict for (a). This result tells us that, for a fixed query, inserting an endogenous tuples may extend the set of actual cases, but it may shrink by inserting an endogenous tuple. It is also easy to verify that most responsible causes may not be such anymore after the insertion of endogenous tuples.

Database Repairs. Given a set *IC* of ICs, a *subset-repair* (simply, S-repair) of a possibly inconsistent instance D for schema $\mathcal{S}$ is an instance D' for $\mathcal{S}$ that satisfies *IC* and makes $\Delta(D, D') = (D \smallsetminus D') \cup (D' \smallsetminus D)$ minimal under set inclusion. $\mathit{Srep}(D, \mathit{IC})$ denotes the set of S-repairs of D wrt. *IC* (Arenas, Bertossi, and Chomicki 1999). $\bar{c}$ is a *consistent answer* to query $\mathcal{Q}(\bar{x})$ if $D' \models \mathcal{Q}[\bar{c}]$ for every $D' \in \mathit{Srep}$, denoted $D \models_S \mathcal{Q}[\bar{c}]$. S-repairs and consistent query answers for DCs were investigated in detail (Chomicki, and Marcinkowski 2005). (Cf. (Bertossi 2011) for more references.)

Similarly, D' is a *cardinality repair* (simply C-repair) of D if D' satisfies *IC* and minimizes $|\Delta(D, D')|$. $\mathit{Crep}(D, \mathit{IC})$ denotes the class of C-repairs of D wrt. *IC*. That $\bar{c}$ is a consistent answer to $\mathcal{Q}(\bar{x})$ wrt. C-repairs is denoted by $D \models_C \mathcal{Q}[\bar{c}]$. C-repairs were investigated in detail in (Lopatenko, and Bertossi 2007).

C-repairs are S-repairs of minimum cardinality, and, for DCs, they are obtained from the original instance by deleting a cardinality-minimum or a subset-minimal set of tuples, respectively. Obtaining repairs and consistent answers is a non-monotonic process. That is, after an update of D to $u(D)$, obtained by tuple insertions, a repair or a consistent answer for D may not be such for $u(D)$ (Bertossi 2011).

Consistency-Based Diagnosis. The starting point of this consistency-based approach to diagnosis is a diagnosis problem of the form $\mathcal{M} = (SD, COMPS, OBS)$, where SD is the description in logic of the intended properties of a system under the *explicit* assumption that all its components, those in the set of constants ***COMPS***, are normal (or working normally). OBS is a finite set of FO sentences (usually a conjunction of ground literals) that represents the observations.

Now, if the system does not behave as expected (as shown by the observations), then the logical theory obtained from $SD \cup OBS$ plus the explicit assumption, say $\bigwedge_{c \in COMPS} \neg ab(c)$, that the components are indeed behaving normally, becomes inconsistent.[2] This inconsistency is captured via the *minimal conflict sets*, i.e. those minimal subsets $COMPS_0$ of ***COMPS***, such that $SD \cup OBS \cup \{\bigwedge_{c \in COMPS_0} \neg ab(c)\}$ is still inconsistent. As expected, different notions of minimality can be used at this point. It is common to use the distinguished predicate $ab(\cdot)$ for denoting *abnormal* (or abnormality). So, $ab(c)$ says that component c is abnormal.

On this basis, a *minimal diagnosis* for $\mathcal{M}$ is a minimal subset Δ of $COMPS$, such that $SD \cup OBS \cup \{\neg ab(c) \mid c \in COMPS \smallsetminus \Delta\} \cup \{ab(c) \mid c \in \Delta\}$ is consistent. That is, consistency is restored by flipping the normality assumption to abnormality for a minimal set of components, and those are the ones considered to be (jointly) faulty. The notion of minimality commonly used is subset-minimality, i.e. a minimal diagnosis must not have a proper subset that is still a diagnosis. We will use this kind of minimality in relation to diagnosis. Diagnosis can be obtained from conflict sets (Reiter 1987). See also (Struss 2008, sec. 10.4) for a broader review of model-based diagnosis.

Diagnostic reasoning is non-monotonic in the sense that a diagnosis may not survive after the addition of new observations (Reiter 1987).

3 Repairs and Causality for Query Answers

Let $D = D^n \cup D^x$ be a database instance for schema $\mathcal{S}$, and $\mathcal{Q} : \exists \bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$ be a boolean conjunctive query (BCQ). Suppose $\mathcal{Q}$ is unexpectedly true in D. Actually, it is expected that $D \not\models \mathcal{Q}$, or equivalently, that $D \models \neg\mathcal{Q}$. Now, $\neg\mathcal{Q}$ is logically equivalent to a formula of the form $\kappa(\mathcal{Q}) : \forall \bar{x} \neg(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, which has the form of a denial constraint. The requirement that $\neg\mathcal{Q}$ holds can be captured by imposing the corresponding DC $\kappa(\mathcal{Q})$ to D.

[2] Here, and as usual, the atom $ab(c)$ expresses that component c is (behaving) abnormal(ly).

Since $D \models \mathcal{Q}$, D is inconsistent wrt. the DC $\kappa(\mathcal{Q})$. Now, repairs for (violations of) DCs are obtained by tuple deletions. Intuitively, tuples that account for violations of $\kappa(\mathcal{Q})$ in D are actual causes for $\mathcal{Q}$. Minimal sets of tuples like this are expected to correspond to S-repairs for D and $\kappa(\mathcal{Q})$. Next we make all this precise.

Given an instance $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and a tuple $t \in D$, we consider the class containing the sets of differences between D and those S-repairs that do not contain tuple $t \in D^n$, and are obtained by removing a subset of D^n:

$$\mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t) \;=\; \{D \smallsetminus D' \mid D' \in \mathit{Srep}(D, \kappa(\mathcal{Q})), \\ t \in (D \smallsetminus D') \subseteq D^n\}.$$

Now, $s \in \mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t)$ can written as $s = s' \cup \{t\}$. From the definition of a S-repair, including its S-minimality, $D \smallsetminus (s' \cup \{t\}) \models \kappa(\mathcal{Q})$, but $D \smallsetminus s' \models \neg\kappa(\mathcal{Q})$, i.e. $D \smallsetminus (s' \cup \{t\}) \not\models \mathcal{Q}$, but $D \smallsetminus s' \models \mathcal{Q}$. So, we obtain that t is an actual cause for $\mathcal{Q}$ with contingency set s'. The following proposition formalizes this result.

Proposition 2. Given an instance $D = D^n \cup D^x$, and a BCQ $\mathcal{Q}$, $t \in D^n$ is an actual cause for $\mathcal{Q}$ iff $\mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t) \neq \emptyset$. □

The next proposition shows that the responsibility of a tuple can also be determined from $\mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t)$.

Proposition 3. Given an instance $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and $t \in D^n$,

1. If $\mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t) = \emptyset$, then $\rho(t) = 0$.
2. Otherwise, $\rho(t) = \frac{1}{|s|}$, where $s \in \mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t)$ and there is no $s' \in \mathcal{DF}(D, D^n, \kappa(\mathcal{Q}), t)$ such that, $|s'| < |s|$. □

Example 3. (ex. 1 cont.) Consider the same instance D and query $\mathcal{Q}$. In this case, the DC $\kappa(\mathcal{Q})$ is, in Datalog notation as a negative rule: $\leftarrow S(x), R(x, y), S(y)$.

Here, $\mathit{Srep}(D, \kappa(\mathcal{Q})) = \{D_1, D_2, D_3\}$ and $\mathit{Crep}(D, \kappa(\mathcal{Q})) = \{D_1\}$, with $D_1 = \{R(a_4, a_3), R(a_2, a_1), R(a_3, a_3), S(a_4), S(a_2)\}$, $D_2 = \{R(a_2, a_1), S(a_4), S(a_2), S(a_3)\}$, $D_3 = \{R(a_4, a_3), R(a_2, a_1), S(a_2), S(a_3)\}$.

For tuple $R(a_4, a_3)$, $\mathcal{DF}(D, D, \kappa(\mathcal{Q}), R(a_4, a_3)) = \{D \smallsetminus D_2\} = \{\{R(a_4, a_3), R(a_3, a_3)\}\}$. This, together with Propositions 2 and 3, confirms that $R(a_4, a_3)$ is an actual cause, with responsibility $\frac{1}{2}$.

For tuple $S(a_3)$, $\mathcal{DF}(D, D, \kappa(\mathcal{Q}), S(a_3)) = \{D \smallsetminus D_1\} = \{S(a_3)\}$. So, $S(a_3)$ is an actual cause with responsibility 1. Similarly, $R(a_3, a_3)$ is an actual cause with responsibility $\frac{1}{2}$, because $\mathcal{DF}(D, D, \kappa(\mathcal{Q}), R(a_3, a_3)) = \{D \smallsetminus D_2, D \smallsetminus D_3\} = \{\{R(a_4, a_3), R(a_3, a_3)\}, \{R(a_3, a_3), S(a_4)\}\}$.

It is easy to verify that $\mathcal{DF}(D, D, \kappa(\mathcal{Q}), S(a_2))$ and $\mathcal{DF}(D, D, \kappa(\mathcal{Q}), R(a_2, a_1))$ are empty, because all repairs contain those tuples. This means that they do not participate in the violation of $\kappa(\mathcal{Q})$, or equivalently, they do not contribute to make $\mathcal{Q}$ true. So, $S(a_2)$ and $R(a_2, a_1)$ are not actual causes for $\mathcal{Q}$, confirming the result in Example 1. □

Now, we reduce computation of repairs for inconsistent databases wrt. a denial constraint to corresponding problems for causality.

Consider the database instance D for schema $\mathcal{S}$ and a denial constraint $\kappa : \leftarrow A_1(\bar{x}_1), \ldots, A_n(\bar{x}_n)$, to which a boolean conjunct ive *violation view* $V^\kappa : \exists\bar{x}(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n))$ can be associated: D violates (is inconsistent wrt.) κ iff $D \models V^\kappa$.

Intuitively, actual causes for V^κ, together with their contingency sets, account for violations of κ by D. Removing those tuples from D should remove the inconsistency.

Given an inconsistent instance D wrt. κ, we collect all S-minimal contingency sets associated with the actual cause t for V^κ, as follows:

$$\mathcal{CT}(D, D^n, V^\kappa, t) \;=\; \{s \subseteq D^n \mid D \smallsetminus s \models V^\kappa, \\ D \smallsetminus (s \cup \{t\}) \not\models V^\kappa, \text{ and} \\ \forall s'' \subsetneq s,\ D \smallsetminus (s'' \cup \{t\}) \models V^\kappa\}.$$

Notice that for sets $s \in \mathcal{CT}(D, D^n, V^\kappa, t)$, $t \notin s$. Now consider, $t \in \mathcal{CS}(D, \emptyset, V^\kappa)$, the set of actual causes for V^κ when the entire database is endogenous. From the definition of an actual cause and the S-minimality of sets $s \in \mathcal{CT}(D, D, V^\kappa, t)$, $s'' = s \cup \{t\}$ is an S-minimal set such that $D \smallsetminus s'' \not\models V^\kappa$. So, $D \smallsetminus s''$ is an S-repair for D. We obtain:

Proposition 4. (a) Given an instance D and a DC κ, D is consistent wrt. κ iff $\mathcal{CS}(D, \emptyset, V^\kappa) = \emptyset$. (b) $D' \subseteq D$ is an S-repair for D iff, for every $t \in D \smallsetminus D'$, $t \in \mathcal{CS}(D, \emptyset, V^\kappa)$ and $D \smallsetminus (D' \cup \{t\}) \in \mathcal{CT}(D, D, V^\kappa, t)$. □

Now we establish a connection between most responsible actual causes and C-repairs. For this, we collect the most responsible actual causes for V^κ:

$$\mathcal{MRC}(D, V^\kappa) \;=\; \{t \in D \mid t \in \mathcal{CS}(D, \emptyset, V^\kappa), \\ \nexists t' \in \mathcal{CS}(D, \emptyset, V^\kappa) \text{ with } \rho(t') > \rho(t)\}.$$

Proposition 5. For an instance D and denial constraint κ, D' is a C-repair for D wrt. κ iff for a $t \in D \smallsetminus D'$: $t \in \mathcal{MRC}(D, V^\kappa)$ and $D \smallsetminus (D' \cup \{t\}) \in \mathcal{CT}(D, V^\kappa, t)$. □

Example 4. Consider $D = \{P(a, b), R(b, c), R(b, b)\}$, and the denial constraint $\kappa : \leftarrow P(x, y), R(y, z)$, which prohibits a join between P and R. The corresponding violation view (query) is, $V^\kappa : \exists xyz(P(x, y) \wedge R(y, z))$. Since $D \models V^\kappa$, D is inconsistent wrt. κ.

Here, $\mathcal{CS}(D, \emptyset, V^\kappa) = \{P(a, b), R(b, c), R(b, b)\}$, each of whose members is associated with S-minimal contingency sets: $\mathcal{CT}(D, D, V^\kappa, R(b, c)) = \{\{R(b, b)\}\}$, $\mathcal{CT}(D, D, V^\kappa, R(b, b)) = \{\{R(b, c)\}\}$, and $\mathcal{CT}(D, D, V^\kappa, P(a, b)) = \{\emptyset\}$.

According to Proposition 4, the instance obtained by removing each actual cause for V^κ together with its contingency set forms a S-repair for D. Therefore, $D_1 = D \smallsetminus \{P(a, b)\} = \{R(b, c), R(b, b)\}$ is an S-repair. Notice that the S-minimal contingency set associated to $P(a, b)$ is an empty set. Likewise, $D_2 = D \smallsetminus \{R(b, c), R(b, b)\} =$

$\{P(a,b)\}$ is a S-repair. It is easy to verify that D does not have any S-repair other than D_1 and D_2.

Furthermore, $\mathcal{MRC}(D, V^\kappa) = \{P(a,b)\}$. So, according to Proposition 5, D_1 is also a C-repair for D. □

Given an instance D, a DC κ and a ground atomic query A, the following proposition establishes the relationship between consistent query answers to A wrt. the S-repair semantics and actual cases for the violation view V^κ.

Proposition 6. A ground atomic query A, is consistently true, i.e. $D \models_S A$, iff $A \in D \smallsetminus \mathcal{CS}(D, \emptyset, V^\kappa)$. □

Example 5. Consider $D = \{P(a,b), R(b,c), R(a,d)\}$, the DC $\kappa : \leftarrow P(x,y), R(y,z)$, and the ground atomic query $\mathcal{Q} : R(a,d)$. It is easy to see that $\mathcal{CS}(D, \emptyset, V^\kappa) = \{P(a,b), R(b,c)\}$. Then, according to Proposition 6, $R(a,d)$ is consistently true in D, because $D \smallsetminus \mathcal{CS}(D, \emptyset, V^\kappa) = \{R(a,d)\}$. □

4 Causes for IC violations

We may consider a set Σ of ICs ψ that have violation views V^ψ that become boolean conjunctive queries, e.g. denial constraints. Each of such views has the form $V^\psi : \exists\bar{x}(A_1(\bar{x}_1) \wedge \cdots \wedge A_n(\bar{x}_n))$. When the instance D is inconsistent wrt. Σ, some of these views (queries) get the answer *yes* (they become true), and for each of them there is a set $\mathcal{C}(D, D^n, V^\psi)$ whose elements are of the form $\langle t, \{C_1(t), \ldots, C_m(t)\}\rangle$, where t is a tuple that is an actual cause for V^ψ, together with their contingency sets $C_i(t)$, possibly minimal in some sense. The natural question is whether we can obtain repairs of D wrt. Σ from the sets $\mathcal{C}(D, D^n, V^\psi)$.

In the following we consider the case where $D^n = D$, i.e. we consider the sets $\mathcal{C}(D, D, V^\psi)$, simply denoted $\mathcal{C}(D, V^\psi)$. We recall that $\mathcal{CS}(D, V^\psi)$ denotes the set of actual causes for V^ψ. We denote with $\mathcal{CT}(D, V^\psi, t)$ the set of all subset-minimal contingency sets associated with the actual cause t for V^ψ.

The (naive) Algorithm *SubsetRepairs* that we describe in high-level term in the following accepts as input an instance D, a set of DCs Σ, and the sets $\mathcal{C}(D, V^\psi)$, each of them with elements of the form $\langle t, \{C_1(t), \ldots, C_m(t)\}\rangle$ where each $C_i(t)$ is subset-minimal. The output of the algorithm is $Srep(D, \Sigma)$, the set of S-repairs for D.

The idea of the algorithm is as follows. For each V^ψ, $D \smallsetminus (\{t\} \cup C(t))$ where, $t \in \mathcal{CS}(D, V^\psi)$ and $C(t) \in \mathcal{CT}(D, V^\psi, t)$, is consistent with ψ since, according to the definition of an actual cause, $D \smallsetminus (\{t\} \cup C(t)) \not\models V_\psi$.

Therefore, $D' = D \smallsetminus \bigcup_{\psi\in\Sigma}\{\{t\} \cup C(t) \mid t \in \mathcal{CS}(D, V^\psi)$ and $C(t) \in \mathcal{CT}(D, V^\psi, t)\}$ is consistent with Σ. However, it may not be an S-repair, because some violation views may have common causes.

In order to obtain S-repairs, the algorithm finds common causes for the violation views, and avoids removing redundant tuples to resolve inconsistencies. In this direction, the algorithm forms a set collecting all the actual causes for violation views: $S = \{t \mid \exists\psi \in \Sigma, t \in \mathcal{CS}(D, V^\psi)\}$. It also builds the collection of non-empty sets of actual causes for each violation view: $\mathcal{C} = \{\mathcal{CS}(D, V^\psi) \mid \exists\psi \in \Sigma, \mathcal{CS}(D, V^\psi) \neq \emptyset\}$. Clearly, $\mathcal{C}$ is a collection of subsets of set S.

Next, the algorithm computes the set of all subset-minimal *hitting sets* of the collection C.[3] Intuitively, an S-minimal hitting set of C contains an S-minimal set of actual causes that covers all violation views, i.e. each violation view has an actual cause in the hitting set. The algorithm collects all S-minimal hitting sets of C in $\mathcal{H}$.

Now, for a hitting set $h \in \mathcal{H}$, for each $t \in h$, if t covers V_ψ, the algorithm removes both t and $C(t)$ from D (where $C(t) \in \mathcal{CT}(D, V^\psi, t)$). Since it may happen that a violation view is covered by more than one element in h, the algorithm makes sure that just one of them is chosen. The result is an S-repair for D. The algorithm repeats this procedure for all sets in $\mathcal{H}$. The result is $Srep(D, \Sigma)$.

Example 6. Consider the instance $D = \{P(a,b), R(b,c), S(c,d)\}$, and the set of DCs $\Sigma = \{\psi_1, \psi_2\}$, with $\psi_1 : \leftarrow P(x,y), R(y,z)$, and $\psi_2 : \leftarrow R(x,y), S(y,z)$. The corresponding violation views are $V^{\psi_1} : \exists xyz(P(x,y) \wedge R(y,z))$, and $V^{\psi_2} : \exists xyz(R(x,y) \wedge S(y,z))$.

Here, $\mathcal{C}(D, V^{\psi_1}) = \{\langle P(a,b), \{\emptyset\}\rangle, \langle R(b,c), \{\emptyset\}\rangle\}$, and $\mathcal{C}(D, V^{\psi_2}) = \{\langle R(b,c), \{\emptyset\}\rangle, \langle S(c,d), \{\emptyset\}\rangle\}$.

The set S in the algorithm above, actual causes for ψ_1 or ψ_2, is $S = \{P(a,b), R(b,c), S(c,d)\}$. The collection C, of sets of actual causes for ψ_1 and ψ_2, is $C = \{\{P(a,b), R(b,c)\}, \{R(b,c), S(c,d)\}\}$.

The subset-minimal hitting sets for the collection C are: $h_1 = \{R(b,c)\}$, $h_2 = \{S(c,d), P(a,b)\}$. Since the contingency set for each of the actual causes is empty, $D \smallsetminus h_1$ and $D \smallsetminus h_2$ are the S-repairs for D. □

The following theorem states that algorithm *SubsetRepairs* provides a sound and complete method for computing $Srep(D, \Sigma)$.

Theorem 1. Given an instance D, a set Σ of DCs, and the sets $\mathcal{C}(D, V^\psi)$, for $\psi \in \Sigma$, *SubsetRepairs* computes exactly $Srep(D, \Sigma)$. □

The connection between causality and databases repair provides this opportunity to apply results and techniques developed in each context to the other one. In particular, in our future works we will use this connection to provide some complexity results in the context of consistent query answering.

5 Diagnosis and Query Answer Causality

As before, let $D = D^n \cup D^x$ be a database instance for schema $\mathcal{S}$, and $\mathcal{Q} : \exists\bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$ be BCQ. Assume that $\mathcal{Q}$ is, possibly unexpectedly, true in D. Also as above, the associated DC is $\kappa(\mathcal{Q}) : \forall\bar{x}\neg(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$. So, it holds $D \not\models \kappa(\mathcal{Q})$, i.e. D violates the DC.

[3] A set $S' \subseteq S$ is a hitting set for $\mathcal{C}$ if, for every $C_i \in \mathcal{C}$, there is a $c \in C_i$ with $c \in S'$. A hitting set is subset-minimal if no proper subset of it is also a hitting set.

This is our observation, and we want to find causes for it, using a diagnosis-based approach. Those causes will become causes for $\mathcal{Q}$ being true; and the diagnosis will uniquely determine those causes.

In this direction, for each predicate $P \in \mathcal{P}$, we introduce predicate ab_P, with the same arity as P. Any tuple in its extension is said to be *abnormal* for P. Our "system description", SD, for a diagnosis problem will include, among other elements, the original database, expressed in logical terms, and the DC being true "under normal conditions".

More precisely, we consider the following *diagnosis problem*, $\mathcal{M} = (SD, D^n, \mathcal{Q})$, associated to $\mathcal{Q}$. Here, SD is the FO system description that contains the following elements: (a) $Th(D)$, which is Reiter's logical reconstruction of D as a FO theory (Reiter 1982). (b) Sentence $\kappa(\mathcal{Q})^{ext}$, which is $\kappa(\mathcal{Q})$ rewritten as follows:

$$\kappa(\mathcal{Q})^{ext} : \forall \bar{x} \neg (P_1(\bar{x}_1) \wedge \neg ab_{P_1}(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \neg ab_{P_m}(\bar{x}_m)). \quad (1)$$

(c) The sentence $\neg\kappa(\mathcal{Q}) \longleftrightarrow \mathcal{Q}$, where $\mathcal{Q}$ is the initial boolean query. (d) The inclusion dependencies: $\forall \bar{x}(ab_P(\bar{x}) \rightarrow P(\bar{x}))$.

Now, the last entry in $\mathcal{M}$, $\mathcal{Q}$, is the *observation*, which together with *SD* will produce (see below) and inconsistent theory. This is because in $\mathcal{M}$ we make the initial and explicit assumption that all the abnormality predicates are empty (equivalently, that all tuples are normal), i.e. we consider, for each predicate P, the sentence

$$\forall \bar{x}(ab_P(\bar{x}) \rightarrow \textbf{false}), \quad (2)$$

where, **false** is a propositional atom that is always false. Actually, the second entry in $\mathcal{M}$ tells us how we can restore consistency, namely by (minimally) changing the abnormality condition of tuples in D^n. In other words, the rules (2) are subject to qualifications: some endogenous tuples may be abnormal. Each diagnosis for the diagnosis problem shows a subset-minimal set of endogenous tuples that are abnormal.

Example 7. (ex. 2 cont.) For the instance $D = \{S(a_3), S(a_4), R(a_4, a_3)\}$, with $D^n = \{S(a_4), S(a_3)\}$, consider the diagnostic problem $\mathcal{M} = (SD, \{S(a_4), S(a_3)\}, \mathcal{Q})$, where SD contains the following sentences:

(a) Predicate completion axioms:
$\forall xy(R(x,y) \leftrightarrow x = a_4 \wedge y = a_3)$,
$\forall x(S(x) \leftrightarrow x = a_3 \vee x = a_4)$.

(a) Unique names assumption: $a_4 \neq a_3$.

(b) $\kappa(\mathcal{Q})^{ext} : \forall xy \neg(S(x) \wedge \neg ab_S(x) \wedge R(x,y) \wedge \neg ab_R(x,y) \wedge S(y) \wedge \neg ab_S(y))$.

(c) $\neg\kappa(\mathcal{Q}) \longleftrightarrow \mathcal{Q}$ (with $\kappa(\mathcal{Q})$ and $\mathcal{Q}$ as before).

(d) $\forall xy(ab_R(x,y) \rightarrow R(x,y))$, $\forall x(ab_S(x) \rightarrow S(x))$.

The explicit assumption about the normality of all tuples is captured by:
$\forall xy(ab_R(x,y) \rightarrow \textbf{false})$, $\forall x(ab_S(x) \rightarrow \textbf{false})$. □

Now, the observation is $\mathcal{Q}$ (is true), obtained by evaluating query $\mathcal{Q}$ on (theory of) D. In this case, $D \not\models \kappa(\mathcal{Q})$. Since all the abnormality predicates are assumed to be empty, $\kappa(\mathcal{Q})$ is equivalent to $\kappa(\mathcal{Q})^{ext}$, which also becomes false wrt D. As a consequence, $SD \cup \{(2)\} \cup \{\mathcal{Q}\}$ is an inconsistent FO theory. Now, a diagnosis is a set of endogenous tuples that, by becoming abnormal, restore consistency.

Definition 1. (a) A *diagnosis* for a diagnosis problem $\mathcal{M}$ is a $\Delta \subseteq D^n$, such that $SD \cup \{ab_P(\bar{c}) \mid P(\bar{c}) \in \Delta\} \cup \{\neg ab_P(\bar{c}) \mid P(\bar{c}) \in D \setminus \Delta\} \cup \{\mathcal{Q}\}$ becomes consistent. (b) $\mathcal{D}(\mathcal{M}, t)$ denotes the set of subset-minimal diagnoses for $\mathcal{M}$ that contain a tuple $t \in D^n$. (c) $\mathcal{MCD}(\mathcal{M}, t)$ denotes the set of diagnoses of $\mathcal{M}$ that contain a tuple $t \in D^n$ and have the minimum cardinality (among those diagnoses that contain t). □

Clearly, $\mathcal{MCD}(\mathcal{M}, t) \subseteq \mathcal{D}(\mathcal{M}, t)$. The following proposition specifies the relationship between minimal diagnoses for $\mathcal{M}$ and actual causes for $\mathcal{Q}$.

Proposition 7. Consider $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, and the diagnosis problem $\mathcal{M}$ associated to $\mathcal{Q}$. Tuple $t \in D^n$ is an actual cause for $\mathcal{Q}$ iff $\mathcal{D}(\mathcal{M}, t) \neq \emptyset$. □

The next proposition tells us that the responsibility of an actual cause t is determined by the cardinality of the diagnoses in $\mathcal{MCD}(\mathcal{M}, t)$.

Proposition 8. Consider $D = D^n \cup D^x$, a BCQ $\mathcal{Q}$, the diagnosis problem $\mathcal{M}$ associated to $\mathcal{Q}$, and a tuple $t \in D^n$.
(a) $\rho(t) = 0$ iff $\mathcal{MCD}(\mathcal{M}, t) = \emptyset$.
(b) Otherwise, $\rho(t) = \frac{1}{|s|}$, where $s \in \mathcal{MCD}(\mathcal{M}, t)$. □

Example 8. (ex. 7 cont.) The diagnosis problem $\mathcal{M}$ has two diagnosis namely, $\Delta_1 = \{S(a_3)\}$ and $\Delta_4 = \{S(a_4)\}$.

Here, $\mathcal{D}(\mathcal{M}, S(a_3)) = \mathcal{MCD}(\mathcal{M}, S(a_3)) = \{\{S(a_3)\}\}$ and $\mathcal{D}(\mathcal{M}, S(a_4)) = \mathcal{MCD}(\mathcal{M}, S(a_4)) = \{\{ S(a_4)\}\}$. Therefore, according to Proposition 7 and 8, both $S(a_3)$ and $S(a_4)$ are actual cases for $\mathcal{Q}$, with responsibility 1. □

Notice that the consistency-based approach to causality provided in this section can be considered as a technique for computing repairs for inconsistent databases wrt. denial constraints (it is a corollary of 4 and 8). It is worth mentioning that this approach has been implicitly used before in databases repairing in (Arenas et al. 2003), where the authors introduce *conflict graphs* to characterize S-repairs for inconsistent databases wrt. FDs. We will use this connection in our future work to provide some complexity results in the context of causality.

6 Discussion

Here we discuss some directions of possible or ongoing research.

Open queries. We have limited our discussion to boolean queries. It is possible to extend our work to consider conjunctive queries with free variables, e.g. $\mathcal{Q}(x) : \exists yz(R(x,y) \wedge S(y,z))$. In this case, a query answer would be of the form $\langle a \rangle$, for a a constant, and causes would be found for such an answer. In this case, the associated denial constraint would be of the form $\kappa^{\langle a \rangle} : \leftarrow R(a,y), S(y,z)$, and the rest would be basically as above.

Algorithms and complexity. Given the connection between causes and different kinds of repairs, we might take advantage for causality of algorithms and complexity results obtained for database repairs. This is matter of our ongoing research. In this work, apart from providing a naive algorithm for computing repairs from causes, we have not gone into detailed algorithm or complexity issues. The results we already have in this direction will be left for an extended version of this work.

Endogenous repairs. The partition of a database into endogenous and exogenous tuples has been exploited in the context of causality. However, this kind of partition is also of interest in the context of repairs. Considering that we should have more control on endogenous tuples than on exogenous ones, which may come from external sources, it makes sense to consider *endogenous repairs* that are obtained by updates (of any kind) on endogenous tuples. For example, in the case of violation of denial constraints, endogenous repairs would be obtained -if possible- by deleting endogenous tuples only. If there are no repairs based on endogenous tuples only, a preference condition could be imposed on repairs (Yakout et al. 2011; Staworko, Chomicki, and Marcinkowski 2012), privileging those that change exogenous the least. (Of course, it could also be the other way around, that is we may feel more inclined to change exogenous tuples than our endogenous ones.)

As a further extension, it could be possible to assume that combinations of (only) exogenous tuples never violate the ICs, something that could be checked at upload time. In this sense, there would be a part of the database that is considered to be consistent, while the other is subject to possible repairs. A situation like this has been considered, for other purposes and in a different form, in (Greco, Pijcke, and Wijsen 2014).

Actually, going a bit further, we could even consider the relations in the database with an extra, binary attribute, N, that is used to annotate if a tuple is endogenous or exogenous (it could be both), e.g. a tuple like $R(a, b, yes)$. ICs could be annotated too, e.g. the "exogenous" version of DC κ, could be $\kappa^E : \leftarrow P(x, y, yes), R(y, z, yes)$, and could be assumed to be satisfied.

ASP specification of causes. Above we have presented a connection between causes and repairs. S-repairs can be specified by means of answer set programs (ASPs) (Arenas, Bertossi, and Chomicki 2003; Barcelo, and Bertossi 2002; Barcelo, Bertossi, and Bravo 2003), and C-repairs too, with the use of weak program constraints (Arenas, Bertossi, and Chomicki 2003). This should allow for the introduction of ASPs in the context of causality, for specification and reasoning. There are also ASP-based specifications of diagnosis (Eiter et al. 1999) that could be brought into a more complete picture.

Causes and functional dependencies. Functional dependencies (FDs), that can be considered as denial constraints, have violation views that are conjunctive, but contain inequalities. They are still monotonic views though. Much has been done in the area of repairs and consistent query answering (Bertossi 2011). On the other side, in causality only conjunctive queries without built-ins have been considered (Meliou et al. 2010a). It is possible that causality can be extended to conjunctive queries with built-ins through the repair connection; and also to non-conjunctive queries via repairs wrt. more complex integrity constraints.

View updates. Another venue to explore for fruitful connections relates to the *view update problem*, which is about updating a database through views. This old and important problem in databases has also been treated from the point of view of abductive reasoning (Kakas, and Mancarella 1990; Console, Sapino, and Theseider-Dupre 1995).[4] User knowledge imposed through view updates creates or reflects *uncertainty* about the base data, because alternative base instances may give an account of the intended view updates.

The view update problem, specially in its particular form of of *deletion propagation*, has been recently related in (Kimelfeld 2012; Kimelfeld, Vondrak, and Williams 2012) to causality as introduced in (Meliou et al. 2010a).[5]

Database repairs are also related to the view update problem. Actually, *answer set programs* (ASP) for database repairs (Barcelo, Bertossi, and Bravo 2003) implicity repair the database by updating intentional, annotated predicates.

Even more, in (Bertossi, and Li 2013), in order to protect sensitive information, databases are explicitly and virtually "repaired" through secrecy views that specify the information that has to be kept secret. In order to protect information, a user is allowed to interact only with the virtually repaired versions of the original database that result from making those views empty or contain only null values. Repairs are specified and computed using ASP, and in (Bertossi, and Li 2013) an explicit connection to prioritized attribute-based repairs (Bertossi 2011) is made.

7 Conclusions

In this work, we have uncovered the relationships between causality in databases, database repairs, and consistency-based reasoning, as three forms of non-monotonic reasoning. Establishing the connection between these problems allows us to apply results and techniques developed for each of them to the others. This should be particularly beneficial for causality in databases, where still a limited number of results and techniques have been obtained or developed. This becomes matter of our ongoing and future research.

Our work suggests that diagnostic reasoning, as a form of non-monotonic reasoning, can provide a solid theoretical foundation for query answer explanation and provenance. The need for such foundation and the possibility of using non-monotonic logic for this purpose are mentioned in (Cheney et al. 2009; Cheney 2011).

[4] Abduction has also been explicitly applied to database repairs (Arieli et al. 2004).

[5] Notice only tuple deletions are used with violation views and repairs associated to denial constraints.

Acknowledgments: Research funded by NSERC Discovery, and the NSERC Strategic Network on Business Intelligence (BIN). L. Bertossi is a Faculty Fellow of IBM CAS. Conversations on causality in databases with Alexandra Meliou during Leo Bertossi's visit to U. of Washington in 2011 are much appreciated. He is also grateful to Dan Suciu and Wolfgang Gatterbauer for their hospitality. Leo Bertossi is also grateful to Benny Kimelfeld for stimulating conversations at LogicBlox, and pointing out to (Kimelfeld 2012; Kimelfeld, Vondrak, and Williams 2012).

References

Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. *Proc. ACM PODS*, 1999, pp. 68-79.

Arenas, M., Bertossi, L., Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 2003, 3(4&5):393-424.

Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296:405-434.

Arieli, O., Denecker, M., Van Nuffelen, B. and Bruynooghe, M. Coherent Integration of Databases by Abductive Logic Programming. *J. Artif. Intell. Res.*, 2004, 21:245-286.

Barcelo, P. and Bertossi, L. Repairing Databases with Annotated Predicate Logic. *Proc. NMR*, 2002.

Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics of Databases*, Springer LNCS 2582, 2003, pp. 1-27.

Bertossi, L. and Li, L. Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates. *IEEE Transaction on Knowledge and Data Engineering*, 2013, 25(5):987-1000.

Bertossi, L. *Database Repairing and Consistent Query Answering*. Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

Bertossi, L. Consistent Query Answering in Databases. *ACM SIGMOD Record*, 2006, 35(2):68-76.

Borgida, A., Calvanese, D. and Rodriguez-Muro, M. Explanation in DL-Lite. *Proc. DL WS*, CEUR-WS 353, 2008.

Buneman, P., Khanna, S. and Tan, W. C. Why and Where: A Characterization of Data Provenance. *Proc. ICDT*, 2001, pp. 316–330.

Buneman, P. and Tan, W. C. Provenance in Databases. *Proc. ACM SIGMOD*, 2007, pp. 1171–1173.

Chapman, A., and Jagadish, H. V. Why Not? *Proc. ACM SIGMOD*, 2009, pp.523–534.

Cheney, J., Chiticariu, L. and Tan, W. C. Provenance in Databases: Why, How, And Where. *Foundations and Trends in Databases*, 2009, 1(4): 379-474.

Cheney, J., Chong, S., Foster, N., Seltzer, M. I. and Vansummeren, S. Provenance: A Future History. *OOPSLA Companion (Onward!)*, 2009, pp. 957–964.

Cheney, J. Is Provenance Logical? *Proc. LID*, 2011, pp. 2–6.

Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 2005, 197(1-2):90-121.

Chockler, H. and Halpern, J. Y. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Intell. Res.*, 2004, 22:93-115.

Console, L., Sapino M. L., Theseider-Dupre, D. The Role of Abduction in Database View Updating. *J. Intell. Inf. Syst.*, 1995, 4(3): 261-280.

Cui, Y., Widom, J. and Wiener, J. L. Tracing The Lineage of View Data in a Warehousing Environment. *ACM Trans. Database Syst.*, 2000, 25(2):179-227.

Eiter, Th., Faber, W., Leone, N. and Pfeifer, G. The Diagnosis Frontend of the DLV System. *AI Commun.*, 1999, 12(1-2):99-111.

Gertz, M. Diagnosis and Repair of Constraint Violations in Database Systems. PhD Thesis, Universität Hannover, 1996.

Greco, S., Pijcke, F. and Wijsen, J. Certain Query Answering in Partially Consistent Databases. *PVLDB*, 2014, 7(5):353-364.

Halpern, Y. J., and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1 *Proc. UAI*, 2001, pp. 194-202.

Halpern, Y. J., and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1. *British J. Philosophy of Science*, 2005, 56:843-887.

Huang, J., Chen, T., Doan, A. and Naughton, J. F. On The Provenance of Non-Answers to Queries over Extracted Data. *PVLDB*, 2008, 1(1):736–747.

Kakas A. C. and Mancarella, P. Database Updates through Abduction. *Proc. VLDB*, 1990, pp. 650-661.

Karvounarakis, G. and Green, T. J. Semiring-Annotated Data: Queries and Provenance? *SIGMOD Record*, 2012, 41(3):5-14.

Karvounarakis, G. Ives, Z. G. and Tannen, V. Querying Data Provenance. *Proc. ACM SIGMOD*, 2010, pp. 951–962.

Kimelfeld, B. A Dichotomy in the Complexity of Deletion Propagation with Func- tional Dependencies. Proc. ACM PODS, 2012.

Kimelfeld, B., Vondrak, J. and Williams, R. Maximizing Conjunctive Views in Deletion Propagation. ACM Trans. Database Syst., 2012, 37(4):24.

Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. *Proc. ICDT*, 2007, Springer LNCS 4353, pp. 179-193.

Meliou, A., Gatterbauer, W. Moore, K. F. and Suciu, D. The Complexity of Causality and Responsibility for Query Answers and Non-Answers. *Proc. VLDB*, 2010, pp. 34-41.

Meliou, A., Gatterbauer. W., Halpern, J. Y., Koch, C., Moore K. F. and Suciu, D. Causality in Databases. *IEEE Data Eng. Bull*, 2010, 33(3):59-67.

Reiter, R. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 1987, 32(1):57-95.

Reiter, R. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, Springer, 1984, pp. 191-233.

Staworko, S., Chomicki, J. and Marcinkowski, J. Prioritized Repairing and Consistent Query Answering in Relational Databases. *Ann. Math. Artif. Intell.*, 2012, 64(2-3):209-246.

Struss, P. Model-based Problem Solving. In *Handbook of Knowledge Representation*, chapter 10. Elsevier, 2008.

Tannen, V. Provenance Propagation in Complex Queries. In *Buneman Festschrift*, 2013, Springer LNCS 8000, pp. 483173.

Yakout, M., Elmagarmid, A., Neville, J., Ouzzani, M. and Ilyas, I. Guided Data Repair. *PVLDB*, 2011, 4(5):279-289.

Interactive Debugging of ASP Programs

Kostyantyn Shchekotykhin
University Klagenfurt, Austria
kostya@ifit.uni-klu.ac.at

Abstract

Broad application of answer set programming (ASP) for declarative problem solving requires the development of tools supporting the coding process. Program debugging is one of the crucial activities within this process. Modern ASP debugging approaches allow efficient computation of possible explanations of a fault. However, even for a small program a debugger might return a large number of possible explanations and selection of the correct one must be done manually. In this paper we present an interactive query-based ASP debugging method which extends previous approaches and finds a preferred explanation by means of observations. The system automatically generates a sequence of queries to a programmer asking whether a set of ground atoms must be true in all (cautiously) or some (bravely) answer sets of the program. Since some queries can be more informative than the others, we discuss query selection strategies which, given user's preferences for an explanation, can find the best query. That is, the query an answer of which reduces the overall number of queries required for the identification of a preferred explanation.

Introduction

Answer set programming is a logic programming paradigm (Baral 2003; Brewka, Eiter, and Truszczynski 2011; Gebser et al. 2012) for declarative problem solving that has become popular during the last decades. The success of ASP is based on its fully declarative semantics (Gelfond and Lifschitz 1991) and availability of efficient solvers, e.g. (Simons, Niemelä, and Soininen 2002; Leone et al. 2006; Gebser et al. 2011). Despite a vast body of the theoretical research on foundations of ASP only recently the attention was drawn to the development of methods and tools supporting ASP programmers. The research in this direction focuses on a number of topics including integrated development environments (Febbraro, Reale, and Ricca 2011; Oetsch, Pührer, and Tompits 2011b; Sureshkumar et al. 2007), visualization (Cliffe et al. 2008), modeling techniques (Oetsch et al. 2011) and, last but not least, debugging of ASP programs.

Modern ASP debugging approaches are mostly based on declarative strategies. The suggested methods use elegant techniques applying ASP itself to debug ASP programs. The idea is to transform a faulty program in the special debugging program whose answer sets explain possible causes of a fault. These explanations are given by means of meta-atoms. A set of meta-atoms explaining a discrepancy between the set of actual and expected answer sets is called a *diagnosis*. In practice considering all possible diagnoses might be inefficient. Therefore, modern debugging approaches apply built-in minimization techniques of ASP solvers to compute only diagnoses comprising the minimal number of elements. In addition, the number of diagnoses can be reduced by so called *debugging queries*, i.e. sets of integrity constraints filtering out irrelevant diagnoses.

The computation of diagnoses is usually done by considering answer sets of a debugging program. In the approach of (Syrjänen 2006) a diagnosis corresponds to a set of meta-atoms indicating that a rule is removed from a program. (Brain et al. 2007) use the tagging technique (Delgrande, Schaub, and Tompits 2003) to obtain more fine-grained diagnoses. The approach differentiates between four types of problems: unsatisfied rules, unsupported atoms and unfounded loops. Each problem type is denoted by a special meta-predicate. Extraction of diagnoses can be done by a projection of an answer set of a debugging program to these meta-predicates. The most recent techniques (Gebser et al. 2008; Oetsch, Pührer, and Tompits 2010) apply *meta-programming*, where a program over a meta language is used to manipulate a program over an object language. Answer sets of a debugging meta-program comprise sets of atoms over meta-predicates describing faults of the similar nature as in (Brain et al. 2007).

The main problem of the aforementioned declarative approaches is that in real-world scenarios it might be problematic for a programmer to provide a complete debugging query. Namely, in many cases a programmer can easily specify some small number of atoms that must be true in a desired answer set, but not a complete answer set. In this case the debugging system might return many alternative diagnoses. Our observations of the students developing ASP programs shows that quite often the programs are tested and debugged on some small test instances. This way of development is quite similar to modern programming methodologies relying on unit tests (Beck 2003) which were implemented in ASPIDE (Febbraro et al. 2013) recently. Each test case calls a program for a predefined input and verifies

whether the actual output is the same as expected. In terms of ASP, a programmer often knows a set of facts encoding the test problem instance and a set of output atoms encoding the expected solution of the instance. What is often unknown are the "intermediate" atoms used to derive the output atoms. However, because of these atoms multiple diagnoses are possible. The problem is to find and add these atoms to a debugging query in a most efficient way[1]. Existing debugging systems (Brain and Vos 2005; Gebser et al. 2008; Oetsch, Pührer, and Tompits 2010) can be used in an "interactive" mode in which a user specifies only a partial debugging query as an input. Given a set of diagnoses computed by a debugger the user extends the debugging query, thus, filtering out irrelevant answer sets of a meta-program. However, this sort of interactivity still requires a user to select and provide atoms of the debugging query manually.

Another diagnosis selection issue is due to inability of a programmer to foresee all consequences of a diagnosis, i.e. in some cases multiple interpretations might have the same explanation for not being answer sets. The simplest example is an integrity constraint which can be violated by multiple interpretations. In this case the modification of a program accordingly to a selected diagnosis might have side-effects in terms of unwanted answer sets. These two problem are addressed by our approach which helps a user to identify the *target diagnosis*. The latter is the preferred explanation for a given set of atoms not being true in an answer set, on the one hand, and is not an explanation for unwanted interpretations, on the other.

In this paper we present an *interactive query-based debugging* method for ASP programs which differentiates between the diagnoses by means of additional observations (de Kleer and Williams 1987; Shchekotykhin et al. 2012). The latter are acquired by automatically generating a sequence of *queries* to an oracle such as a user, a database, etc. Each answer is used to reduce the set of diagnoses until the target diagnosis is found. In order to construct queries our method uses the fact that in most of the cases different diagnoses explain why different sets of interpretations are not answer sets. Consequently, we can differentiate between diagnoses by asking an oracle whether a set of atoms must be true or not in all/some interpretations relevant to the target diagnosis. Each set of atoms which can be used as a query is generated by the debugger automatically using discrepancies in the sets of interpretations associated with each diagnosis. Given a set of queries our method finds the best query according to a query selection strategy chosen by a user.

The suggested debugging approach can use a variety of query selection strategies. In this paper we discuss myopic and one step look-ahead strategies which are commonly used in active learning (Settles 2012). A myopic strategy implements a kind of greedy approach which in our case prefers queries that allow to reduce a set of diagnoses by half, regardless of an oracle's answer. The one step look-ahead strategy uses beliefs/preferences of a user for a cause/explanation of an error represented in terms of probability. Such a strategy selects those queries whose answers provide the most information gain, i.e. in whose answers a strategy is most uncertain about. New information provided by each answer is taken into account using Bayes-update. This allows the strategy to adapt its behavior on the fly.

To the best of our knowledge there are no approaches to interactive query-based ASP debugging allowing automatic generation and selection of queries. The method presented in this paper suggests an extension of the current debugging techniques by an effective user involvement in the debugging process.

[1]A recent user study indicates that the same problem can be observed also in the area of ontology debugging (see https://code.google.com/p/rmbd/wiki/UserStudy for preliminary results).

Preliminaries

A *disjunctive logic program* (DLP) Π is a finite set of rules of the form

$$h_1 \vee \cdots \vee h_l \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n$$

where all h_i and b_j are *atoms* and $0 \leq l, m, n$. A *literal* is an atom b or its negation $not\ b$. Each *atom* is an expression of the form $p(t_1, \ldots, t_k)$, where p is a predicate symbol and $t_1, \ldots, t_k$ are terms. A *term* is either a variable or a constant. The former is denoted by a string starting with an uppercase letter and the latter starting with a lowercase one. A literal, a rule or a program is called *ground*, if they are variable-free. A non-ground program Π, its rules and literals can be grounded by substitution of variables with constants appearing in Π. We denote the grounded instantiation of a program Π by $Gr(\Pi)$ and by $At(\Pi)$ the set of all ground atoms appearing in $Gr(\Pi)$.

The set of atoms $H(r) = \{h_1, \ldots, h_l\}$ is called the *head* of the rule r, whereas the set $B(r) = \{b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n\}$ is the body of r. In addition, it is useful to differentiate between the sets $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ comprising *positive* and *negative* body atoms. A rule $c \in \Pi$ with $H(c) = \emptyset$ is an *integrity constraint* and a rule $f \in \Pi$ with $B(f) = \emptyset$ is a *fact*. A rule r is *normal*, if $|H(r)| \leq 1$. A *normal program* includes only normal rules.

An *interpretation* I for Π is a set of ground atoms $I \subseteq At(\Pi)$. A rule $r \in Gr(\Pi)$ is *applicable* under I, if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$, otherwise the rule is *blocked*. We say that r is *unsatisfied* by I, if it is applicable under I and $H(r) \cap I = \emptyset$; otherwise r is *satisfied*. An interpretation I is a *model* of Π, if it satisfies every rule $r \in Gr(\Pi)$. For a ground program $Gr(\Pi)$ and an interpretation I the Gelfond-Lifschitz reduct is defined as $\Pi^I = \{H(r) \leftarrow B^+(r) | r \in Gr(\Pi), I \cap B^-(r) = \emptyset\}$. I is an *answer set* of Π, if I is a minimal model of Π^I (Gelfond and Lifschitz 1991). The program Π is *inconsistent*, if the set of all answer sets $AS(\Pi) = \emptyset$.

(Lee 2005) provides another characterization of answer sets of a program Π based on the notion of support. Thus, a rule $r \in Gr(\Pi)$ is a *support* for $A \subseteq At(\Pi)$ with respect to I, if a rule r is applicable under an interpretation I, $H(r) \cap A \neq \emptyset$ and $H(r) \cap I \subseteq A$. A support is *external*, if $B^+(r) \cap A = \emptyset$. A set of ground atoms A is *unsupported* by Π with

respect to I, if no rule in $Gr(\Pi)$ supports it. A *loop* is a non-empty set $L \subseteq At(\Pi)$ such that for any two distinct atoms $a_i, a_j \in L$ there is a path P in a positive dependency graph $G = (At(\Pi), \{(h,b) | r \in Gr(\Pi), h \in H(r), b \in B^+(r)\})$, where $P \neq \emptyset$ and $P \subseteq L$. A loop L is *unfounded* by Π with respect to I, if no rule in $Gr(\Pi)$ supports it externally; otherwise L is founded. An interpretation I is an answer set of Π, iff I is a model of Π such that each atom $a \in I$ is supported and each loop $L \subseteq I$ is founded (Lee 2005).

Debugging of ASP programs

The approach presented in our paper is based on the meta-programming technique presented in (Gebser et al. 2008). This debugging method focuses on the identification of *semantical errors* in a disjunctive logic program, i.e. disagreements between the actual answer sets of a program and the expected ones. The main idea is to use a program over a meta-language that manipulates another program over an object-language. The latter is a ground disjunctive program Π and the former is a non-ground normal logic program $\Delta[\Pi]$. Each answer set of a meta-program $\Delta[\Pi]$ comprises a set of atoms specifying an interpretation I and a number of meta-atoms showing, why I is not an answer set of a program Π. In addition, the method guarantees that there is at least one answer set of $\Delta[\Pi]$ for each interpretation I which is not an answer set of Π.

The debugger provides explanations of four error types denoted by the corresponding *error-indicating* predicates:

1. *Unsatisfied rules*: I is not a classical model of $Gr(\Pi)$ because the logical implication expressed by a rule r is false under I. Atom $unsatisfied(id_r)$ in an answer set of $\Delta[\Pi]$ expresses that a rule r is unsatisfied by I, where id_r is a unique identifier of a rule $r \in \Pi$.
2. *Violated integrity constraints*: I cannot be an answer set of $Gr(\Pi)$, if a constraint r is applicable under I. Atom $violated(id_r)$ indicates that r is violated under I.
3. *Unsupported atoms*: there is no rule $r \in Gr(\Pi)$ which allows derivation of $\{a\} \subseteq I$ and, therefore, I is not a minimal model of Π^I. Each unsupported atom a is indicated by an atom $unsupported(id_a)$ in an answer set of $\Delta[\Pi]$, where id_a is a unique identifier of an atom $a \in At(\Pi)$.
4. *Unfounded loops*: I is not a minimal model of Π^I, if a loop $L \subseteq I$ is unfounded by a program Π with respect to I. An atom $ufLoop(id_a)$ expresses that an atom $a \in At(\Pi)$ belongs to the unfounded loop L.

The set $Er(\Delta[\Pi]) \subseteq At(\Delta[\Pi])$ comprises all ground atoms over error-indicating predicates of the meta-program $\Delta[\Pi]$.

There are seven static modules in the meta-program $\Delta[\Pi]$, see (Gebser et al. 2008). The input module π_{in} comprises two sets of facts about atoms $\{atom(id_a) \leftarrow | a \in At(\Pi)\}$ and rules $\{rule(id_r) \leftarrow | r \in \Pi\}$ of the program Π. Moreover, for each rule $r \in \Pi$ the module π_{in} defines which atoms are in $H(r)$, $B^+(r)$ and $B^-(r)$. Module π_{int} generates an arbitrary interpretation I of a program Π as follows:

$$int(A) \leftarrow atom(A), not\ \overline{int}(A)$$
$$\overline{int}(A) \leftarrow atom(A), not\ int(A)$$

where atom $int(A)$ is complimentary to the atom $\overline{int}(A)$, i.e. no answer set can comprise both atoms. The module π_{ap} checks for every rule, whether it is applicable or blocked under I. The modules π_{sat}, π_{supp} and π_{ufloop} are responsible for the computation of at least one of the four explanations why I is not an answer set of Π listed above. Note, π_{ufloop} searches for unfounded loops only among atoms supported by Π with respect to I. This method ensures that each of the found loops is critical, i.e. it is a reason for I not being an answer set of Π. The last module, π_{noas} restricts the answer sets of $\Delta[\Pi]$ only to those that include any of the atoms over the error-indicating predicates.

The fault localization is done manually by means of *debugging queries* which specify an interpretation to be investigated as a set of atoms, e.g. $I = \{a\}$. Then I is transformed into a finite set of constraints, e.g. $\{\leftarrow \overline{int}(id_a), \leftarrow int(id_b), \dots\}$, pruning irrelevant answer sets of $\Delta[\Pi]$.

Fault localization in ASP programs

In our work we extend the meta-programming approach by allowing a user to specify *background theory* $\mathcal{B}$ as well as *positive* P and *negative* N test cases. In this section we show how this additional information is used to keep the search focused only on relevant interpretations and diagnoses.

Our idea of background knowledge is similar to (Brain et al. 2007) and suggests that some set of rules $\mathcal{B} \subseteq \Pi$ must be considered as correct by the debugger. In the meta-programming method the background theory can be accounted by addition of integrity constraints to π_{noas} which prune all answer sets of $\Delta[\Pi]$ suggesting that $r \in \mathcal{B}$ is faulty.

Definition 1. *Let $\Delta[\Pi]$ be a meta-program and $\mathcal{B} \subseteq \Pi$ a set of rules considered as correct. Then, a* debugging program *$\Delta[\Pi, \mathcal{B}]$ is defined as an extension of $\Delta[\Pi]$ with the rules:*

$$\{ \leftarrow rule(id_r),\ violated(id_r),$$
$$\leftarrow rule(id_r),\ unsatisfied(id_r) \mid r \in \mathcal{B}\}$$

In addition to background knowledge, further restrictions on the set of possible explanations of a fault can be made by means of test cases.

Definition 2. *Let $\Delta[\Pi, \mathcal{B}]$ be a debugging program. A* test case *for $\Delta[\Pi, \mathcal{B}]$ is a set $A \subseteq At(\Delta[\Pi, \mathcal{B}])$ of ground atoms over $int/1$ and $\overline{int}/1$ predicates.*

The test cases are either specified by a user before a debugging session or acquired by a system automatically as we show in subsequent sections.

Definition 3. *Let $\Delta[\Pi, \mathcal{B}]$ be a debugging program and $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$ a set of atoms over error-indicating predicates. Then a* diagnosis program *for $\mathcal{D}$ is defined as follows:*

$$\Delta[\Pi, \mathcal{B}, \mathcal{D}] := \Delta[\Pi, \mathcal{B}] \cup \{\leftarrow d_i \mid d_i \in Er(\Delta[\Pi, \mathcal{B}]) \setminus \mathcal{D}\}$$

In our approach we allow four types of test cases corresponding to the two ASP reasoning tasks (Leone et al. 2006):

- *Cautious reasoning*: all atoms $a \in A$ are true in *all* answer sets of the diagnosis program, resp. $\Delta[\Pi, \mathcal{B}, \mathcal{D}_t] \models_c A$, or not, resp. $\Delta[\Pi, \mathcal{B}, \mathcal{D}_t] \not\models_c A$. Cautiously true test cases are stored in the set CT^+ whereas cautiously false in the set CT^-.

- *Brave reasoning*: all atoms $a \in A$ are true in *some* answer set of the diagnosis program, resp. $\Delta[\Pi, \mathcal{B}, \mathcal{D}_t] \models_b A$, or not, resp. $\Delta[\Pi, \mathcal{B}, \mathcal{D}_t] \not\models_b A$. The set BT^+ comprises all bravely true test cases and the set BT^- all bravely false test cases.

In the meta-programming approach we handle the test cases as follows: Let $\mathcal{I}$ be a set of ground atoms resulting from a projection of an answer set $as \in AS(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ to the predicates $int/1$ and $\overline{int}/1$. By $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ we denote a set comprising all sets $\mathcal{I}_i$ for all $as_i \in AS(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$. Each set of grounded atoms $\mathcal{I}$ corresponds to an interpretation I of the program Π which is not an answer set of Π as explained by $\mathcal{D}$. The set $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ comprises a meta representation of each such interpretation for a diagnosis $\mathcal{D}$. Given a set of grounded atoms A, we say that $\mathcal{I}$ satisfies A (denoted $\mathcal{I} \models A$), if $A \subseteq \mathcal{I}$. $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ satisfies A (denoted $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models A$), if $\mathcal{I} \models A$ for every $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$. Analogously, we say that a set $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ is consistent with A, if there exists $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ which satisfies A.

Let A be a test case, then $\overline{A}$ denotes a complementary test case, i.e. $\overline{A} = \{\overline{int}(a) \mid int(a) \in A\} \cup \{int(a) \mid \overline{int}(a) \in A\}$. For the verification whether a diagnosis program $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ fulfills all test cases it is sufficient to check if the following conditions hold:

- $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models ct^+ \quad \forall ct^+ \in CT^+$
- $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models \overline{bt^-} \quad \forall bt^- \in BT^-$
- $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \cup \overline{ct^-}$ is consistent $\quad \forall ct^- \in CT^-$
- $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \cup bt^+$ is consistent $\quad \forall bt^+ \in BT^+$

As we can see a diagnosis program has the same verification procedure with respect to both cautiously true CT^+ bravely false BT^- test cases. The same holds for the cautiously false CT^- and bravely true BT^+ test cases. Therefore, in the following we can consider only the set of *positive* test cases P and the set of *negative* test cases N which are defined as:

$$P := CT^+ \cup \left\{\overline{bt^-} \mid bt^- \in BT^-\right\}$$
$$N := BT^+ \cup \left\{\overline{ct^-} \mid ct^- \in CT^-\right\}$$

Definition 4. *Let $\Delta[\Pi, \mathcal{B}]$ be a debugging program, P be a set of positive test cases, N be a set of negative test cases and $Er(\Delta[\Pi, \mathcal{B}])$ denote a set of all ground atoms over error-indicating predicates of $\Delta[\Pi, \mathcal{B}]$. A* diagnosis problem *is to find such set of atoms $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$, called* diagnosis*, such that the following requirements hold:*

- *the diagnosis program $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ is consistent,*
- *$Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models p \quad \forall p \in P$,*
- *$Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ is consistent with $n \quad \forall n \in N$.*

A tuple $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ is a diagnosis problem instance *(DPI).*

In the following we assume that the background theory $\mathcal{B}$ together with the sets of test cases P and N always allow computation of the target diagnosis. That is, a user provides reasonable background knowledge as well as positive and negative test cases that do not interfere with each other.

Proposition 1. *A diagnosis $\mathcal{D}$ for a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ does not exists if either (i) $\Delta' := \Delta[\Pi, \mathcal{B}] \cup \{a_i \leftarrow \mid a_i \in p, \forall p \in P\}$ is inconsistent or (ii) $\exists n \in N$ such that the program $\Delta' \cup \{a_i \leftarrow \mid a_i \in n\}$ is inconsistent.*

Proof. In the first case if Δ' is inconsistent, then either $\Delta[\Pi, \mathcal{B}]$ has no answer sets or every answer set of $\Delta[\Pi, \mathcal{B}]$ comprises an atom over $int/1$ or $\overline{int}/1$ predicate complimentary to some atom of a test case $p \in P$. The latter means that for any $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$ there exists $p \in P$ such that $\Delta[\Pi, \mathcal{B}, \mathcal{D}] \not\models p$. In the second case there exists a negative test case which is not consistent with any possible diagnosis program $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ for any $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$. Therefore in neither of the two cases requirements given in Definition 4 can be fulfilled for any $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$. □

Verification whether a set of atoms over error-indicating predicates is a diagnosis with respect to Definition 4 can be done according to the following proposition.

Proposition 2. *Let $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ be a DPI. Then, a set of atoms $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$ is a diagnosis for $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ iff $\Delta' := \Delta[\Pi, \mathcal{B}, \mathcal{D}] \cup \bigcup_{p \in P} \{a_i \leftarrow \mid a_i \in p\}$ is consistent and $\forall n \in N : \Delta' \cup \{a_i \leftarrow \mid a_i \in n\}$ is consistent.*

Proof. (sketch) ($\Rightarrow$) Let $\mathcal{D}$ be a diagnosis for $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$. Since $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ is consistent and $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models p$ for all $p \in P$ it follows that $\Delta[\Pi, \mathcal{B}, \mathcal{D}] \cup \bigcup_{p \in P} \{a_i \leftarrow \mid a_i \in p\}$ is consistent. The latter program has answer sets because every $p \in P$ is a subset of every $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$. In addition, since the set of meta-interpretations $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ is consistent with every $n \in N$ there exists such set $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ that $n \subseteq \mathcal{I}$. Therefore the program $\Delta[\Pi, \mathcal{B}, \mathcal{D}] \cup \{a_i \leftarrow \mid a_i \in n\}$ has at least one answer set. Taking into account that Δ' is consistent we can conclude that $\Delta' \cup \{a_i \leftarrow \mid a_i \in n\}$ is consistent as well.

($\Leftarrow$) Let $\mathcal{D} \subseteq Er(\Delta[\Pi, \mathcal{B}])$ and $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ be a DPI. Since Δ' is consistent the diagnosis program $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ is also consistent. Moreover, for all $p \in P$ $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}]) \models p$ because $\{a_i \leftarrow \mid a_i \in p\} \subseteq \Delta'$. Finally, for every $n \in N$ consistency of $\Delta' \cup \{a_i \leftarrow \mid a_i \in n\}$ implies that there must exist an interpretation $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}])$ satisfying n. □

Definition 5. *A diagnosis $\mathcal{D}$ for a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N\rangle$ is a* minimal diagnosis *iff there is no diagnosis $\mathcal{D}'$ such that $|\mathcal{D}'| < |\mathcal{D}|$.*

In our approach we consider only minimal diagnoses of a DPI since they might require less changes to the program than non-minimal ones and, thus, are usually preferred by users. However, this does not mean that our debugging approach is limited to minimal diagnoses of an initial DPI. As we will show in the subsequent sections the interactive debugger acquires test cases and updates the DPI automatically such that all possible diagnoses of the initial DPI are investigated. Computation of minimal diagnoses can be done by extension of the debugging program with such optimization criteria that only answer sets including minimal number of atoms over error-indicating predicates are returned by a

solver. Also, in practice a set of all minimal diagnoses is often approximated by a set of n diagnoses in order to improve the response time of a debugging system.

Computation of n diagnoses for the debugging program $\Delta[\Pi, \mathcal{B}]$ of a problem instance $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$ is done as shown in Algorithm 1. The algorithm calls an ASP solver to compute one answer set as of the debugging program (line 3). In case $\Delta[\Pi, \mathcal{B}]$ has an answer set the algorithm obtains a set $\mathcal{D}$ (line 5) and generates a diagnosis program $\Delta[\Pi, \mathcal{B}, \mathcal{D}]$ (line 6). The latter, together with the sets of positive and negative test cases is used to verify whether $\mathcal{D}$ is a diagnosis or not (line 7). All diagnoses are stored in the set $\mathbf{D}$. In order to exclude the answer set as from $AS(\Delta[\Pi, \mathcal{B}])$ the algorithm calls the EXCLUDE function (line 8) which extends the debugging program with the following integrity constraint, where atoms $d_1, \ldots, d_n \in \mathcal{D}$ and $d_{n+1}, \ldots, d_m \in Er(\Delta[\Pi, \mathcal{B}]) \setminus \mathcal{D}$:

$$\leftarrow d_1, \ldots, d_n, not\ d_{n+1}, \ldots, not\ d_m$$

Note, similarly to the model-based diagnosis (Reiter 1987; de Kleer and Williams 1987) our approach assumes that each error-indicating atom $er \in \mathcal{D}$ is relevant to an explanation of a fault, whereas all other atoms $Er(\Delta[\Pi]) \setminus \mathcal{D}$ are not. That is, some interpretations are not an answer sets of a program only because of reasons suggested by a diagnosis. Consequently, if a user selects a diagnosis $\mathcal{D}$ resulting in the debugging process, i.e. declares $\mathcal{D}$ as a correct explanation of a fault, then all other diagnoses automatically become incorrect explanations.

Example Let us exemplify our debugging approach on the following program Π_e:

$$\mathrm{r}_1 : a \leftarrow not\ d \qquad \mathrm{r}_2 : b \leftarrow a \qquad \mathrm{r}_3 : c \leftarrow b$$
$$\mathrm{r}_4 : d \leftarrow c \qquad \mathrm{r}_5 : \leftarrow d$$

Assume also that the *background theory* $\mathcal{B} = \{\leftarrow d\}$ and, therefore, the debugging program $\Delta[\Pi_e, \mathcal{B}]$ comprises two integrity constraints:

$$\leftarrow rule(id_{r_5}),\ violated(id_{r_5})$$
$$\leftarrow rule(id_{r_5}),\ unsatisfied(id_{r_5})$$

Since the program Π_e is inconsistent, a user runs the debugger to clarify the reason. In fact, the inconsistency is caused by an odd loop. That is, if d is set to false, then the body of the rule r_1 is satisfied and a is derived. However, given a and the remaining rules d must be set to true. In case when d is true, a is not derived and, consequently, there is no justification for d. The debugging program $\Delta[\Pi_e, \mathcal{B}]$ of a $DPI_1 := \langle \Delta[\Pi_e, \mathcal{B}], \emptyset, \emptyset \rangle$ has 16 answer sets. The addition of optimization criteria allows to reduce the number of answer sets to 4 comprising only the minimal number of atoms over the error-indicating predicates. Since both sets of test cases are empty, a projection of these answer sets to the error-indicating predicates results in the following diagnoses:

$$\mathcal{D}_1 : \{unsatisfied(id_{\mathrm{r}_1})\} \quad \mathcal{D}_2 : \{unsatisfied(id_{\mathrm{r}_2})\}$$
$$\mathcal{D}_3 : \{unsatisfied(id_{\mathrm{r}_3})\} \quad \mathcal{D}_4 : \{unsatisfied(id_{\mathrm{r}_4})\}$$

Definition 4 allows to identify the target (preferred) diagnosis $\mathcal{D}_t$ for the program Π_e by providing sufficient information in the sets $\mathcal{B}$, P and N. Assume that DPI_1 is updated with two test cases – one positive $\{int(a)\}$ and one negative $\{\overline{int}(b)\}$ – and the debugger generates $DPI_2 := \langle \Delta[\Pi_e, \mathcal{B}], \{\{int(a)\}\}, \{\{\overline{int}(b)\}\} \rangle$. These test cases require $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_t]) \models \{int(a)\}$ and $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_t])$ to be consistent with $\{\overline{int}(b)\}$ correspondingly. Given this information the debugger will return only one diagnosis in our example, namely $\mathcal{D}_2$, since $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_2]) \models \{int(a)\}$ and $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_2])$ is consistent with $\{\overline{int}(b)\}$. Indeed, a simple correction of Π_e by a user removing the rule r_2 results in a consistent program Π_2 such that all new answer sets of Π_2 fulfill all given test cases. All other sets of atoms $\mathcal{D}_1, \mathcal{D}_3, \mathcal{D}_4$ are not diagnoses of DPI_2 because they violate the requirements. Thus, $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_1]) \not\models \{int(a)\}$ and $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_i])$ is not consistent with $\{\overline{int}(b)\}$ for $\mathcal{D}_i \in \{\mathcal{D}_3, \mathcal{D}_4\}$. Consequently, $\mathcal{D}_2$ is the only possible diagnosis and it is accepted by a user as the target diagnosis $\mathcal{D}_t$.

Query-based diagnosis discrimination

The debugging system might generate a set of diagnoses for a given DPI. In our example for simple DPI_1 the debugger returns four minimal diagnoses $\{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$. As it is shown in the previous section, additional information, provided in the background theory and test cases of a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$ can be used by the debugging system to reduce the set of diagnoses. However, in a general case the user does not know which sets of test cases should be provided to the debugger s.t. the target diagnosis can be identified. That is, in many cases it might be difficult to provide a complete specification of a debugging query localizing a fault. Therefore, the debugging method should be able to find an appropriate set of atoms $A \subseteq At(\Pi)$ on its own and only query the user or some other oracle, whether these atoms are cautiously/bravely true/false in the interpretations associated with the target diagnosis. To generate a query for a set of diagnoses $\mathbf{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ the debugging system can use the diagnosis programs $\Delta[\Pi, \mathcal{B}, \mathcal{D}_i]$, where $\mathcal{D}_i \in \mathbf{D}$.

Since in many cases different diagnoses explain why different sets of interpretations of a program Π are not its answer sets, we can use discrepancies between the sets of interpretations to discriminate between the corresponding diagnoses. In our example, for each diagnosis program $\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_i]$ an ASP solver returns a set of answer sets encoding an interpretation which is not an answer set of Π_e and a diagnosis, see Table 1. Without any additional information the debugger cannot decide which of these atoms must be true in the missing answer sets of Π_e. To get this information the debugging algorithm should be able to access some oracle which can answer a number of queries.

Definition 6. *Let $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$ be a DPI, then a* query *is set of ground atoms $Q \subseteq At(\Pi)$.*

Each answer of an oracle provides additional information which is used to update the actual DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$. Thus, if an oracle answers

Algorithm 1: COMPUTEDIAGNOSES($\langle \Delta[\Pi, \mathcal{B}], P, N \rangle, n$)

Input: DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$, maximum number of minimal diagnoses n
Output: a set of diagnoses $\mathbf{D}$

$\mathbf{D} \leftarrow \emptyset$;
while $|\mathbf{D}| < n$ **do**
  $as \leftarrow$ GETANSWERSET($\Delta[\Pi, \mathcal{B}]$);
  if $as = \emptyset$ **then exit loop**;
  $\mathcal{D} \leftarrow as \cap Er(\Delta[\Pi, \mathcal{B}]))$;
  $\Delta[\Pi, \mathcal{B}, \mathcal{D}] \leftarrow$ DIAGNOSISPROGRAM($\Delta[\Pi, \mathcal{B}], \mathcal{D}$);
  if VERIFY($\Delta[\Pi, \mathcal{B}, \mathcal{D}], P, N$) **then** $\mathbf{D} \leftarrow \mathbf{D} \cup \{\mathcal{D}\}$;
  $\Delta[\Pi, \mathcal{B}] \leftarrow$ EXCLUDE($\Delta[\Pi, \mathcal{B}], \mathcal{D}$);
return $\mathbf{D}$;

Diagnosis	Interpretations
$\mathcal{D}_1 : unsatisfied(id_{r_1})$	$\{\{\overline{int}(a), \overline{int}(b), \overline{int}(c), \overline{int}(d)\}\}$
$\mathcal{D}_2 : unsatisfied(id_{r_2})$	$\{\{int(a), \overline{int}(b), \overline{int}(c), \overline{int}(d)\}\}$
$\mathcal{D}_3 : unsatisfied(id_{r_3})$	$\{\{int(a), int(b), \overline{int}(c), \overline{int}(d)\}\}$
$\mathcal{D}_4 : unsatisfied(id_{r_4})$	$\{\{int(a), int(b), int(c), \overline{int}(d)\}\}$

Table 1: Interpretations $Int(\Delta[\Pi_e, \mathcal{B}, \mathcal{D}_i])$ for each of the diagnoses $\mathbf{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$.

- *cautiously true*, the set $\{int(a) \mid a \in Q\}$ is added to P;
- *cautiously false*, the set $\{\overline{int}(a) \mid a \in Q\}$ is added to N;
- *bravely true*, the set $\{int(a) \mid a \in Q\}$ is added to N;
- *bravely false*, the set $\{\overline{int}(a) \mid a \in Q\}$ is added to P.

The goal of asking a query is to obtain new information characterizing the target diagnosis. For instance, the debugger asks a user about classification of the set of atoms $\{c\}$. If the answer is *cautiously true*, the new $DPI_3 = \langle \Delta[\Pi_e, \mathcal{B}], \{\{int(c)\}\}, \emptyset \rangle$ has only one diagnosis $\mathcal{D}_4$ which is the target diagnosis w.r.t. a user answer. All other minimal sets of atoms over error-indicating predicates are not diagnoses because they do not fulfill the necessary requirements of Definition 4. If the answer is *bravely false*, then the set $\{\overline{int}(c)\}$ is added to P and $\mathcal{D}_4$ is rejected. Consequently, we have to ask an oracle another question in order to discriminate between the remaining diagnoses. Since there are many subsets of $At(\Pi)$ which can be queried, the debugger has to generate and ask only those queries which allow to discriminate between the diagnoses of the current DPI.

Definition 7. *Each diagnosis $\mathcal{D}_i \in \mathbf{D}$ for a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$ can be assigned to one of the three sets $\mathbf{D^P}$, $\mathbf{D^N}$ or $\mathbf{D}^\emptyset$ depending on the query Q where:*

- $\mathcal{D}_i \in \mathbf{D^P}$ *if it holds that:*

$$Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_i]) \models \{int(a) \mid a \in Q\}$$

- $\mathcal{D}_i \in \mathbf{D^N}$ *if it holds that:*

$$Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_i]) \models \{\overline{int}(a) \mid a \in Q\}$$

- $\mathcal{D}_i \in \mathbf{D}^\emptyset$ *if* $\mathcal{D}_i \notin (\mathbf{D^P} \cup \mathbf{D^N})$

A partition *of the set of diagnoses $\mathbf{D}$ with respect to a query Q is denoted by a tuple $\langle Q, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$.*

Given a DPI we say that the diagnoses in $\mathbf{D^P}$ predict a positive answer (*yes*) as a result of the query Q, diagnoses in $\mathbf{D^N}$ predict a negative answer (*no*), and diagnoses in $\mathbf{D}^\emptyset$ do not make any predictions. Note, the answer *yes* corresponds to classification of the query to the set of positive test cases P, whereas the answer *no* is a result of a classification of the query to the set of negative test cases N. Therefore, without limiting the generality, in the following we consider only these two answers.

The notion of a partition has an important property. Namely, each partition $\langle Q, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$ indicates the changes in the set of diagnoses after the sets of test cases of an actual DPI are updated with respect to the answer of an oracle.

Property 1. *Let $\mathbf{D}$ be a set of diagnoses for a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$, Q be a query, $\langle Q, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$ be a partition of $\mathbf{D}$ with respect to Q and $v \in \{yes, no\}$ be an answer of an oracle to a query Q.*

- *if $v = yes$, then the set of diagnoses $\mathbf{D'}$ for the updated DPI $\langle \Delta[\Pi, \mathcal{B}], P', N \rangle$ does not comprise any elements of $\mathbf{D^N}$, i.e. $\mathbf{D'} \cap \mathbf{D^N} = \emptyset$ and $(\mathbf{D^P} \cup \mathbf{D}^\emptyset) \subseteq \mathbf{D'}$.*
- *if $v = no$, then for set of diagnoses $\mathbf{D'}$ of the updated DPI $\langle \Delta[\Pi, \mathcal{B}], P, N' \rangle$ it holds that $\mathbf{D'} \cap \mathbf{D^P} = \emptyset$ and $(\mathbf{D^N} \cup \mathbf{D}^\emptyset) \subseteq \mathbf{D'}$.*

Consequently, depending on the answer of an oracle to a query Q the set of diagnoses of an updated diagnosis problem instance comprises either $\mathbf{D^P} \cup \mathbf{D}^\emptyset$ or $\mathbf{D^N} \cup \mathbf{D}^\emptyset$.

In order to generate queries, we have to investigate for which sets $\mathbf{D^P}, \mathbf{D^N} \subseteq \mathbf{D}$ a query exists that can be used to

Algorithm 2: FINDPARTITIONS($\langle \Delta[\Pi, \mathcal{B}], P, N \rangle, \mathbf{D}$)

Input: DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$, a set of diagnoses $\mathbf{D}$
Output: a set of partitions $\mathbf{PR}$

1 $\mathbf{PR} \leftarrow \emptyset$;
2 **foreach** $\mathbf{D_i^P} \in \mathcal{P}(\mathbf{D})$ **do**
3 $\quad E_i \leftarrow$ COMMONATOMS($\mathbf{D_i^P}$);
4 $\quad Q_i \leftarrow \{a \mid int(a) \in E_i\}$;
5 $\quad$ **if** $Q_i \neq \emptyset$ **then**
6 $\quad\quad \langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle \leftarrow$ GENERATEPARTITION($Q_i, \mathbf{D}, \mathbf{D_i^P}$);
7 $\quad\quad$ **if** $\mathbf{D_i^N} \neq \emptyset$ **then** $\mathbf{PR} \leftarrow \mathbf{PR} \cup \{\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle\}$;
8 **return** $\mathbf{PR}$;

differentiate between them. A straight forward approach to query generation is to generate and verify all possible subsets of $\mathbf{D}$. This is feasible if we limit the number n of minimal diagnoses to be considered during the query generation and selection. For instance, given $n = 9$ the algorithm has to verify 512 partitions in the worst case. In general, the number of diagnoses n must be selected by a user depending on personal time requirements. The larger is the value of n the more time is required to compute a query, but an answer to this query will provide more information to a debugger.

Given a set of diagnoses $\mathbf{D}$ for a DPI $\langle \Delta[\Pi, \mathcal{B}], P, N \rangle$ Algorithm 2 computes a set of partitions $\mathbf{PR}$ comprising all queries that can be used to discriminate between the diagnoses in $\mathbf{D}$. For each element $\mathbf{D_i^P}$ of the power set $\mathcal{P}(\mathbf{D})$ the algorithm checks whether there is a set of atoms common to all interpretations of all diagnoses in $\mathbf{D_i^P}$. The function COMMONATOMS (line 3) returns an intersection of all sets $\mathcal{I} \in Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_j])$ for all $\mathcal{D}_j \in \mathbf{D_i^P}$. Given a non-empty query the function GENERATEPARTITION (line 6) uses Definition 7 to obtain a partition by classifying each diagnosis $\mathcal{D}_k \in \mathbf{D} \setminus \mathbf{D_i^P}$ into one of the sets $\mathbf{D_i^P}$, $\mathbf{D_i^N}$ or $\mathbf{D_i^\emptyset}$. Finally, all partitions allowing to discriminate between the diagnoses, i.e. comprising non-empty sets $\mathbf{D_i^P}$ and $\mathbf{D_i^N}$, are added to the set $\mathbf{PR}$.

Example (cont.) Reconsider the set of diagnoses $\mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$ for the DPI $\langle \Delta[\Pi_e, \{\leftarrow d\}], \emptyset, \emptyset \rangle$. The power set $\mathcal{P}(\mathbf{D}) = \{\{\mathcal{D}_1\}, \{\mathcal{D}_2\}, \ldots, \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}\}$ comprises 15 elements, assuming we omit the element corresponding to $\emptyset$ since it does not allow to compute a query. In each iteration an element of $\mathcal{P}(\mathbf{D})$ is assigned to the set $\mathbf{D_i^P}$. For instance, the algorithm assigned $\mathbf{D_0^P} = \{\mathcal{D}_1, \mathcal{D}_2\}$. In this case the set Q_0 is empty since the set $E_0 = \{\overline{int}(b), \overline{int}(c), \overline{int}(d)\}$ (see Table 1). Therefore, the set $\{\mathcal{D}_1, \mathcal{D}_2\}$ is rejected and removed from $\mathcal{P}(\mathbf{D})$. Assume that in the next iteration the algorithm selected $\mathbf{D_1^P} = \{\mathcal{D}_2, \mathcal{D}_3\}$, for which the set of common atoms $E_1 = \{int(a), \overline{int}(c), \overline{int}(d)\}$ and, thus, $Q_1 = \{a\}$. The remaining diagnoses $\mathcal{D}_1$ and $\mathcal{D}_4$ are classified according to Definition 7. That is, the algorithm selects the first diagnosis $\mathcal{D}_1$ and verifies whether $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_1]) \models \{int(a)\}$. Given the negative answer, the algorithm checks if $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_1]) \models \{\overline{int}(a)\}$. Since the condition is satisfied the diagnosis $\mathcal{D}_1$ is added to the set $\mathbf{D_1^N}$. The second diagnosis $\mathcal{D}_4$ is added to the set $\mathbf{D_1^P}$ as it satisfies the first requirement $Int(\Delta[\Pi, \mathcal{B}, \mathcal{D}_4]) \models \{int(a)\}$. The resulting partition $\langle \{a\}, \{\mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1\}, \emptyset \rangle$ is added to the set $\mathbf{PR}$. In general, Algorithm 2 returns a large number of possible partitions and the debugger has to select the best one. A random selection might not be a good strategy as it can overload an oracle with unnecessary questions (see (Shchekotykhin et al. 2012) for an evaluation of a random strategy). Therefore, the debugger has to decide query of which partition should be asked first in order to minimize the total number of queries to be answered. Query selection is the central topic of *active learning* (Settles 2012) which is an area of machine learning developing methods that are allowed to query an oracle for labels of unlabeled data instances. Most of the query selection measures used in active learning can be applied within our approach. In this paper, we discuss two query selection strategies, namely, myopic and one step look-ahead.

Myopic query strategies determine the best query using only the set of partitions $\mathbf{PR}$. A popular "Split-in-half" strategy prefers those queries which allow to remove a half of the diagnoses from the set $\mathbf{D}$, regardless of the answer of an oracle. That is, "Split-in-half" selects a partition $\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$ such that $|\mathbf{D_i^P}| = |\mathbf{D_i^N}|$ and $\mathbf{D_i^\emptyset} = \emptyset$. In our example, $\langle \{b\}, \{\mathcal{D}_3, \mathcal{D}_4\}, \{\mathcal{D}_1, \mathcal{D}_2\}, \emptyset \rangle$ is the preferred partition, since the set of all diagnoses of an updated DPI will comprise only two elements regardless of the answer of an oracle.

One step look-ahead strategies, such as prior entropy or information gain (Settles 2012), allow to find the target diagnosis using less queries by incorporating heuristics assessing the prior probability $p(\mathcal{D}_i)$ of each diagnosis $\mathcal{D}_i \in \mathbf{D}$ to be the target one (de Kleer and Williams 1987; Shchekotykhin et al. 2012). Such heuristics can express different preferences/expectations of a user for a fault explanation. For instance, one heuristic can state that rules including many literals are more likely to be faulty. Another heuristics can assign higher probabilities to diagnoses comprising atoms over $unsatisfiable/1$ predicate if a user expects this type of error. In addition, personalized heuris-

tics can be learned by analyzing the debugging actions of a user in, e.g., ASPIDE (Febbraro, Reale, and Ricca 2011) or SeaLion (Oetsch, Pührer, and Tompits 2011b).

A widely used one step look-ahead strategy (de Kleer and Williams 1987) suggests that the best query is the one which, given the answer of an oracle, minimizes the expected entropy of the set of diagnoses. Let $p(Q_i = v)$ denote the probability that an oracle gives an answer $v \in \{yes, no\}$ to a query Q_i and $p(\mathcal{D}_j|Q_i = v)$ be the probability of diagnosis $\mathcal{D}_j$ given an oracle's answer. The expected entropy after querying Q_i is computed as (see (Shchekotykhin et al. 2012) for details):

$$H_e(Q_i) = \sum_{v \in \{yes,no\}} p(Q_i = v) \times \\ - \sum_{\mathcal{D}_j \in \mathbf{D}} p(\mathcal{D}_j|Q_i = v) \log_2 p(\mathcal{D}_j|Q_i = v)$$

The required probabilities can be computed from the partition $\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$ for the query Q_i as follows:

$$p(Q_i = yes) = p(\mathbf{D_i^P}) + p(\mathbf{D_i^\emptyset})/2$$
$$p(Q_i = no) = p(\mathbf{D_i^N}) + p(\mathbf{D_i^\emptyset})/2$$

where the total probability of a set of diagnoses can be determined as: $p(\mathbf{S_i}) = \sum_{\mathcal{D}_j \in \mathbf{S_i}} p(\mathcal{D}_j)$, since all diagnoses are considered as mutually exclusive, i.e. they cannot occur at the same time. The latter follows from the fact that the goal of the interactive debugging process is identification of *exactly one* diagnosis that explains a fault and is accepted by a user. As soon as the user accepts the preferred diagnosis all other diagnoses become irrelevant. The total probability of diagnoses in the set $\mathbf{D_i^\emptyset}$ is split between positive and negative answers since these diagnoses make no prediction about outcome of a query, i.e. both outcomes are equally probable. Formally, the probability of an answer v for a query Q_i given a diagnosis $\mathcal{D}_j$ is defined as:

$$p(Q_i = v|\mathcal{D}_j) = \begin{cases} 1, & \text{if } \mathcal{D}_j \text{ predicted } Q_s = v; \\ 0, & \text{if } \mathcal{D}_j \text{ is rejected by } Q_s = v; \\ \frac{1}{2}, & \text{if } \mathcal{D}_j \in \mathbf{D_s^\emptyset}. \end{cases}$$

The probability of a diagnosis given an answer, required for the calculation of the entropy, can be found using the Bayes rule:

$$p(\mathcal{D}_j|Q_i = v) = \frac{p(Q_i = v|\mathcal{D}_j)p(\mathcal{D}_j)}{p(Q_i = v)}$$

After a query Q_s is selected by a strategy

$$Q_s = \arg\min_{Q_i} H_e(Q_i)$$

the system asks an oracle to provide its classification. Given the answer v of an oracle, i.e. $Q_s = v$, we have to update the probabilities of the diagnoses to take the new information into account. The update is performed by the Bayes rule given above.

In order to reduce the number of queries a user can specify a threshold, e.g. $\sigma = 0.95$. If the absolute difference in probabilities between two most probable diagnoses is greater than this threshold, the query process stops and returns the most probable diagnosis.

Note that, in the worst case the number of queries required to find the preferred diagnosis equals to the number of diagnoses of the initial DPI. In real-world applications, however, the worst case scenario is rarely the case. It is only possible if a debugger always prefers queries of such partitions $\langle Q_i, \mathbf{D_i^P}, \mathbf{D_i^N}, \mathbf{D_i^\emptyset} \rangle$ that either $|\mathbf{D_i^P}| = 1$ or $|\mathbf{D_i^N}| = 1$ and an answer of an oracle always unfavorable. That is, only one diagnosis of the actual DPI will not appear the set of diagnoses of the updated DPI.

We have not found any representative set of faulty ASP programs for which the preferred explanation of a fault, i.e. the target diagnosis, is known. Therefore, we do not report in this paper about the number of queries required to find such diagnosis. However, the evaluation results presented in (Shchekotykhin et al. 2012) show that only a small number of queries is usually required to find the preferred diagnosis. In the worst case their approach asked 12 queries on average to find the preferred diagnosis from over 1700 possible diagnoses. In better cases only 6 queries were required. This study indicates a great potential of the suggested method for debugging of ASP programs. We plan verify this conjecture in out future work. In addition, our approach can use RIO (Rodler et al. 2013), which is a query strategy balancing method that automatically selects the best query selection strategy during the diagnosis session, thus, preventing the worst case scenario.

The interactive debugging system (Algorithm 3) takes a ground program or a ground instantiation of non-ground program as well as a query selection strategy as an input. Optionally a user can provide background knowledge, relevant test cases as well as a set of heuristics assessing probabilities of diagnoses. If the first three sets are not specified, then the corresponding arguments are initialized with $\emptyset$. In case a user specified no heuristics, we add a simple function that assigns a small probability value to every diagnosis. The algorithm starts with the initialization of a DPI. The debugging program $\Delta[\Pi, \mathcal{B}]$ is generated by `spock`[2], which implements the meta-programming approach of (Gebser et al. 2008). First, the main loop of Algorithm 3 computes the required number of diagnoses such that $|\mathbf{D}| = n$. Next, we find a set of partitions for the given diagnoses and select a query according to a query strategy S selected by a user. If the user selected the myopic strategy then probabilities of diagnoses are ignored by SELECTQUERY. The oracle is asked to classify the query and the answer is used to update the DPI as well as a the set $\mathbf{D}$ from which we remove all elements that are not diagnoses of the updated DPI. The main loop of the algorithm exits if either there is a diagnosis which probability satisfies the threshold σ or only one diagnosis remains. Finally, the most probable diagnosis or, in case of a myopic strategy, the first diagnosis is returned to a user. Algorithm 3 was prototypically implemented as a part of a general diagnosis framework[3]. A plug-in for SeaLion providing a user-friendly interface for our interactive debug-

[2] www.kr.tuwien.ac.at/research/debug

[3] https://code.google.com/p/rmbd/wiki/AspDebugging

Algorithm 3: INTERACTIVEDEBUGGING($\Pi, S, \mathcal{B}, P, N, H, n, \sigma$)

Input: ground disjunctive program Π, query selection strategy S, background knowledge $\mathcal{B}$, sets of positive P and negative N test cases, set of heuristics H, maximum number minimal diagnoses n, acceptance threshold σ

Output: a diagnosis $\mathcal{D}$

```
⟨Δ[Π, B], P, N⟩ ← GENERATEDPI(Π, B); D ← ∅;
while BELOWTHRESHOLD(D, H, σ) ∧ |D| > 1 do
    D ← D ∪ COMPUTEDIAGNOSES(⟨Δ[Π, B], P, N⟩, n − |D|);
    PR ← FINDPARTITIONS(⟨Δ[Π, B], P, N⟩, D);
    Q ← SELECTQUERY(PR, H, S);
    if Q = ∅ then exit loop;
    A ← GETANSWER(Q);
    ⟨Δ[Π, B], P, N⟩ ← UPDATEDPI(A, ⟨Δ[Π, B], P, N⟩);
    D ← UPDATEDIADNOSES(A, Q, PR, H);
return MOSTPROBABLEDIAGNOSIS(D, S, H);
```

ging method is currently in development.

Summary and future work

In this paper we presented an approach to the interactive query-based debugging of disjunctive logic programs. The differentiation between the diagnoses is done by means of queries which are automatically generated from answer sets of the debugging meta-program. Each query partitions a set of diagnoses into subsets that make different predictions for an answer of an oracle. Depending on the availability of heuristics assessing the probability of a diagnosis to be the target one, the debugger can use different query selection strategies to find the most informative query allowing efficient identification of the target diagnosis.

In the future work we are going to investigate the applicability of our approach to the method of (Oetsch, Pührer, and Tompits 2010) since (a) this method can be applied to non-grounded programs and (b) it was recently extended to programs with choice rules, cardinality and weight constraints (Polleres et al. 2013). In addition, there is a number of other debugging methods for ASP that might be integrated with the suggested query selection approach. For instance, the method of (Mikitiuk, Moseley, and Truszczynski 2007) can be used to translate the program and queries into a natural language representation, thus, simplifying the query classification problem. Another technique that can be used to simplify the query answering is presented in (Pontelli, Son, and El-Khatib 2009) where the authors suggest a graph-based justification technique for truth values with respect to an answer set. Moreover, we would like to research whether query generation and selection ideas can be applied in the debugging method of (Oetsch, Pührer, and Tompits 2011a). This interactive framework allows a programmer to step through an answer set program by iteratively extending a state of a program (partial reduct) with new rules. The authors suggest a filtering approach that helps a user to find such rules and variable assignments that can be added to a state. We want to verify whether the filtering can be extended by querying about disagreements between the next states, such as "if user adds a rule r_1 then r_2 cannot be added".

One more interesting source of heuristics, that we also going to investigate, can be obtained during testing of ASP programs (Janhunen et al. 2010). The idea comes from spectrum-based fault localization (SFL) (Harrold et al. 1998), which is widely applied to software debugging. Given a set of test cases specifying inputs and outputs of a program SFL generates an observation matrix A which comprises information about: (i) parts of a program executed for a test case and (ii) an error vector E comprising results of tests executions. Formally, given a program with n software components $C := \{c_1, \ldots, c_n\}$ and a set of test cases $T := \{t_1, \ldots, t_m\}$ a hit spectra is a pair (A, E). A is a $n \times m$ matrix where each $a_{ij} = 1$ if c_j was involved in execution of the test case t_i and $a_{ij} = 0$ otherwise. Similarly for each $e_i \in E$, $e_i = 1$ if the test case t_i failed and $e_i = 0$ in case of a success. Obviously, statistics collected by the hit spectra after execution of all tests allows to determine the components that were involved in execution of failed test cases. Consequently, we can obtain a set of fault probabilities for the components C. The same methodology can be applied to debugging and testing of ASP programs. For each test case t_i we have to keep a record which sets of ground rules (Gelfond-Lifschitz reducts) were used to obtain answer sets that violate/satisfy t_i. Next, we can use the obtained statistics to derive fault probabilities for ground rules of an ASP program being debugged. The probabilities of diagnoses can then be computed from the probabilities of rules as it is shown in (Shchekotykhin et al. 2012).

Acknowledgments

The authors would like to thank Gerhard Friedrich and Patrick Rodler for the discussions regarding query selection strategies. We are also very thankful to anonymous reviewers for their helpful comments.

References

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.

Beck, K. 2003. *Test-driven development: by example*. Addison-Wesley Professional.

Brain, M., and Vos, M. D. 2005. Debugging Logic Programs under the Answer Set Semantics. In *Proceedings of the 3rd International Workshop on Answer Set Programming*, 141–152.

Brain, M.; Gebser, M.; Pührer, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2007. Debugging ASP programs by means of ASP. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, 31–43.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Cliffe, O.; Vos, M.; Brain, M.; and Padget, J. 2008. Aspviz: Declarative visualisation and animation using answer set programming. In Garcia de la Banda, M., and Pontelli, E., eds., *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, 724–728. Springer Berlin Heidelberg.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2003. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming* 3(02):129–187.

Febbraro, O.; Leone, N.; Reale, K.; and Ricca, F. 2013. Applications of Declarative Programming and Knowledge Management. In Tompits, H.; Abreu, S.; Oetsch, J.; Pührer, J.; Seipel, D.; Umeda, M.; and Wolf, A., eds., *Applications of Declarative Programming and Knowledge Management*, volume 7773 of *Lecture Notes in Computer Science*, 345–364. Berlin, Heidelberg: Springer Berlin Heidelberg.

Febbraro, O.; Reale, K.; and Ricca, F. 2011. ASPIDE: Integrated development environment for answer set programming. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, 317–330. Springer.

Gebser, M.; Pührer, J.; Schaub, T.; and Tompits, H. 2008. A meta-programming technique for debugging answer-set programs. In *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, 448–453.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications* 24(2):107–124.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publischers.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New generation computing* 9(3-4):365–386.

Harrold, M. J.; Rothermel, G.; Wu, R.; and Yi, L. 1998. An empirical investigation of program spectra. *ACM SIGPLAN Notices* 33(7):83–90.

Janhunen, T.; Niemelä, I.; Oetsch, J.; Pührer, J.; and Tompits, H. 2010. On Testing Answer-Set Programs. In *19th European Conference on Artificial Intelligence (ECAI-2010)*, 951–956.

Lee, J. 2005. A Model-theoretic Counterpart of Loop Formulas. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, 503–508. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)* 7(3):499–562.

Mikitiuk, A.; Moseley, E.; and Truszczynski, M. 2007. Towards Debugging of Answer-Set Programs in the Language PSpb. In *Proceedings of the 2007 International Conference on Artificial Intelligence*, 635–640.

Oetsch, J.; Pührer, J.; Seidl, M.; Tompits, H.; and Zwickl, P. 2011. VIDEAS : Supporting Answer-Set Program Development using Model-Driven Engineering Techniques. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, 382–387.

Oetsch, J.; Pührer, J.; and Tompits, H. 2010. Catching the Ouroboros: On Debugging Non-ground Answer-Set Programs. *Theory and Practice of Logic Programming* 10(4-6):2010.

Oetsch, J.; Pührer, J.; and Tompits, H. 2011a. Stepping through an Answer-Set Program. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, volume 231875, 134–147.

Oetsch, J.; Pührer, J.; and Tompits, H. 2011b. The SeaLion has Landed: An IDE for Answer-Set Programming – Preliminary Report. *CoRR* abs/1109.3989.

Polleres, A.; Frühstück, M.; Schenner, G.; and Friedrich, G. 2013. Debugging Non-ground ASP Programs with Choice Rules, Cardinality and Weight Constraints. In Cabalar, P., and Son, T., eds., *Logic Programming and Nonmonotonic Reasoning*, volume 8148 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 452–464.

Pontelli, E.; Son, T. C.; and El-Khatib, O. 2009. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming* 9(01):1.

Reiter, R. 1987. A Theory of Diagnosis from First Principles. *Artificial Intelligence* 32(1):57–95.

Rodler, P.; Shchekotykhin, K.; Fleiss, P.; and Friedrich, G. 2013. RIO: Minimizing User Interaction in Ontology Debugging. In Faber, W., and Lembo, D., eds., *Web Reasoning and Rule Systems*, volume 7994 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 153–167.

Settles, B. 2012. *Active Learning*, volume 6 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publischers.

Shchekotykhin, K.; Friedrich, G.; Fleiss, P.; and Rodler, P. 2012. Interactive ontology debugging: Two query strategies for efficient fault localization. *Web Semantics: Science, Services and Agents on the World Wide Web* 12-13(0):88 – 103.

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

Sureshkumar, A.; Vos, M. D.; Brain, M.; and Fitch, J. 2007.

APE: An AnsProlog* Environment. In *Software Engineering for Answer Set Programming*, 101–115.

Syrjänen, T. 2006. Debugging Inconsistent Answer Set Programs. In *Proceedings of the 11th International Workshop on Non-Monotonic Reasoning*, 77–84.

Semantics and Compilation of Answer Set Programming with Generalized Atoms

Mario Alviano
University of Calabria, Italy
mario@alviano.com

Wolfgang Faber
University of Huddersfield, UK
wf@wfaber.com

Abstract

Answer Set Programming (ASP) is logic programming under the stable model or answer set semantics. During the last decade, this paradigm has seen several extensions by generalizing the notion of atom used in these programs. Among these, there are aggregate atoms, HEX atoms, generalized quantifiers, and abstract constraints. In this paper we refer to these constructs collectively as generalized atoms. The idea common to all of these constructs is that their satisfaction depends on the truth values of a set of (non-generalized) atoms, rather than the truth value of a single (non-generalized) atom. Motivated by several examples, we argue that for some of the more intricate generalized atoms, the previously suggested semantics provide unintuitive results and provide an alternative semantics, which we call supportedly stable or SFLP answer sets. We show that it is equivalent to the major previously proposed semantics for programs with convex generalized atoms, and that it in general admits more intended models than other semantics in the presence of non-convex generalized atoms. We show that the complexity of supportedly stable models is on the second level of the polynomial hierarchy, similar to previous proposals and to stable models of disjunctive logic programs. Given these complexity results, we provide a compilation method that compactly transforms programs with generalized atoms in disjunctive normal form to programs without generalized atoms. Variants are given for the new supportedly stable and the existing FLP semantics, for which a similar compilation technique has not been known so far.

Introduction

Answer Set Programming (ASP) is a widely used problem-solving framework based on logic programming under the stable model semantics. The basic language relies on Datalog with negation in rule bodies and possibly disjunction in rule heads. When actually using the language for representing practical knowledge, it became apparent that generalizations of the basic language are necessary for usability. Among the suggested extensions are aggregate atoms (similar to aggregations in database queries) (Niemelä, Simons, and Soininen 1999; Niemelä and Simons 2000; Dell'Armi et al. 2003; Faber et al. 2008) and atoms that rely on external truth valuations (Calimeri, Cozza, and Ianni 2007; Eiter et al. 2004; 2005). These extensions are characterized by the fact that deciding the truth values of the new kinds of atoms depends on the truth values of a set of traditional atoms rather than a single traditional atom. We will refer to such atoms as generalized atoms, which cover also several other extensions such as abstract constraints, generalized quantifiers, and HEX atoms.

Concerning semantics for programs containing generalized atoms, there have been several different suggestions. All of these appear to coincide for programs that do not contain generalized atoms in recursive definitions. The two main semantics that emerged as standards are the PSP semantics defined in (Pelov 2004; Pelov, Denecker, and Bruynooghe 2007) and (Son and Pontelli 2007), and the FLP semantics defined in (Faber, Leone, and Pfeifer 2004; 2011). In a recent paper (Alviano and Faber 2013) the relationship between these two semantics was analyzed in detail; among other, more intricate results, it was shown that the semantics coincide up to convex generalized atoms. It was already established earlier that each PSP answer set is also an FLP answer set, but not vice versa. So for programs containing non-convex generalized atoms, some FLP answer sets are not PSP answer sets. In particular, there are programs that have FLP answer sets but no PSP answer sets.

In this paper, we argue that the FLP semantics is still too restrictive, and some programs that do not have any FLP answer set should instead have answer sets. In order to illustrate the point, consider a coordination game that is remotely inspired by the prisoners' dilemma. There are two players, each of which has the option to confess or defect. Let us also assume that both players have a fixed strategy already, which however still depends on the choice of the other player as well. In particular, each player will confess exactly if both players choose the same option, that is, if both players confess or both defect. The resulting program is P_1 in Example 2, where a means that the first player confesses and b means that the second player confesses. As will be explained later, the FLP semantics does not assign any answer set to this program, and therefore also the PSP semantics will not assign any answer sets to this program. However, this is peculiar, as the scenario in which both players confess seems like a reasonable one; indeed, even a simple inflationary operator would result in this solution.

Looking at the reason why this is not an FLP answer set, we observe that it has two countermodels that prevent it from being an answer set: One in which only the first player con-

fesses, and another one in which only the second player confesses. Both of these countermodels are models in the classical sense, but they are weak in the sense that they are not supported, meaning that there is no rule justifying their truth. This is a situation that does not occur for aggregate-free programs, which always have supported countermodels. We argue that one needs to look at supported countermodels, instead of looking at minimal countermodels. It turns out that doing this yields the same results not only for aggregate-free programs, but also for programs containing convex aggregates, which we believe is the reason why this issue has not been noticed earlier.

In this paper, we define a new semantics along these lines and call it supportedly stable or SFLP (supportedly FLP) semantics. It provides answer sets for more programs than FLP and PSP, but is shown to be equal on convex programs. Analyzing the computational complexity of the new semantics, we show that it is in the same classes as the FLP and PSP semantics when considering polynomial-time computable generalized atoms. It should also be mentioned that the new semantics has its peculiarities, for instance adding "tautological" rules like $a \leftarrow a$ can change the semantics of the program.

This complexity result directly leads us to the second contribution of this paper. While it has been known for quite some time that the complexity of programs with generalized atoms (even without disjunctions) is equal to the complexity of disjunctive programs, no compact transformation from programs with generalized atoms to disjunctive standard programs is known yet. We provide a contribution with this respect and show how to achieve such a compact compilation for both FLP and SFLP semantics when non-convex aggregates are in disjunctive normal form. It hinges on the use of disjunction and fresh symbols to capture satisfaction of a generalized atom.

The remainder of this paper is structured as follows. In the next section, we present the syntax and FLP semantics for programs with generalized atoms. After that, we analyze issues with the FLP semantics and define the SFLP semantics, followed by a section that proves several useful properties of the new semantics. The subsequent section then deals with compiling programs with generalized atoms into generalized-atom-free programs, followed by conclusions.

Syntax and FLP Semantics

In this section we present the syntax used in this paper and present the FLP semantics (Faber, Leone, and Pfeifer 2004; 2011). To ease the presentation, we will directly describe a propositional language here. This can be easily extended to the more usual ASP notations of programs involving variables, which stand for their ground versions (that are equivalent to a propositional program).

Syntax

Let $\mathcal{B}$ be a countable set of *propositional atoms*.

Definition 1. A *generalized atom* A on $\mathcal{B}$ is a mapping from $2^{\mathcal{B}}$ to Boolean truth values. Each generalized atom A has an associated, finite[1] domain $D_A \subseteq \mathcal{B}$, indicating those propositional atoms that are relevant to the generalized atom.

Example 1. A generalized atom A_1 modeling a conjunction $a_1, \ldots, a_n$ ($n \geq 0$) of propositional atoms is such that $D_{A_1} = \{a_1, \ldots, a_n\}$ and, for every $I \subseteq \mathcal{B}$, A_1 maps I to true if and only if $D_{A_1} \subseteq I$.

A generalized atom A_2 modeling a conjunction $a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n$ ($n \geq m \geq 0$) of literals, where $a_1, \ldots, a_n$ are propositional atoms and $\sim$ denotes *negation as failure*, is such that $D_{A_2} = \{a_1, \ldots, a_n\}$ and, for every $I \subseteq \mathcal{B}$, A_2 maps I to true if and only if $\{a_1, \ldots, a_m\} \subseteq I$ and $\{a_{m+1}, \ldots, a_n\} \cap I = \emptyset$.

A generalized atom A_3 modeling an aggregate $COUNT(\{a_1, \ldots, a_n\}) \neq k$ ($n \geq k \geq 0$), where $a_1, \ldots, a_n$ are propositional atoms, is such that $D_{A_3} = \{a_1, \ldots, a_n\}$ and, for every $I \subseteq \mathcal{B}$, A_3 maps I to true if and only if $|D_{A_3} \cap I| \neq k$.

In the following, when convenient, we will represent generalized atoms as conjunctions of literals or aggregate atoms. Subsets of $\mathcal{B}$ mapped to true by such generalized atoms will be those satisfying the associated conjunction.

Definition 2. A general rule r is of the following form:

$$H(r) \leftarrow B(r) \tag{1}$$

where $H(r)$ is a disjunction $a_1 \vee \cdots \vee a_n$ ($n \geq 0$) of propositional atoms in $\mathcal{B}$ referred to as the head of r, and $B(r)$ is a generalized atom on $\mathcal{B}$ called the body of r. For convenience, $H(r)$ is sometimes considered a set of propositional atoms.

A general program P is a set of general rules.

Example 2. Consider the following rules:

$$\begin{aligned} r_1: \quad & a \quad \leftarrow \quad COUNT(\{a, b\}) \neq 1 \\ r_2: \quad & b \quad \leftarrow \quad COUNT(\{a, b\}) \neq 1 \end{aligned}$$

The following are general programs:

$$\begin{aligned} P_1 &:= \{r_1; r_2\} \\ P_2 &:= \{r_1; r_2; a \leftarrow b; b \leftarrow a\} \\ P_3 &:= \{r_1; r_2; \leftarrow {\sim}a; \leftarrow {\sim}b\} \\ P_4 &:= \{r_1; r_2; a \vee b \leftarrow\} \\ P_5 &:= \{r_1; r_2; a \leftarrow {\sim}b\} \end{aligned}$$

FLP Semantics

An *interpretation* I is a subset of $\mathcal{B}$. I is a *model* for a generalized atom A, denoted $I \models A$, if A maps I to true. Otherwise, if A maps I to false, I is not a model of A, denoted $I \not\models A$. I is a model of a rule r of the form (1), denoted $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $I \models B(r)$. I is a model of a program P, denoted $I \models P$, if $I \models r$ for every rule $r \in P$.

Generalized atoms can be partitioned into two classes according to the following definition.

Definition 3 (Convex Generalized Atoms)**.** A generalized atom A is convex if for all triples I, J, K of interpretations such that $I \subset J \subset K$, $I \models A$ and $K \models A$ implies $J \models A$.

[1] In principle, we could also consider infinite domains, but refrain to do so for simplicity.

Note that convex generalized atoms are closed under conjunction (but not under disjunction or negation). A convex program is a general program whose rules have convex bodies.

We now describe a reduct-based semantics, usually referred to as FLP, which has been introduced and analyzed in (Faber, Leone, and Pfeifer 2004; 2011).

Definition 4 (FLP Reduct)**.** The FLP reduct P^I of a program P with respect to I is defined as the set $\{r \in P \mid I \models B(r)\}$.

Definition 5 (FLP Answer Sets)**.** I is an FLP answer set of P if $I \models P$ and for each $J \subset I$ it holds that $J \not\models P^I$. Let $FLP(P)$ denote the set of FLP answer sets of P.

Example 3. Consider the programs from Example 2. The models of P_1 are $\{a\}$, $\{b\}$ and $\{a, b\}$, none of which is an FLP answer set. Indeed,

$$P_1^{\{a\}} = P_1^{\{b\}} = \emptyset,$$

which have the trivial model $\emptyset$, which is of course a subset of $\{a\}$ and $\{b\}$. On the other hand

$$P_1^{\{a,b\}} = P_1,$$

and so

$$\{a\} \models P_1^{\{a,b\}},$$

where $\{a\} \subset \{a, b\}$. We will discuss in the next section why this is a questionable situation.

Concerning P_2, it has one model, namely $\{a, b\}$, which is also its unique FLP answer set. Indeed,

$$P_2^{\{a,b\}} = P_2,$$

and hence the only model of $P_2^{\{a,b\}}$ is $\{a, b\}$.

Interpretation $\{a, b\}$ is also the unique model of program P_3, which however has no FLP answer set. Here,

$$P_3^{\{a,b\}} = P_1,$$

hence similar to P_1,

$$\{a\} \models P_3^{\{a,b\}}$$

and $\{a\} \subset \{a, b\}$.

P_4 instead has two FLP answer sets, namely $\{a\}$ and $\{b\}$, and a further model $\{a, b\}$. In this case,

$$P_4^{\{a\}} = \{a \vee b \leftarrow\},$$

and no proper subset of $\{a\}$ satisfies it. Also

$$P_4^{\{b\}} = \{a \vee b \leftarrow\},$$

and no proper subset of $\{b\}$ satisfies it. Instead, for $\{a, b\}$, we have

$$P_4^{\{a,b\}} = P_4,$$

and hence

$$\{a\} \models P_4^{\{a,b\}}$$

and $\{a\} \subset \{a, b\}$.

Finally, P_5 has tree models, $\{a\}$, $\{b\}$ and $\{a, b\}$, but only one answer set, namely $\{a\}$. In fact, $P_5^{\{a\}} = \{a \leftarrow \sim b\}$ and $\emptyset$ is not a model of the reduct. On the other hand, $\emptyset$ is a model of $P_5^{\{b\}} = \emptyset$, and $\{a\}$ is a model of $P_5^{\{a,b\}} = P_1$.

SFLP Semantics

As noted in the introduction, the fact that P_1 has no FLP answer sets is striking. If we first assume that both a and b are false (interpretation $\emptyset$), and then apply a generalization of the well-known one-step derivability operator, we obtain truth of both a and b (interpretation $\{a, b\}$). Applying this operator once more again yields the same interpretation, a fix-point. $\{a, b\}$ is also a supported model, that is, for all true atoms there exists a rule in which this atom is the only true head atom, and in which the body is true.

It is instructive to examine why this seemingly robust model is not an FLP answer set. Its reduct is equal to the original program, $P_1^{\{a,b\}} = P_1$. There are therefore two models of P_1, $\{a\}$ and $\{b\}$, that are subsets of $\{a, b\}$ and therefore inhibit $\{a, b\}$ from being an FLP answer set. The problem is that, contrary to $\{a, b\}$, these two models are rather weak, in the sense that they are not supported. Indeed, when considering $\{a\}$, there is no rule in P_1 such that a is the only true atom in the rule head and the body is true in $\{a\}$: The only available rule with a in the head has a false body. The situation for $\{b\}$ is symmetric.

It is somewhat counter-intuitive that a model like $\{a, b\}$ should be inhibited by two weak models like $\{a\}$ and $\{b\}$. Indeed, this is a situation that normally does not occur in ASP. For programs that do not contain generalized atoms, whenever one finds a $J \subseteq I$ such that $J \models P^I$ there is for sure also a $K \subseteq I$ such that $K \models P^I$ and K is supported. Indeed, we will show in the following section that this is the case also for programs containing only convex generalized atoms. Our feeling is that since such a situation does not happen for a very wide set of programs, it has been overlooked so far.

We will now attempt to repair this kind of anomaly by stipulating that one should only consider supported models for finding inhibitors of answer sets. In other words, one does not need to worry about unsupported models of the reduct, even if they are subsets of the candidate. Let us first define supported models explicitly.

Definition 6 (Supportedness)**.** A model I of a program P is supported if for each $a \in I$ there is a rule $r \in P$ such that $I \cap H(r) = \{a\}$ and $I \models B(r)$. In this case we will write $I \models_s P$.

Example 4. Continuing Example 3, programs P_1, P_2, and P_3 have one supported model, namely $\{a, b\}$. The model $\{a\}$ of P_1 is not supported because the body of the the rule with a in the head has a false body with respect to $\{a\}$. For a symmetric argument, model $\{b\}$ of P_1 is not supported either. The supported models of P_4, instead, are $\{a\}$, $\{b\}$, and $\{a, b\}$, so all models of the program are supported. Note that both models $\{a\}$ and $\{b\}$ have the disjunctive rule as the only supporting rule for the respective single true atom, while for $\{a, b\}$, the two rules with generalized atoms serve as supporting rules for a and b. Finally, the supported models of P_5 are $\{a\}$ and $\{a, b\}$.

We are now ready to formally introduce the new semantics. In this paper we will normally refer to it as SFLP answer sets or SFLP semantics, but also call it supportedly stable models occasionally.

Definition 7 (SFLP Answer Sets)**.** I is a supportedly FLP answer set (or SFLP answer set, or supportedly stable model) of P if $I \models_s P$ and for each $J \subset I$ it holds that $J \not\models_s P^I$. Let $SFLP(P)$ denote the set of SFLP answer sets of P.

Example 5. Consider again the programs from Example 2. Recall that P_1 has only one supported model, namely $\{a, b\}$, and

$$P_1^{\{a,b\}} = P_1,$$

but

$$\emptyset \not\models_s P_1^{\{a,b\}},$$
$$\{a\} \not\models_s P_1^{\{a,b\}},$$
$$\{b\} \not\models_s P_1^{\{a,b\}},$$

therefore no proper subset of $\{a, b\}$ is a supported model, hence it is an SFLP answer set.

Concerning P_2, it has one model, namely $\{a, b\}$, which is supported and also its unique SFLP answer set. Indeed, recall that

$$P_2^{\{a,b\}} = P_2,$$

and hence no proper subset of $\{a, b\}$ can be a model (let alone a supported model) of $P_2^{\{a,b\}}$.

Interpretation $\{a, b\}$ is the unique model of program P_3, which is supported and also its SFLP answer set. In fact

$$P_3^{\{a,b\}} = P_1.$$

P_4 has two SFLP answer sets, namely $\{a\}$ and $\{b\}$. In this case, recall

$$P_4^{\{a\}} = \{a \vee b \leftarrow\},$$

and no proper subset of $\{a\}$ satisfies it. Also

$$P_4^{\{b\}} = \{a \vee b \leftarrow\},$$

and no proper subset of $\{b\}$ satisfies it. Instead, for $\{a, b\}$, we have

$$P_4^{\{a,b\}} = P_4,$$

hence since

$$\{a\} \models_s P_4^{\{a,b\}},$$
$$\{b\} \models_s P_4^{\{a,b\}},$$

we obtain that $\{a, b\}$ is not an SFLP answer set.

Finally, P_5 has two SFLP answer sets, namely $\{a\}$ and $\{a, b\}$. In fact, $P_5^{\{a\}} = \{a \leftarrow \sim b\}$ and $P_5^{\{a,b\}} = P_1$.

The programs, models, FLP answer sets, supported models, and SFLP answer sets are summarized in Table 1.

An alternative, useful characterization of SFLP answer sets can be given in terms of Clark's completion (Clark 1978). In fact, it is well-known that supported models of a program are precisely the models of its completion. We define this notion in a somewhat non-standard way, making use of the concept of generalized atom.

Next, we first define the completion of a propositional atom a with respect to a general program P as a generalized atom encoding the supportedness condition for a.

Definition 8. The completion of a propositional atom $a \in \mathcal{B}$ with respect to a general program P is a generalized atom $comp(a, P)$ mapping to true any interpretation I containing a and such that there is no rule $r \in P$ for which $I \models B(r)$ and $I \cap H(r) = \{a\}$.

These generalized atoms are then used to effectively define a program whose models are the supported model of P.

Definition 9. The completion of a general program P is a general program $comp(P)$ extending P with a rule

$$\leftarrow comp(a, P)$$

for each propositional atom a occurring in P.

Example 6. Consider again programs from Example 2. Program $comp(P_1)$ extends P_1 with the following rules:

$$\leftarrow \ a,\ COUNT(\{a, b\}) = 1$$
$$\leftarrow \ b,\ COUNT(\{a, b\}) = 1$$

Program $comp(P_2)$ extends P_2 with the following rules:

$$\leftarrow \ a,\ COUNT(\{a, b\}) = 1,\ \sim b$$
$$\leftarrow \ b,\ COUNT(\{a, b\}) = 1,\ \sim a$$

Program $comp(P_3)$ is equal to $comp(P_1)$, and program $comp(P_4)$ extends P_4 with the following rules:

$$\leftarrow \ a,\ COUNT(\{a, b\}) = 1,\ b$$
$$\leftarrow \ b,\ COUNT(\{a, b\}) = 1,\ a$$

Program $comp(P_5)$ instead extends P_5 with the following rules:

$$\leftarrow \ a,\ COUNT(\{a, b\}) = 1,\ b$$
$$\leftarrow \ b,\ COUNT(\{a, b\}) = 1$$

The only model of $comp(P_1)$, $comp(P_2)$, and $comp(P_3)$ is $\{a, b\}$. The models of $comp(P_4)$ and $comp(P_5)$ instead are $\{a\}$, $\{b\}$, and $\{a, b\}$.

Proposition 1. *Let P be a general program and I an interpretation. $I \models_s P$ iff $I \models comp(P)$.*

This characterization (which follows directly from (Clark 1978)) provides us with a means for implementation that relies only on model checks, rather than supportedness checks.

Proposition 2. *Let P be a general program and I an interpretation. I is a supportedly FLP answer set of P if $I \models comp(P)$ and for each $J \subset I$ it holds that $J \not\models comp(P^I)$.*

Properties

The new semantics has a number of interesting properties that we report in this section. First of all, it is an extension of the FLP semantics, in the sense that each FLP answer set is also an SFLP answer set.

Theorem 1. *Let P be a general program. $FLP(P) \subseteq SFLP(P)$.*

Proof. Let I be an FLP answer set of P. Hence, each $J \subset I$ is such that $J \not\models P^I$. Thus, we can conclude that $J \not\models_s P^I$ for any $J \subset I$. Therefore, I is a SFLP answer set of P. □

Table 1: (Supported) models and (S)FLP answer sets of programs in Example 2, where $A := COUNT(\{a, b\}) \neq 1$.

	Rules				Models	FLP	Supported Models	SFLP
P_1	$a \leftarrow A$	$b \leftarrow A$			$\{a\}, \{b\}, \{a, b\}$	—	$\{a, b\}$	$\{a, b\}$
P_2	$a \leftarrow A$	$b \leftarrow A$	$a \leftarrow b$	$b \leftarrow a$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
P_3	$a \leftarrow A$	$b \leftarrow A$	$\leftarrow \sim a$	$\leftarrow \sim b$	$\{a, b\}$	—	$\{a, b\}$	$\{a, b\}$
P_4	$a \leftarrow A$	$b \leftarrow A$	$a \vee b \leftarrow$		$\{a\}, \{b\}, \{a, b\}$	$\{a\}, \{b\}$	$\{a\}, \{b\}, \{a, b\}$	$\{a\}, \{b\}$
P_5	$a \leftarrow A$	$b \leftarrow A$	$a \leftarrow \sim b$		$\{a\}, \{b\}, \{a, b\}$	$\{a\}$	$\{a\}, \{a, b\}$	$\{a\}, \{a, b\}$

The inclusion is strict in general. In fact, P_1 is a simple program for which the two semantics disagree (see Examples 2–5 and Table 1). On the other hand, the two semantics are equivalent for a large class of programs, as shown below.

Theorem 2. *If P is a convex program then $FLP(P) = SFLP(P)$.*

Proof. $FLP(P) \subseteq SFLP(P)$ holds by Theorem 1. For the other direction, consider an interpretation I not being an FLP answer set of P. Hence, there is $J \subset I$ such that $J \models P^I$. We also assume that J is a subset-minimal model of P^I, that is, there is no $K \subset J$ such that $K \models P^I$. We shall show that $J \models_s P^I$. To this end, suppose by contradiction that there is $a \in J$ such that for each $r \in P^I$ either $J \not\models B(r)$ or $J \cap H(r) \neq \{a\}$. Consider $J \setminus \{a\}$ and a rule $r \in P^I$ such that $J \setminus \{a\} \models B(r)$. Since $r \in P^I$, $I \models B(r)$, and thus $J \models B(r)$ because $B(r)$ is convex. Therefore, $J \cap H(r) \neq \{a\}$. Moreover, $J \cap H(r) \neq \emptyset$ because $J \models P^I$ by assumption. Hence, $(J \setminus \{a\}) \cap H(r) \neq \emptyset$, and therefore $J \setminus \{a\} \models P^I$. This contradicts the assumption that J is a subset-minimal model of P^I. □

We will now focus on computational complexity. We consider here the problem of determining whether an SFLP answer set exists. We note that the only difference to the FLP semantics is in the stability check. For FLP, subsets need to be checked for being a model, for SFLP, subsets need to be checked for being a supported model. Intuitively, one would not expect that this difference can account for a complexity jump, which is confirmed by the next result.

Theorem 3. *Let P be a general program whose generalized atoms are polynomial-time computable functions. Checking whether $SFLP(P) \neq \emptyset$ is in Σ_2^P in general; it is Σ_2^P-hard already in the disjunction-free case if at least one form of non-convex generalized atom is permitted. The problem is NP-complete if P is disjunction-free and convex.*

Proof. For the membership in Σ_2^P one can guess an interpretation I and check that there is no $J \subset I$ such that $J \models_s P$. The check can be performed by a $coNP$ oracle.

To prove Σ_2^P-hardness we note that extending a general program P by rules $a \leftarrow a$ for every propositional atom occurring in P is enough to guarantee that all models of any reduct of P are supported. We thus refer to the construction and proof by (Alviano and Faber 2013).

If P is disjunction-free and convex then $SFLP(P) = FLP(P)$ by Theorem 2. Hence, NP-completeness follows from results in (Liu and Truszczyński 2006). □

We would like to point out that the above proof also illustrates a peculiar feature of SFLP answer sets, which it shares with the supported model semantics: the semantics is sensitive to tautological rules like $a \leftarrow a$, as their addition can turn non-SFLP answer sets into SFLP answer sets.

Compilation

The introduction of generalized atoms in logic programs does not increase the computational complexity of checking FLP as well as SFLP answer set existence, as long as one is allowed to use disjunctive rule heads. However, so far no compilation method that compactly transforms general programs to logic programs without generalized atoms has been presented for the FLP semantics. In the following we provide such a compilation for non-convex aggregates in disjunctive normal form. The compilation is also extended for the new SFLP semantics. We point out that such compilations are not necessarily intended to provide efficient methods for computing answer sets of general programs. Their purpose is instead to provide insights that may lead to obtain such methods in the future.

In this section we only consider generalized atoms in disjunctive normal form, that is, a generalized atom A will be associated with an equivalent propositional formula of the following form:

$$\bigvee_{i=1}^{k} a_{i_1} \wedge \ldots \wedge a_{i_m} \wedge \sim a_{i_{m+1}} \wedge \ldots \wedge \sim a_{i_n} \tag{2}$$

where $k \geq 1$, $i_n \geq i_m \geq 0$ and $a_{i_1}, \ldots, a_{i_n}$ are propositional atoms for $i = 1, \ldots, k$. We will also assume that the programs to be transformed have atomic heads. To generalize our compilations to cover disjunctive general rules is a problem to be addressed in future work.

Let P be a program. In our construction we will use the following *fresh* propositional atoms, i.e., propositional atoms not occurring in P: A^T for each generalized atom A; A^{F_i} for each generalized atom A and integer $i \geq 0$. For a generalized atom A of the form (2) and integer $i = 1, \ldots, k$, let $tr(A, i)$ denote the following rule:

$$A^T \vee a_{i_{m+1}} \vee \cdots \vee a_{i_n} \leftarrow a_{i_1}, \ldots, a_{i_m}, \sim A^{F_0}. \tag{3}$$

Moreover, let $fls(A, i, j)$ denote

$$A^{F_i} \leftarrow \sim a_{i_j}, \sim A^T \tag{4}$$

for $j = i_1, \ldots, i_m$, and

$$A^{F_i} \leftarrow a_{i_j}, \sim A^T \tag{5}$$

for $j = i_{m+1}, \ldots, i_n$. Abusing of notation, let $fls(A)$ denote the following rule:

$$A^{F_0} \leftarrow A^{F_1}, \ldots, A^{F_k}, {\sim}A^T. \tag{6}$$

Intuitively, rule $tr(A, i)$ forces truth of A^T whenever the i-th disjunct of A is true. Similarly, rule $fls(A, i, j)$ forces truth of A^{F_i} whenever the i-th disjunct of A is false due to atom a_{i_j}; if all disjuncts of A are false, rule $fls(A)$ forces truth of A^{F_0} to model that A is actually false. Note that atoms occurring in negative literals of the i-th disjunct of A have been moved in the head of $tr(A, i)$. In this way, the information encoded by $tr(A, i)$ is preserved in the reduct with respect to an interpretation I whenever the i-th disjunct of A is true with respect to a subset of I, not necessarily I itself.

The rewriting of A, denoted $rew(A)$, is the following set of rules:

$$\begin{array}{c} \{tr(A, i) \mid i = 1, \ldots, k\} \cup \{fls(A)\} \cup \\ \{fls(A, i, j) \mid i = 1, \ldots, k \wedge j = 1, \ldots, n\} \end{array} \tag{7}$$

The rewriting of P, denoted $rew(P)$, is obtained from P by replacing each generalized atom A by A^T. The FLP-rewriting of P, denoted $rew^{FLP}(P)$, is obtained from $rew(P)$ by adding rules in $rew(A)$ for each generalized atom A occurring in P. The SFLP-rewriting of P, denoted $rew^{SFLP}(P)$, is obtained from $rew^{FLP}(P)$ by adding a rule $supp(a)$ of the form

$$A_1^T \vee \cdots \vee A_n^T \leftarrow a \tag{8}$$

for each propositional atom a occurring in P, where $a \leftarrow A_i$ $(i = 1, \ldots, n)$ are the rules of P having head a.

Example 7. Let A be the generalized atom in Example 2. Its disjunctive normal form is ${\sim}a \wedge {\sim}b \vee a \wedge b$. Rules r_1 and r_2 are then $a \leftarrow A$ and $b \leftarrow A$. Program $rew^{FLP}(P_1)$ is

$$\begin{array}{rrcl} rew(\{r_1\}): & a & \leftarrow & A^T \\ rew(\{r_2\}): & b & \leftarrow & A^T \\ tr(A,1): & A^T \vee a \vee b & \leftarrow & {\sim}A^{F_0} \\ tr(A,2): & A^T & \leftarrow & a, b, {\sim}A^{F_0} \\ fls(A,1,1): & A^{F_1} & \leftarrow & a, {\sim}A^T \\ fls(A,1,2): & A^{F_1} & \leftarrow & b, {\sim}A^T \\ fls(A,2,1): & A^{F_2} & \leftarrow & {\sim}a, {\sim}A^T \\ fls(A,2,2): & A^{F_2} & \leftarrow & {\sim}b, {\sim}A^T \\ fls(A): & A^{F_0} & \leftarrow & A^{F_1}, A^{F_2}, {\sim}A^T \end{array}$$

One can check that $rew^{FLP}(P_1)$ has no answer set. In particular, $\{a, b, A^T\}$ is not an answer set of $rew^{FLP}(P_1)$. Its FLP reduct consists of the first four rules

$$\begin{array}{rcl} a & \leftarrow & A^T \\ b & \leftarrow & A^T \\ A^T \vee a \vee b & \leftarrow & {\sim}A^{F_0} \\ A^T & \leftarrow & a, b, {\sim}A^{F_0} \end{array}$$

and both $\{a\}$ and $\{b\}$ are minimal models of the reduct. On the other hand, neither $\{a\}$ nor $\{b\}$ are models of the original program, and so also not answer sets.

Program $rew^{SFLP}(P_1)$ extends $rew^{FLP}(P_1)$ with the following rules:

$$\begin{array}{rrcl} supp(a): & A^T & \leftarrow & a \\ supp(b): & A^T & \leftarrow & b \end{array}$$

The program $rew^{SFLP}(P_1)$ has one answer set:

$$\{a, b, A^T\}.$$

In contrast to $rew^{FLP}(P_1)$ its FLP reduct now consists of the first four rules of $rew^{FLP}(P_1)$ plus the two additional rules:

$$\begin{array}{rcl} a & \leftarrow & A^T \\ b & \leftarrow & A^T \\ A^T \vee a \vee b & \leftarrow & {\sim}A^{F_0} \\ A^T & \leftarrow & a, b, {\sim}A^{F_0} \\ A^T & \leftarrow & a \\ A^T & \leftarrow & b \end{array}$$

These two additional rules impede $\{a\}$ and $\{b\}$ to be models, and indeed only $\{a, b, A^T\}$ is a model of the reduct.

Program $rew^{FLP}(P_2)$ is $rew^{FLP}(P_1) \cup \{a \leftarrow b; b \leftarrow a\}$. (To simplify the presentation, bodies equivalent to atomic literals are not rewritten.)

In this case,

$$\{a, b, A^T\}$$

is its only answer set. Different to $rew^{FLP}(P_2)$, the additional rules will be present in the reduct for $\{a, b, A^T\}$:

$$\begin{array}{rcl} a & \leftarrow & A^T \\ b & \leftarrow & A^T \\ A^T \vee a \vee b & \leftarrow & {\sim}A^{F_0} \\ A^T & \leftarrow & a, b, {\sim}A^{F_0} \\ a & \leftarrow & b \\ b & \leftarrow & a \end{array}$$

Thus the reduct models $\{a\}$ and $\{b\}$ are avoided.

Program $rew^{SFLP}(P_2)$ extends $rew^{FLP}(P_2)$ with

$$\begin{array}{rrcl} supp(a)': & A^T \vee b & \leftarrow & a \\ supp(b)': & A^T \vee a & \leftarrow & b \end{array}$$

It is easy to see that these additional rules do not alter answer sets, so also $rew^{SFLP}(P_2)$ has a single answer set $\{a, b, A^T\}$.

Program $rew^{FLP}(P_3)$ is $rew^{FLP}(P_1) \cup \{\leftarrow {\sim}a; \leftarrow {\sim}b\}$. This program has no answer sets for the same reason as $rew^{FLP}(P_1)$. Indeed, the two additional rules are not in the reduct for $\{a, b, A^T\}$, and so $\{a\}$ and $\{b\}$ are again minimal models.

Program $rew^{SFLP}(P_3)$ is $rew^{SFLP}(P_1) \cup \{\leftarrow {\sim}a; \leftarrow {\sim}b\}$. For the same reason as for $rew^{SFLP}(P_1)$, this program has exactly one answer set:

$$\{a, b, A^T\}.$$

The two new rules disappear in the reduct, but the rules present in $rew^{SFLP}(P_1)$ but not in $rew^{FLP}(P_1)$ do not allow models $\{a\}$ and $\{b\}$.

Program P_4 contains a disjunctive rule and is thus not in the domain of rew^{FLP} and rew^{SFLP} described here.

In the examples provided so far, it can be checked that answer sets are preserved by our transformations if auxiliary symbols are ignored. In the remainder of this section we will formalize this intuition.

Definition 10. The expansion of an interpretation I for a program P, denoted $exp(I)$, is the following interpretation:

$$\begin{aligned} I \cup \{A^T \mid A^T \text{ occurs in } rew(P),\ I \models A\} \\ \cup \{A^{F_i} \mid A^{F_i} \text{ occurs in } rew(P),\ I \not\models A\}. \end{aligned} \tag{9}$$

The contraction of an interpretation I to the symbols of P, denoted $I|_P$, is the following interpretation:

$$I \cap \{a \in \mathcal{B} \mid a \text{ occurs in } P\}. \tag{10}$$

Below, we show that expansions and contractions define bijections between the answer sets of a program and those of the corresponding compilations. In the claim we consider only FLP answer sets of the rewritten program because it is convex, and thus its FLP and SFLP answer sets coincide by Theorem 2.

Theorem 4. *Let P be a program, and $\mathcal{F} \in \{FLP, SFLP\}$.*

1. *If $I \in \mathcal{F}(P)$ then $exp(I) \in FLP(rew^{\mathcal{F}}(P))$.*
2. *If $I \in FLP(rew^{\mathcal{F}}(P))$ then $I|_P \in \mathcal{F}(P)$.*

Proof (item 1). Let I be an $\mathcal{F}$ answer set of P. Hence, $I \models_s P$ (see Definition 7 and Theorem 1). Since each generalized atom A occurring in P is replaced by A^T in $rew(P)$, and $A^T \in exp(I)$ if and only if $I \models A$, we have $I \models rew(P)$. Consider rules in $rew(A)$ for some generalized atom A of the form (2) occurring in P, and note that either $A^T \in exp(I)$ or $A^{F_0}, \ldots, A^{F_k} \in exp(I)$. In both cases, all rules in $rew(A)$ are satisfied by $exp(I)$. Hence, $exp(I) \models rew^{FLP}(P)$. Consider a rule $supp(a)$ of the form (8) such that $a \in I$. Since $I \models_s P$, there is $i \in \{1, \ldots, n\}$ such that $I \models A_i$. Thus, $A_i^T \in exp(I)$, and therefore $exp(I) \models supp(a)$. We can conclude $exp(I) \models rew^{SFLP}(P)$.

Let $J \subseteq exp(I)$ be such that $J \models rew^{\mathcal{F}}(P)^{exp(I)}$. We first show that $J|_P = I$. Consider a rule $a \leftarrow A$ in P^I such that $I \models A$ and $J|_P \models A$, where A is of the form (2). Hence, there is $i \in \{1, \ldots, k\}$ such that

$$J|_P \models a_{i_1} \wedge \ldots \wedge a_{i_m} \wedge {\sim}a_{i_{m+1}} \wedge \ldots \wedge {\sim}a_{i_n}.$$

Therefore, $A^T \in J$ because $tr(A, i) \in rew^{\mathcal{F}}(P)^{exp(I)}$, and consequently $a \in J$ because of rule $a \leftarrow A^T$ in $rew^{\mathcal{F}}(P)^{exp(I)}$. We thus conclude $J|_P \models P^I$. For $\mathcal{F} = FLP$, this already proves $J|_P = I$. For $\mathcal{F} = SFLP$, let $X \subseteq J|_P$ be the atoms without support, i.e., X is a subset-maximal set such that $a \in X$ implies $J|_P \setminus X \not\models A$ for each rule $a \leftarrow A$ in P^I. Hence, $J|_P \setminus X \models_s P^I$. It follows that $J|_P \setminus X = I$, i.e., $X = \emptyset$ and $J|_P = I$.

We can now show that $J = exp(I)$. Let A be a generalized atom of the form (2). If $J|_P \models A$ there is $i \in \{1, \ldots, k\}$ such that

$$J|_P \models a_{i_1} \wedge \ldots \wedge a_{i_m} \wedge {\sim}a_{i_{m+1}} \wedge \ldots \wedge {\sim}a_{i_n},$$

and thus $A^T \in J$ because $tr(A, i) \in rew^{\mathcal{F}}(P)^{exp(I)}$ and $J \models rew^{\mathcal{F}}(P)^{exp(I)}$. Otherwise, if $J|_P \not\models A$ then for all $i \in \{1, \ldots, k\}$ there is either $j \in \{1, \ldots, m\}$ such that $a_{i_j} \notin J|_P$, or $j \in \{m+1, \ldots, n\}$ such that $a_{i_j} \in J|_P$. Hence, $A^{F_i} \in J$ because $J \models fls(A, i, j)$, and thus $A^{F_0} \in J$ because $J \models fls(A)$. □

Proof (item 2). Let I be an FLP answer set of $rew^{\mathcal{F}}(P)$. Let A be a generalized atom A of the form (2) occurring in P. We prove the following statements:

$$|I \cap \{A^T, A^{F_i}\}| \leq 1 \text{ holds for } i = 1, \ldots, k \tag{11}$$

$$A^T \in I \text{ if and only if } I|_P \models A \tag{12}$$

$$|I \cap \{A^T, A^{F_i}\}| = 1 \text{ holds for } i = 1, \ldots, k \tag{13}$$

To prove (11), define set X as a maximal subset satisfying the following requirements: If $\{A^T, A^{F_i}\} \subseteq I$ (for some $i \in \{1, \ldots, k\}$) then $\{A^T, A^{F_0}, \ldots, A^{F_k}\} \subseteq X$; if an atom a is not supported by $I \setminus X$ in $rew^{FLP}(P)^I$ then $a \in X$. We have $I \setminus X \models rew^{\mathcal{F}}(P)^I$, from which we conclude $X = \emptyset$.

Consider (12). If $A^T \in I$ then by (11) no A^{F_i} belongs to I. Recall that FLP answer sets are supported models, i.e., $I \models_s rew^{\mathcal{F}}(P)$. Thus, for $\mathcal{F} = FLP$, there is $i \in \{1, \ldots, k\}$ such that $I \models B(tr(A, i))$ and $I \cap H(tr(A, i)) = \{A^T\}$. Therefore, $I|_P \models A$. For $\mathcal{F} = SFLP$, we just note that if A^T is supported only by a rule of the form (8), then atom a is only supported by a rule $a \leftarrow A^T$ in $rew^{\mathcal{F}}(P)$. $I \setminus \{a, A^T\}$ would be a model of $rew^{\mathcal{F}}(P)^I$ in this case, then contradicting $I \in FLP(rew^{\mathcal{F}}(P))$. Now consider the right-to-left direction. If $I|_P \models A$ then there is $i \in \{1, \ldots, k\}$ such that $I|_P \models a_{i_1} \wedge \ldots \wedge a_{i_m} \wedge {\sim}a_{i_{m+1}} \wedge \ldots \wedge {\sim}a_{i_n}$, and thus $A^{F_i} \notin I$ (see Equations 4–5). Hence, $A^{F_0} \notin I$ (see Equation 6). From rule $tr(A, i)$ (see Equation 3) we have $A^T \in I$.

Concerning (13), because of (11) and (12), we have just to show that $A^{F_0}, \ldots, A^{F_k} \in I$ whenever $I|_P \not\models A$. In fact, in this case $A^T \notin I$ by (12), and for each $i \in \{1, \ldots, k\}$ there is either $j \in \{1, \ldots, m\}$ such that $a_{i_j} \notin I|_P$, or $j \in \{m+1, \ldots, n\}$ such that $a_{i_j} \in I|_P$. Hence, $A^{F_i} \in I$ because of rules $fls(r, i, j)$ and $fls(r)$.

We can now prove the main claim. We start by showing that $I|_P \models P$. Indeed, for a rule $a \leftarrow A$ in P such that $I|_P \models A$, $rew(P)$ contains a rule $a \leftarrow A^T$. Moreover, $A^T \in I$ by (12), and thus $a \in I$. If $\mathcal{F} = SFLP$, then for each $a \in I$ we have $I \models supp(a)$, where $supp(a)$ is of the form (8). Hence, there is $i \in 1, ..., n$ such that $A_i^T \in I$. Therefore, (12) implies $I|_P \models A_i$, that is, a is supported by $I|_P$ in P. We can thus conclude that $I_P \models_s P$.

To complete the proof, for $\mathcal{F} = FLP$ we consider $X \subseteq I|_P$ such that $I|_P \setminus X \models P^{I|_P}$, while for $\mathcal{F} = SFLP$ we consider $X \subseteq I|_P$ such that $I|_P \setminus X \models_s P^{I|_P}$. Let J be the interpretation obtained from $I \setminus X$ by removing all atom A^T such that $I|_P \setminus X \not\models A$. We shall show that $J \models rew^{\mathcal{F}}(P)^I$, from which we conclude $X = \emptyset$. Consider a rule of the form $a \leftarrow A^T$ in $rew^{\mathcal{F}}(P)^I$ such that $A^T \in J$. Hence, $I|_P \setminus X \models A$ by construction of J. Since $a \leftarrow A$ is a rule in $P^{I|_P}$, we conclude $a \in I|_P \setminus X$ and thus $a \in J$. Consider now a rule $tr(A, i)$ in $rew^{\mathcal{F}}(P)^I$ such that $J \models B(tr(A, i))$ and $A^T \notin J$. Hence, $I|_P \setminus X \not\models A$ by construction of J, which means that there is either $j \in \{1, \ldots, m\}$ such that

$a_{i_j} \notin I|_P \setminus X$, or $j \in \{m+1, \ldots, n\}$ such that $a_{i_j} \in I|_P \setminus X$. We conclude that $J \models tr(A, i)$. Rules $fls(A, i, j)$ and $fls(A)$ are satisfied as well because no A^{F_i} has been removed. For $\mathcal{F} = SFLP$, consider a rule $supp(a)$ of the form (8) such that $a \in J$. Since $I|_P \setminus X \models_s P^{I|_P}$, there is rule $a \leftarrow A$ in $P^{I|_P}$ such that $I|_P \setminus X \models A$. Hence, by construction of J, $A^T \in J$ and thus $J \models supp(a)$. □

Conclusion

In this paper, we have first defined a new semantics for programs with generalized atoms, called supportedly stable models, supportedly FLP, or SFLP semantics. We have motivated its definition by an anomaly that arises for the FLP semantics in connection with non-convex generalized atoms. In particular, only unsupported models may in particular cases inhibit the stability of candidate models. The new definition overcomes this anomaly and provides a robust semantics for programs with generalized atoms. We show several properties of this new semantics, for example it coincides with the FLP semantics (and thus also the PSP semantics) on convex programs, and thus also on standard programs. Furthermore, the complexity of reasoning tasks is equal to the respective tasks using the FLP semantics. We also provide a characterization of the new semantics by a Clark-inspired completion.

We observe that other interesting semantics, such as the one by (Ferraris 2005), are also affected by the anomaly on unsupported models. In particular, the semantics by (Ferraris 2005) is presented for programs consisting of arbitrary set of propositional formulas, and it is based on a reduct in which false subformulas are replaced by $\bot$. Answer sets are then defined as interpretations being subset-minimal models of their reducts. For the syntax considered in this paper, when rewriting generalized atoms to an equivalent formula, the semantics by (Ferraris 2005) coincides with FLP, which immediately shows the anomaly. In (Ferraris 2005) there is also a method for rewriting aggregates, however $COUNT(\{a, b\}) \neq 1$ is not explicitly supported, but should be rewritten to $\neg(COUNT(\{a, b\}) = 1)$. Doing this, one can observe that for P_1, P_2, P_3, and P_5 the semantics of (Ferraris 2005) behaves like SFLP (cf. Table 1), while for P_4 the semantics of (Ferraris 2005) additionally has the answer set $\{a, b\}$, which is not a supported minimal model of the FLP reduct. P_4 therefore shows that the two semantics do not coincide, even if generalized atoms are interpreted as their negated complements, and the precise relationship is left for further study. However, we also believe that rewriting a generalized atom into its negated complement is not always natural, and we are also not convinced that there should be a semantic difference between a generalized atom and its negated complement.

The second part of the paper concerns the question of compactly compiling generalized atoms away, to arrive at a program that contains only traditional atoms whose answer sets are in a one-to-one correspondence with the original program. Previously existing complexity results indicated that such a translation can exist, but that it has to make use of disjunction in rule heads. However, no such method is currently known. We show that similar techniques can be used for both FLP and the new SFLP semantics when non-convex aggregates are represented in disjunctive normal form.

Concerning future work, implementing a reasoner supporting the new semantics would be of interest. However, we believe that it would actually be more important to collect example programs that contain non-convex generalized atoms in recursive definitions. We have experimented with a few simple domains stemming from game theory (as outlined in the introduction), but we are not aware of many other attempts. Our intuition is that such programs would be written in several domains that describe features with feedback loops, which applies to many so-called complex systems. Also computing or checking properties of neural networks might be a possible application in this area. Another, quite different application area could be systems that loosely couple OWL ontologies with rule bases, for instance by means of HEX programs. HEX atoms interfacing to ontologies will in general not be convex, and therefore using them in recursive definitions falls into our framework, where the FLP and SFLP semantics differ.

Another area of future work arises from the fact that rules like $a \leftarrow a$ are not irrelevant for the SFLP semantics. To us, it is not completely clear whether this is a big drawback. However, we intend to study variants of the SFLP semantics that do not exhibit this peculiarity.

References

Alviano, M., and Faber, W. 2013. The complexity boundary of answer set programming with generalized atoms under the flp semantics. In Cabalar, P., and Tran, S. C., eds., *Logic Programming and Nonmonotonic Reasoning — 12th International Conference (LPNMR 2013)*, number 8148 in Lecture Notes in AI (LNAI), 67–72. Springer Verlag.

Calimeri, F.; Cozza, S.; and Ianni, G. 2007. External sources of knowledge and value invention in logic programming. *Annals of Mathematics and Artificial Intelligence* 50(3–4):333–361.

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

Dell'Armi, T.; Faber, W.; Ielpa, G.; Leone, N.; and Pfeifer, G. 2003. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003*, 847–852. Acapulco, Mexico: Morgan Kaufmann Publishers.

Eiter, T.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2004. Combining Answer Set Programming with Description Logics for the Semantic Web. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada*, 141–151. Extended Report RR-1843-03-13, Institut für Informationssysteme, TU Wien, 2003.

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 *Inter-*

national Joint Conference on Artificial Intelligence (IJCAI) 2005, 90–96.

Faber, W.; Pfeifer, G.; Leone, N.; Dell'Armi, T.; and Ielpa, G. 2008. Design and implementation of aggregate functions in the dlv system. *Theory and Practice of Logic Programming* 8(5–6):545–580.

Faber, W.; Leone, N.; and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J. J., and Leite, J., eds., *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in AI (LNAI)*, 200–212. Springer Verlag.

Faber, W.; Leone, N.; and Pfeifer, G. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175(1):278–298. Special Issue: John McCarthy's Legacy.

Ferraris, P. 2005. Answer Sets for Propositional Theories. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings*, volume 3662, 119–131. Springer Verlag.

Liu, L., and Truszczyński, M. 2006. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research* 27:299–334.

Niemelä, I., and Simons, P. 2000. Extending the Smodels System with Cardinality and Weight Constraints. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Dordrecht: Kluwer Academic Publishers. 491–521.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable Model Semantics of Weight Constraint Rules. In Gelfond, M.; Leone, N.; and Pfeifer, G., eds., *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in AI (LNAI)*, 107–116. El Paso, Texas, USA: Springer Verlag.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2007. Well-founded and Stable Semantics of Logic Programs with Aggregates. *Theory and Practice of Logic Programming* 7(3):301–353.

Pelov, N. 2004. *Semantics of Logic Programs with Aggregates*. Ph.D. Dissertation, Katholieke Universiteit Leuven, Leuven, Belgium.

Son, T. C., and Pontelli, E. 2007. A Constructive Semantic Characterization of Aggregates in ASP. *Theory and Practice of Logic Programming* 7:355–375.

A Family of Descriptive Approaches To Preferred Answer Sets

Alexander Šimko
Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava
Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract

In logic programming under the answer set semantics, preferences on rules are used to choose which of the conflicting rules are applied. Many interesting semantics have been proposed. Brewka and Eiter's Principle I expresses the basic intuition behind the preferences. All the approaches that satisfy Principle I introduce a rather imperative feature into otherwise declarative language. They understand preferences as the order, in which the rules of a program have to be applied. In this paper we present two purely declarative approaches for preference handling that satisfy Principle I, and work for general conflicts, including direct and indirect conflicts between rules. The first approach is based on the idea that a rule cannot be defeated by a less preferred conflicting rule. This approach is able to ignore preferences between non-conflicting rules, and, for instance, is equivalent with the answer set semantics for the subclass of stratified programs. It is suitable for the scenarios, when developers do not have full control over preferences. The second approach relaxes the requirement for ignoring conflicting rules, which ensures that it stays in the NP complexity class. It is based on the idea that a rule cannot be defeated by a rule that is less preferred or depends on a less preferred rule. The second approach can be also characterized by a transformation to logic programs without preferences. It turns out that the approaches form a hierarchy, a branch in the hierarchy of the approaches by Delgrande et. al., Wang et. al., and Brewka and Eiter. Finally, we show an application for which the existing approaches are not usable, and the approaches of this paper produce expected results.

Introduction

Preferences on rules are an important knowledge representation concept. In logic programming, one usually writes general rules, and needs to express exceptions. Consider we have the following rules

$$\begin{array}{lrcl} r_1\colon & select(car_1) & \leftarrow & nice(car_1) \\ r_2\colon & \neg select(car_1) & \leftarrow & expensive(car_1) \\ r_3\colon & select(car_1) & \leftarrow & fast(car_1) \end{array}$$

If a car_1 is both nice, expensive, and fast, the rules lead to contradiction. If we have preferences on rules, e.g., we prefer r_1 over r_2, and r_2 over r_3, we can use default negation to express exceptions between rules. Since the rules r_1 and r_3 have the same head, we have to use an auxiliary literal in order to ensure that r_3 does not defeat r_2.

$$\begin{array}{lrcl} r_{1a}\colon & aux & \leftarrow & nice(car_1) \\ r_{1b}\colon & select(car_1) & \leftarrow & select(car_1) \\ r_2\colon & \neg select(car_1) & \leftarrow & expensive(car_1), not\ aux \\ r_3\colon & select(car_1) & \leftarrow & fast(car_1), not\ \neg select(car_1) \end{array}$$

The hand-encoding of preferences has to use auxiliary literals, we have to split rules, and the resulting program is less readable. If the complementary literals are derived via other rules, and the program has hundreds of rules, the hand-encoding becomes even less readable.

More readable way to encode the exceptions between the rules is to make rules mutually exclusive, represent preferences using a relation on rules, and use a semantics for logic programs with preferences, in order to handle preferences.

$$\begin{array}{lrcl} r_1\colon & select(car_1) & \leftarrow & nice(car_1), not\ \neg select(car_1) \\ r_2\colon & \neg select(car_1) & \leftarrow & expensive(car_1), not\ select(car_1) \\ r_3\colon & select(car_1) & \leftarrow & fast(car_1), not\ \neg select(car_1) \end{array}$$

$$r_3 < r_2 < r_1$$

The rules r_1 and r_2 are mutually exclusive: whenever we apply the rule r_1, the rule r_2 is not applicable, and vice versa. We call this mutual exclusivity a conflict. The resulting program is much tolerant to changes. If we decide that the rule r_3 is the most preferred, and r_3 is the least preferred, only the preference relation needs to be changed, and the rules stay intact.

Several semantics for logic programs with preferences on rules have been proposed in the literature. In the first group are semantics that extend the well-founded semantics (Van Gelder, Ross, and Schlipf 1991): (Brewka 1996; Wang, Zhou, and Lin 2000; Schaub and Wang 2002) modify the alternating fixpoint characterization of the well-founded semantics in order to take preferences into account.

In the second group are the semantics that extend the answer set semantics (Gelfond and Lifschitz 1991). Each model of a program with preferences, called a preferred answer set, is guaranteed to be an answer set of the underlying program without preferences. (Brewka and Eiter 1999; Wang, Zhou, and Lin 2000; Delgrande, Schaub, and Tompits 2003) provide prescriptive (Delgrande et al. 2004) semantics, i.e. preferences are understood as the order in which the rules of a program have to be applied. A rule can be defeated only by rules that were applied before it w.r.t. to this order. Each answer set is tested whether it can be con-

structed in aforementioned way. (Zhang and Foo 1997) iteratively non deterministically removes from a program less preferred rules that are defeated by the remainder of the program. (Sakama and Inoue 2000) transforms preferences on rules to preferences on literals, which leads to comparison of the sets of generating rules. Roughly speaking, answer set generated by maximal rules (w.r.t. a preference relation) are selected. (Šefránek 2008) understands preference handling as a kind of argumentation.

Brewka and Eiter have proposed Principle I (Brewka and Eiter 1999) that captures the intuition behind preferences on rules. If two answer sets are generated by the same rules except for two rules, and one rule is preferred over the other, an answer set generated by the less preferred rule should not be preferred.

The existing approaches to preference handling that satisfy Principle I (Brewka and Eiter 1999; Wang, Zhou, and Lin 2000; Delgrande, Schaub, and Tompits 2003), denoted here as $\mathcal{PAS}_{BE}$, $\mathcal{PAS}_{WZL}$ and $\mathcal{PAS}_{DST}$, introduce a rather imperative feature into the otherwise declarative language. They understand preferences on rules as the order in which the rules of a program have to be applied. This, on the one hand goes against declarative spirit of logic programming. On the other hand, it makes the approaches unusable in the situations when we need to automatically generate preferences.

Example 1 *Consider a modified version of the scenario from (Brewka and Eiter 1999). Imagine we have a car recommender system. A program written by the developers of the system contains a database of cars and recommends them to a user.*

r_1: $nice(car_1) \leftarrow$
r_2: $safe(car_2) \leftarrow$

r_3: $rec(car_1) \leftarrow nice(car_1), not\ \neg rec(car_1)$
r_4: $rec(car_2) \leftarrow nice(car_2), not\ \neg rec(car_2)$

The system recommends nice cars to the user. We allow the user to write his/her own rules during the run time of a system. Imagine the user writes the following rules

u_1: $\neg rec(car_2) \leftarrow rec(car_1)$
u_2: $\neg rec(car_1) \leftarrow rec(car_2)$

u_3: $rec(car_1) \leftarrow safe(car_1), not\ \neg rec(car_1)$
u_4: $rec(car_2) \leftarrow safe(car_2), not\ \neg rec(car_2)$

to say that maximally one car should be recommended, and that the user is interested in safe cars.

Due to the rules u_1 and u_2, the rule u_3 is conflicting with r_4: (i) The rule u_1 depends on r_3, and its head is in the negative body of u_4. (ii) The rule u_2 depends on u_4, and its head is in the negative body of r_3. We also have that u_3 is conflicting with u_4, and r_3 is conflicting with r_4 and u_4. All the conflicts are indirect – without the rules u_1 and u_2 there are no conflicts.

The purpose of the user's rules is to override the default behaviour of the system in order to provide the user the best experience possible. Therefore we want the rule u_3 to override r_4, and u_4 to override r_3. Since the u_i rules are only known at the run time, preferences cannot be specified beforehand by the developers of the system. Moreover, we cannot expect a user to know all the r_i rules. It is reasonable to prefer each u_i rule over each r_j rule, and let the semantics to ignore preferences between non-conflicting rules. Hence we have the preferences:

u_1 is preferred over r_1
u_1 is preferred over r_2
...
u_4 is preferred over r_4

The prerequisites $nice(car_2)$ and $safe(car_1)$ of r_4 and u_3 cannot be derived. The only usable conflicting rules are r_3 and u_4. The rule u_4 being preferred, u_4 defines an exception to r_3. We expect u_4 to be applied, and r_3 defeated. The only answer set that uses u_4 is $S = F \cup \{\neg rec(car_1), rec(car_2)\}$ where $F = \{nice(car_1), safe(car_2)\}$. Hence S is the unique expected preferred answer set.

None of the existing approaches satisfying Principle I works as expected. $\mathcal{PAS}_{BE}$ does not handle indirect conflicts, and provides two preferred answer sets S and $S_2 = F \cup \{rec(car_1), \neg rec(car_2)\}$. $\mathcal{PAS}_{DST}$ and $\mathcal{PAS}_{WZL}$ provide no preferred answer set due to they imperative nature. Since u_4 is preferred over r_2, they require that u_4 is applied before r_2. It is impossible as r_2 is the only rule that derives r_4's prerequisite.

It is not crucial for the example that the facts r_1 and r_2 are less preferred. If one feels that they should be separated from the rest of the rules, we can easily modify the program, e.g., by replacing the fact $safe(car_2)$ by the fact $volvo(car_2)$ and the rule $safe(car_2) \leftarrow volvo(car_2)$.

Our goal is to develop an approach to preference handling that (i) is purely declarative, (ii) satisfies Brewka and Eiter's Principle I, and (iii) is usable in the above-mentioned situation.

We have already proposed such a semantics for the case of direct conflicts, and we denote it by $\mathcal{PAS}_D$ (Šimko 2013). We understand this semantics as the reference semantics for the case of direct conflicts, and extend it to the case of general conflicts in this paper.

We present two approaches. The first one, denoted by $\mathcal{PAS}_G$, is based on the intuition that *a rule cannot be defeated by a less preferred (generally) conflicting rule*. The approach is suitable for situations when we need to ignore preferences between non-conflicting rules, and is equivalent to the answer set semantics for the subclass of stratified programs. We consider this property to be important for the aforementioned situations as stratified programs contain no conflicts.

The second approach, denoted $\mathcal{PAS}_{GNO}$, relaxes the requirement for ignoring preferences between non-conflicting rules, and stays is the NP complexity class. There are stratified programs with answer sets and no preferred answer sets according to the approach. The approach is suitable in situations when a developer has a full control over a program. The approach is based on the intuition that *a rule cannot be defeated by a less preferred rule or a rule that depends on a*

less preferred rule. The approach can be also characterized by a transformation from logic programs with preferences to logic programs without preferences such that the answer sets of the transformed program (modulo new special-purpose literals) are the preferred answer sets of an original one.

The two approaches of this paper and our approach for direct conflicts $\mathcal{PAS}_D$ form a hierarchy, which in general does not collapse. Preferred answer sets of $\mathcal{PAS}_{GNO}$ are preferred according to $\mathcal{PAS}_G$, and preferred answer sets of $\mathcal{PAS}_G$ are preferred according to $\mathcal{PAS}_D$.

$\mathcal{PAS}_D$ is thus the reference semantics for the case of direct conflicts. $\mathcal{PAS}_{GNO}$ can be viewed as a computationally acceptable approximation of $\mathcal{PAS}_G$. $\mathcal{PAS}_{GNO}$ is sound w.r.t. $\mathcal{PAS}_G$, but it is not complete w.r.t. $\mathcal{PAS}_G$, meaning that each preferred answer set according to $\mathcal{PAS}_{GNO}$ is a preferred answer set according to $\mathcal{PAS}_G$, but not vice versa.

When dealing with preferences, it is always important to remember what the abstract term "preferences" stands for. Different interpretations of the term lead to different requirements on a semantics. We want to stress that we understand preferences as a mechanism for encoding exceptions between rules in this paper.

The rest of the paper is organized as follows. We first recapitulate preliminaries of logic programming, answer set semantics and our approach to preferred answer sets for direct conflicts $\mathcal{PAS}_D$. Then we provide the two approaches to preferred answer sets for general conflicts. After that we show relation between the approaches of this paper, and also between approaches of this paper and existing approaches. Finally we show how the approaches work on the problematic program from Example 1. Proofs not presented here can be found in the technical report (Šimko 2014).

Preliminaries

In this section, we give preliminaries of logic programming and the answer set semantics. We recapitulate the alternative definition of answer sets based on generating sets from (Šimko 2013), upon which this paper builds.

Syntax

Let At be a set of all atoms. A *literal* is an atom or an expression $\neg a$, where a is an atom. Literals of the form a and $\neg a$ where a is an atom are *complementary*. A *rule* is an expression of the form $l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$, where $0 \leq m \leq n$, and each l_i $(0 \leq i \leq n)$ is a literal. Given a rule r of the above form we use $head(r) = l_0$ to denote the *head* of r, $body(r) = \{l_1, \ldots, not\ l_n\}$ the *body* of r. Moreover, $body^+(r) = \{l_1, \ldots, l_m\}$ denotes the *positive* body of r, and $body^-(r) = \{l_{m+1}, \ldots, l_n\}$ the *negative* body of r. For a set of rules R, $head(R) = \{head(r) : r \in R\}$. A *fact* is a rule with the empty body. A *logic program* is a finite set of rules.

We say that a rule r_1 defeats a rule r_2 iff $head(r_1) \in body^-(r_2)$. A set of rules R defeats a rule r iff $head(R) \cap body^-(r) \neq \emptyset$. A set of rules R_1 defeats a set of rules R_2 iff R defeats a rule $r_2 \in R_2$.

For a set of literals S and a program P we use $G_P(S) = \{r \in P : body^+(r) \subseteq S \text{ and } body^-(r) \cap S = \emptyset\}$.

A *logic program with preferences* is a pair $(P, <)$ where: (i) P is a logic program, and (ii) $<$ is a transitive and asymmetric relation on P. If $r_1 < r_2$ for $r_1, r_2 \in P$ we say that r_2 is *preferred over* r_1.

Answer Set Semantics

A set of literals S is consistent iff $a \in S$ and $\neg a \in S$ holds for no atom a.

A set of rules $R \subseteq P$ *positively satisfies* a logic program P iff for each rule $r \in P$ we have that: If $body^+(r) \subseteq head(R)$, then $r \in R$. We will use $\mathcal{Q}(P)$ to denote the minimal (w.r.t. $\subseteq$) set of rules that positively satisfies P. It contains all the rules from P that can be applied in the iterative manner: we apply a rule which positive body is derived by the rules applied before.

Example 2 *Consider the following program P:*

r_1: $a \leftarrow$
r_2: $b \leftarrow a$
r_3: $d \leftarrow c$

We have that $R_1 = \{r_1, r_2\}$ and $R_2 = \{r_1, r_2, r_3\}$ positively satisfy P. On the other hand $R_3 = \{r_1\}$ doe not positively satisfy P as $body^+(r_2) \subseteq head(R_3)$ and $r_2 \notin R_3$.

We also have that $\mathcal{Q}(P) = R_1$.

The *reduct* P^R of a logic program P w.r.t. a set of rules $R \subseteq P$ is obtained from P by removing each rule r with $head(R) \cap body^-(r) \neq \emptyset$.

A set of rules $R \subseteq P$ is a *generating set* of a logic program P iff $R = \mathcal{Q}(P^R)$.

Definition 1 (Answer set) *A consistent set of literals S is an* answer set *of a logic program P iff there is a generating set R such that $head(R) = S$.*

Example 3 *Consider the following program P*

r_1: $a \leftarrow not\ b$
r_2: $c \leftarrow d, not\ b$
r_3: $b \leftarrow not\ a$

Let $R = \{r_1\}$. When constructing P^R we remove r_3 as $body^-(r_3) \cap head(R) \neq \emptyset$. We get that $P^R = \{r_1, r_2\}$, and $\mathcal{Q}(P^R) = \{r_1\}$. The rule r_2 is not included as $d \in body^+(r_2)$ cannot be derived. We have that $\mathcal{Q}(P^R) = R$. Therefore R is a generating set of P and $\{a\} = head(R)$ is an answer set of P.

It holds that: if a set of rules R is a generating set of a logic program P, and $S = head(R)$ is consistent, then $R = G_P(S)$.

Conflicts

Informally, two rules are conflicting, if their applicability is mutually exclusive: if the application of one rule causes the other rule to be inapplicable, and vice versa. We divide general conflicts into two disjunctive categories:

- direct conflicts, and
- indirect conflicts.

In case of a direct conflict, application of a conflicting rule causes immediately the other rule to be inapplicable.

Definition 2 (Directly Conflicting Rules) *We say that rules r_1 and r_2 are* directly conflicting *iff: (i) r_1 defeats r_2, and (ii) r_2 defeats r_1.*

Example 4 *Consider the following program*

r_1: $a \leftarrow not\ b$
r_2: $b \leftarrow not\ a$

The rules r_1 and r_2 are directly conflicting. If r_1 is used, then r_2 is not applicable, and vice versa.

In case of an indirect conflict, another, intermediate rule, has to be used. The following example illustrated the idea.

Example 5 *Consider the following program*

r_1: $x \leftarrow not\ b$
r_2: $b \leftarrow not\ a$
r_3: $a \leftarrow x$

Now, the rule r_1 is not able to make r_2 inapplicable on its own. The rule r_3 is also needed. Therefore we say that r_1 and r_2 are indirectly conflicting, and the conflict is formed via the rules r_3.

When trying to provide a formal definition of a general conflict, one has to address several difficulties.

First, an indirect conflict is not always effectual. The following example illustrates what we mean by that.

Example 6 *Consider the following program.*

r_1: $x \leftarrow not\ b$
r_2: $b \leftarrow not\ a$
r_3: $a \leftarrow x, not\ y$
r_4: $y \leftarrow$

When the rule r_2 is used, the rule r_1 cannot be used. However, if we use r_1, the rule r_2 is still applicable as the rule r_3 that depends on r_1 and defeats r_2 is defeated by the fact r_4. Note that this cannot happen in the case of direct conflicts.

Second, we need to define that an indirect conflict is formed via rules that are somehow related to a conflicting rule.

Example 7 *Consider the following program:*

r_1: $a \leftarrow not\ b$
r_2: $x \leftarrow not\ a$
r_3: $b \leftarrow$

If we fail to see that r_3 does not depend on r_2, we can come to wrong conviction that r_1 and r_2 are conflicting via r_3 as (i) r_1 defeats r_2, and (ii) r_3 defeats r_1.

Third, in general, the rules depending on a rule are conflicting, thus creating alternatives, in which the rule is/is not conflicting. The following example illustrates this.

Example 8 *Consider the following program:*

r_1: $x \leftarrow not\ c$
r_2: $a \leftarrow x, not\ b$
r_3: $b \leftarrow x, not\ a$
r_4: $c \leftarrow not\ a$

Since the rules r_2 and r_3 are directly conflicting, they cannot be used at the same time. If r_2 is used, r_1 and r_4 are conflicting via r_2. If r_3 is used, r_1 and r_4 are not conflicting.

In this paper we are going to address these issues from a different angle. Instead of defining a general conflict between two rules, we will move to sets of rules and define conflicts between sets of rules in the later sections.

Approach to Direct Conflicts

In this section we recapitulate our semantics for directs conflicts (Šimko 2013), which we generalize in this paper for the case of general conflicts.

We say that a rule r_1 *directly overrides* a rule r_2 w.r.t. a preference relation $<$ iff (i) r_1 and r_2 are directly conflicting, and (ii) $r_2 < r_1$.

The *reduct* $\mathcal{P}^R$ of a logic program with preferences $\mathcal{P} = (P, <)$ w.r.t. a set of rules $R \subseteq P$ is obtained from P by removing each rule $r_1 \in P$, for which there is a rule $r_2 \in R$ such that:

- r_2 defeats r_1, and
- r_1 does not directly override r_2 w.r.t. $<$.

A set of rules $R \subseteq P$ is a *preferred generating set* of a logic program with preferences $\mathcal{P} = (P, <)$ iff $R = \mathcal{Q}(\mathcal{P}^R)$.

A consistent set of literals S is a *preferred answer set* of a logic program with preferences $\mathcal{P}$ iff there is a preferred generating set R of $\mathcal{P}$ such that $head(R) = S$.

We will use $\mathcal{PAS}_D(\mathcal{P})$ to denote the set of all the preferred answer sets of $\mathcal{P}$ according to this definition.

It holds that each preferred generating set of $\mathcal{P} = (P, <)$ is a generating set of P.

Principles

An important direction in preference handling research is the study of principles that a reasonable semantics should satisfy. Brewka and Eiter have proposed first two principles (Brewka and Eiter 1999).

Principle I tries to capture the meaning of preferences. If two answer sets are generated by the same rules except for two rules, the one generated by a less preferred rule is not preferred.

Principle I ((Brewka and Eiter 1999)) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, S_1, S_2 be two answer sets of P. Let $G_P(S_1) = R \cup \{r_1\}$ and $G_P(S_2) = R \cup \{r_2\}$ for $R \subset P$. Let $r_2 < r_1$. Then S_2 is not a preferred answer set of $\mathcal{P}$.*

Principle II says that the preferences specified on a rule with an unsatisfied positive body are irrelevant.

Principle II ((Brewka and Eiter 1999)) *Let S be a preferred answer set of a logic program with preferences $\mathcal{P} = (P, <)$, and r be a rule such that $body^+(r) \not\subseteq S$. Then S is a preferred answer set of a logic program with preferences $\mathcal{P}' = (P', <')$, where $P' = P \cup \{r\}$ and $<' \cap (P \times P) = <$.*

Principle III[1] requires that a program has a preferred answer set whenever a standard answer set of the underlying program exists. It follows the view that the addition of preferences should not cause a consistent program to be inconsistent.

Principle III *Let $\mathcal{P} = (P, <)$ be a logic program with preferences. If P has an answer set, then $\mathcal{P}$ has a preferred answer set.*

Before we proceed, we remind that our approach to preference handling is for general conflicts, and understands preferences on rules as a mechanism for expressing exception between rules. Using this view, we show that Principle II and Principle III should be violated by a semantics, and hence are not relevant under this understanding of preferences.

Example 9 *Consider the following program $\mathcal{P} = (P, <)$*

r_1: $select(a) \leftarrow not\ \neg select(a)$
r_2: $select(b) \leftarrow not\ \neg select(b)$

r_3: $\neg select(a) \leftarrow select(b)$

$r_2 < r_1$

The program is stratified, and has the unique answer set $S = \{\neg select(a), select(b)\}$. Since there are no conflicts between the rules, the unique answer set should be preferred.

We construct $\mathcal{P}' = (P', <)$, $P' = P \cup \{r_4\}$, by adding the rule

r_4: $\neg select(b) \leftarrow select(a)$

We have an indirect conflict between the rules r_1 and r_2 via r_3 and r_4. The rule r_1 being preferred, S should not be a preferred answer set of $\mathcal{P}'$.

Hence Principle II is violated: $body^+(r_4) = \{select(a)\} \not\subseteq S$, but S is not a preferred answer set of $\mathcal{P}'$.

Example 10 *Consider the following program $\mathcal{P} = (P, <)$.*

r_1: $select(a) \leftarrow not\ \neg select(a)$
r_2: $\neg select(a) \leftarrow not\ select(a)$

$r_2 < r_1$

When we interpret preference $r_1 < r_2$ as a way of saying that r_1 defines an exception to r_2 and not vice versa, the program has the following meaning:

r_1: $select(a) \leftarrow$
r_2: $\neg select(a) \leftarrow not\ select(a)$

Hence $S = \{select(a)\}$ is the unique preferred answer set of $\mathcal{P}$.

We construct $\mathcal{P}' = (P', <)$, $P' = P \cup \{r_3\}$, by adding the rule

$r_3 : inc \leftarrow select(a), not\ inc$

[1]It is an idea from Proposition 6.1 from (Brewka and Eiter 1999). Brewka and Eiter did not consider it as a principle. On the other hand (Šefránek 2008) did.

The program $\mathcal{P}'$ has the following meaning:

r_1: $select(a) \leftarrow$
r_2: $\neg select(a) \leftarrow not\ select(a)$
r_3: $inc \leftarrow select(a), not\ inc$

The program has no answer set, and hence $\mathcal{P}'$ has no preferred answer set.

Hence Principle III is violated: The program P' has an answer set, but $\mathcal{P}'$ has no preferred answer set.

Approach One to General Conflicts

In this section we generalize our approach to direct conflicts to the case of general conflicts. As we have already noted, we deliberately avoid defining what a general conflict between two rules is. We will define when two sets of rules are conflicting instead. For this reason we develop an alternative definition of an answer set as a set of sets of rules, upon which the semantics for preferred answer sets will be defined.

Alternative Definition of Answer Sets

A building block of the alternative definition of answer sets is a fragment. The intuition behind a fragment is that it is a set of rules that can form the one hand side of a conflict. The positive bodies of the rules must be supported in a non-cyclic way.

Definition 3 (Fragment) *A set of rules $R \subseteq P$ is a* fragment *of a logic program P iff $\mathcal{Q}(R) = R$.*

Example 11 *Consider the following program P that we will use to illustrate the definitions of this paper.*

r_1: $a \leftarrow x$
r_2: $x \leftarrow not\ b$
r_3: $b \leftarrow not\ a$

The sets $F_1 = \emptyset$, $F_2 = \{r_2\}$, $F_3 = \{r_3\}$, $F_4 = \{r_2, r_1\}$, $F_5 = \{r_2, r_3\}$, $F_6 = \{r_1, r_2, r_3\}$ are all the fragments of the program. For example, $\{r_1\}$ is not a fragment as $\mathcal{Q}(\{r_1\}) = \emptyset$.

Notation 1 *We will denote by $F(P)$ the set of all the fragments of a program P.*

Notation 2 *Let P be a logic program and $E \subseteq F(P)$.*

We will denote $R(E) = \bigcup_{X \in E} X$, and $head(E) = head(R(E))$.

Given a guess of fragments, we define the reduct. Since fragments are sets of rules, we can speak about defeating between fragments.

Definition 4 (Reduct) *Let P be a logic program and $E \subseteq F(P)$.*

The reduct P^E of P w.r.t. E is obtained from $F(P)$ by removing each fragment $X \in F(P)$ for which there is $Y \in E$ that defeats X.

Example 12 (Example 11 continued) *Let $E_1 = \{F_1, F_2, F_4\}$. We have that $P^{E_1} = \{F_1, F_2, F_4\}$. The fragments F_3, F_5, and F_6 are removed as they contain the rule r_3 which is defeated by $F_4 \in E_1$.*

Let $E_2 = \{F_2\}$. We have that $P^{E_2} = \{F_1, F_2, F_3, F_4, F_5, F_6\}$. Since no rule has x in its negative body, no fragment is removed.

A stable fragment set, an alternative notion to the notion of answer set, is a set of fragments that is stable w.r.t. to the reduction.

Definition 5 (Stable fragment set) *A set $E \subseteq F(P)$ is a* stable fragment set *of a program P iff $P^E = E$.*

Example 13 (Example 12 continued) *We have that $P^{E_1} = E_1$, so E_1 is a stable fragment set. On the other hand, E_2 is not a stable fragment set as $P^{E_2} \neq E_2$.*

Proposition 1 *Let P be a logic program, and $E \subseteq F(P)$.*

E is a stable fragment set of P iff $R(E)$ is a generating set of P and $E = \{T : T = \mathcal{Q}(T)$ and $T \subseteq R(E)\}$.

From Proposition 1 we directly have that the following is an alternative definition of answer sets.

Proposition 2 *Let P be a logic program and S a consistent set of literals.*

S is an answer set of P iff there is a stable fragment set E of P such that $head(E) = S$.

Example 14 (Example 13 continued) *$E_1 = \{F_1, F_2, F_4\}$ and $E_3 = \{F_1, F_3\}$ are the only stable fragment sets of the program. The sets $\{a, x\} = head(E_1)$ and $\{b\} = head(E_3)$ are the only answer sets of the program.*

Preferred Answer Sets

In this subsection we develop our first definition of preferred answer sets for general conflicts from the alternative definition of answer sets based on stable fragment sets.

The basic intuition behind the approach is that *a rule cannot be defeated by a less preferred conflicting rule*. This intuition is realized by modifying the definition of reduct. We do not allow a fragment X to be removed because of a fragment Y if Y uses less preferred conflicting rules. For this purpose we use the term "override".

Definition 6 (Conflicting Fragments) *Fragments X and Y are* conflicting *iff (i) X defeats Y, and (ii) Y defeats X.*

Example 15 (Example 11 continued) *Let us recall the fragments: $F_2 = \{r_2\}$, $F_3 = \{r_3\}$, and $F_4 = \{r_2, r_1\}$. The fragments F_3 and F_4 are conflicting as $head(r_3) \in body^-(r_2)$ and $head(r_1) \in body^-(r_3)$. On the other hand, F_2 and F_3 are not conflicting. The fragment F_3 defeats F_2, but not the other way around as $head(r_2) \notin body^-(r_3)$.*

Definition 7 (Override) *Let X and Y be conflicting fragments. We say that X* overrides *Y w.r.t. a preference relation $<$ iff for each $r_1 \in X$ that is defeated by Y, there is $r_2 \in Y$ defeated by X, and $r_2 < r_1$.*

Example 16 (Example 15 continued) *Let us continue with preference $r_2 < r_3$. We have that F_3 overrides F_4 and F_3 overrides F_6. On the other hand F_3 does not override F_2 because F_2 does not defeat F_3. From the following Proposition 3 we also have that F_6 does not override F_6.*

Proposition 3 *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, X and Y be fragments of P.*

If X overrides Y w.r.t. $<$, then Y does not override X w.r.t. $<$.

When constructing the reduct w.r.t. a guess, a fragment X cannot be removed because of a fragment Y which is overridden by X.

Definition 8 (Reduct) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and $E \subseteq F(P)$.*

The reduct $\mathcal{P}^E$ of $\mathcal{P}$ w.r.t. E is obtained from $F(P)$ by removing each $X \in F(P)$ such that there is $Y \in E$ that:

- *Y defeats X, and*
- *X does not override Y w.r.t. $<$.*

Example 17 (Example 16 continued) *Let $E_1 = \{F_1, F_2, F_4\}$. We have that $\mathcal{P}^{E_1} = \{F_1, F_2, F_3, F_4\}$. Now, the fragment F_3 is not removed as the only fragment from E_1 that defeats it is F_4, but F_3 overrides F_4.*

Definition 9 (Preferred stable fragment set) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences., and $E \subseteq F(P)$.*

We say that E is a preferred stable fragment set *of $\mathcal{P}$ iff $\mathcal{P}^E = E$.*

Example 18 (Example 16 continued) *Now we have that $\mathcal{P}^{E_1} \neq E_1$, so E_1 is not a preferred stable fragment set. On the other hand, $E_3 = \{F_1, F_3\}$ is a preferred stable fragment set as $\mathcal{P}^{E_3} = E_3$.*

Definition 10 (Preferred answer set) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and S be a consistent set of literals.*

S is a preferred answer set *of $\mathcal{P}$ iff there is a preferred stable fragment set E of $\mathcal{P}$ such that $head(E) = S$.*

We will use $\mathcal{PAS}_G(\mathcal{P})$ to denote the set of all the preferred answer sets of $\mathcal{P}$ according to this definition.

Example 19 (Example 18 continued) *The set $E_3 = \{F_1, F_3\}$ is the only preferred stable fragment set, and $\{b\} = head(E_3)$ is the only preferred answer set of the program.*

Proposition 4 *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and $E \subseteq F(P)$.*

If E is a preferred stable fragment set of $\mathcal{P}$, then E is a stable fragment set of P.

Properties

Preferred answer sets as defined in Definition 10 enjoy following nice properties.

Proposition 5 *Let $\mathcal{P} = (P, <)$ be a logic program with preferences. Then $\mathcal{PAS}_G(\mathcal{P}) \subseteq \mathcal{AS}(P)$.*

Proposition 6 *Let $\mathcal{P} = (P, \emptyset)$ be a logic program with preferences. Then $\mathcal{PAS}_G(\mathcal{P}) = \mathcal{AS}(P)$.*

Proposition 7 *Preferred answer sets as defined in Definition 10 satisfy Principle I.*

Proposition 8 *Let $\mathcal{P}_1 = (P, <_1)$, $\mathcal{P}_2 = (P, <_2)$ be logic programs with preferences such that $<_1 \subseteq <_2$.*

Then $\mathcal{PAS}_G(\mathcal{P}_2) \subseteq \mathcal{PAS}_G(\mathcal{P}_1)$.

On the subclass of stratified programs, the semantics is equivalent to the answer set semantics. We consider this property to be an important one as stratified programs contain no conflicts.

Proposition 9 *Let $\mathcal{P} = (P, <)$ be a logic program with preferences such that P is stratified. Then $\mathcal{PAS}_G(\mathcal{P}) = \mathcal{AS}(P)$.*

The following example illustrates how the approach works on stratified programs.

Example 20 *Consider a problematic program from (Brewka and Eiter 1999):*

r_1: $a \leftarrow not\ b$
r_2: $b \leftarrow$

$r_2 < r_1$

The program is stratified and has a unique answer set $S = \{b\}$.

The program has the following fragments $F_0 = \emptyset$, $F_1 = \{r_1\}$, $F_2 = \{r_2\}$, $F_3 = \{r_1, r_2\}$. The set $E = \{F_0, F_2\}$ is a unique stable fragment set.

We have that F_2 defeats both F_1, and F_3. Neither F_1 nor F_3 override F_2 as they are not conflicting with F_2. This is the reason why preference $r_2 < r_1$ is ignored here, and both F_1 and F_3 are removed during the reduction: $\mathcal{P}^E = \{F_0, F_2\} = E$. Therefore S is a unique preferred answer set.

From the computational complexity point of view, so far, we have established only the upper bound. Establishing the lower bound remains among open problems for future work.

Proposition 10 *Given a logic program with preferences $\mathcal{P}$, deciding whether $\mathcal{P}$ has a preferred answer set is in Σ_3^P.*

Approach Two to General Conflicts

If we have an application domain, where we can relax the requirements for preference handling in a sense that we no longer require preferences between non-conflicting rules to be ignored, we can ensure that the semantics stays in the NP complexity class.

In this section we simplify our first approach by using the following intuition for preference handling: *a rule cannot be defeated by a less preferred rule or a rule depending on a less preferred rule*.

The definition of the approach follows the structure of our approach for direct conflicts. The presented intuition is realized using a set T_r^R in the definition of reduct.

Definition 11 (Reduct) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and $R \subseteq P$ be a set of rules.*

The reduct $\mathcal{P}^R$ *of $\mathcal{P}$ w.r.t. R is obtained from P by removing each rule $r \in P$ such that $body^-(r) \cap head(T_r^R) \neq \emptyset$, where $T_r^R = \mathcal{Q}(\{p \in R : p \not< r\})$.*

Example 21 (Example 16 continued) *Let us recall the program:*

r_1: $a \leftarrow x$
r_2: $x \leftarrow not\ b$
r_3: $b \leftarrow not\ a$

$r_2 < r_3$

Let $R_1 = \{r_1, r_2\}$. We have that $T_{r_1}^{R_1} = R_1$, $T_{r_2}^{R_1} = R_1$. On the other hand $T_{r_3}^{R_1} = \emptyset$ as $r_2 < r_3$ and r_1 depends on r_2. No rule less preferred, and no rule that depends on a rule less preferred than r_3 can be used to defeat r_3. In this case no rule can defeat r_3.

Hence $\mathcal{P}^{R_1} = \{r_1, r_2, r_3\}$.

Definition 12 (Preferred generating set) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and R be a generating set of P.*

We say that R is a preferred generating set *of $\mathcal{P}$ iff $R = \mathcal{Q}(\mathcal{P}^R)$.*

Example 22 (Example 21 continued) *We have that $\mathcal{Q}(\mathcal{P}^{R_1}) = P \neq R_1$. Hence R_1 is not a preferred generating set.*

Definition 13 (Preferred answer set) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences, and S be a consistent set of literals.*

S is a preferred answer set of $\mathcal{P}$ iff there is a preferred generating set R such that $S = head(R)$.

We will use $\mathcal{PAS}_{GNO}(\mathcal{P})$ to denote the set of all the preferred answer sets of $\mathcal{P}$ according to this definition.

Example 23 (Example 22 continued) *The set $R_2 = \{r_3\}$ is the only preferred generating set, and $\{b\} = head(R_2)$ is the only preferred answer set.*

Transformation

It turns out that the second approach can be characterized by a transformation from programs with preferences to programs without preferences in a way that the answer sets of the transformed program correspond (modulo new special-purpose literals) to the preferred answer sets of an original program.

The idea of the transformation is to use special-purpose literals and auxiliary rules in order to allow a rule r to be defeated only by T_r^R where R is a preferred generating set guess. We first present the definition of the transformation and then explain each rule.

Notation 3 *If r is a rule of a program P, then n_r denotes a new literal not occurring in P.*

If r is a rule of a program P, and x is a literal of P, then x^r denotes a new literal not occurring in P and different from n_q for each $q \in P$. For a set of literals S, S^r denotes $\{x^r : x \in S\}$.

We will also use inc to denote a literal not occurring in P and different from all previously mentioned literals.

Definition 14 (Transformation) *Let $\mathcal{P} = (P, <)$ be a logic program with preferences.*

Let r be a rule. Then $t_{\mathcal{P}}(r)$ is the set of the rules

$$head(r) \leftarrow n_r \quad (1)$$
$$n_r \leftarrow body^+(r), not\ body^-(r)^r \quad (2)$$

and the rule

$$head(p)^r \leftarrow body^+(p)^r, n_p \quad (3)$$

for each $p \in P$ such that $p \not< r$, and the rule

$$inc \leftarrow n_r, x, not\ inc \quad (4)$$

for each $x \in body^-(r)$.
$t(\mathcal{P}) = \bigcup_{r \in P} t_{\mathcal{P}}(r)$.

A preferred generating set guess R is encoded using n_r literals. The meaning of a literal n_r is that a rule r was applied. In order to derive n_r literals, we split each rule r of a program into two rules: The rule (2) derives literal n_r, and the rule (1) derives the head of the original rule r.

The special-purpose literals x^r are used in the negative body of the rule (2) in order to ensure that only T_r^R can defeat a rule r. The x^r literals are derived using the rules of the form (3).

The rules of the form (4) ensure that no answer set of $t(\mathcal{P})$ contains both n_r and x. This condition is needed in order to ensure that R is also a generating set.

Example 24 *Consider again our running program* $\mathcal{P}$*:*

r_1: $a \leftarrow x$
r_2: $x \leftarrow not\ b$
r_3: $b \leftarrow not\ a$

$r_2 < r_3$

$t(\mathcal{P})$ *is as follows:*

$a \leftarrow n_{r_1}$ $\quad x \leftarrow n_{r_2}$ $\quad b \leftarrow n_{r_3}$
$n_{r_1} \leftarrow x$ $\quad n_{r_2} \leftarrow not\ b^{r_2}$ $\quad n_{r_3} \leftarrow not\ a^{r_3}$

$a^{r_1} \leftarrow x^{r_1}, n_{r_1}$ $\quad a^{r_2} \leftarrow x^{r_2}, n_{r_1}$ $\quad a^{r_3} \leftarrow x^{r_3}, n_{r_1}$
$x^{r_1} \leftarrow n_{r_2}$ $\quad x^{r_2} \leftarrow n_{r_2}$
$b^{r_1} \leftarrow n_{r_3}$ $\quad b^{r_2} \leftarrow n_{r_3}$ $\quad b^{r_3} \leftarrow n_{r_3}$

$inc \leftarrow n_{r_2}, b, not\ inc$
$inc \leftarrow n_{r_3}, a, not\ inc$

Now, as $r_2 < r_3$*, a transformed rule deriving* x^{r_3} *coming from* r_2 *is not included.*

The transformation captures the semantics of preferred answer sets as defined in Definition 13.

Proposition 11 *Let* $\mathcal{P} = (P, <)$ *be a logic program with preferences. Let* Lit *be a set of all the literals constructed from the atoms of* P*, and* $N_{\mathcal{P}}(S) = \{n_r : r \in G_P(S)\}$*, and* $Aux(S) = \bigcup_{r \in P} head(T_r^R)^r$*, where* $R = G_P(S)$.

If S *is a preferred answer set of* $\mathcal{P}$*, then* $A = S \cup N_{\mathcal{P}}(S) \cup Aux(S)$ *is an answer set of* $t(\mathcal{P})$.

If A *is an answer set of* $t(\mathcal{P})$*, then* $S = A \cap Lit$ *is a preferred answer set of* $\mathcal{P}$*, and* $A = S \cup N_{\mathcal{P}}(S) \cup Aux(S)$.

Properties

Preferred answer sets as defined in Definition 13 enjoy several nice properties.

Proposition 12 *Let* $\mathcal{P} = (P, <)$ *be a logic program with preferences. Then* $\mathcal{PAS}_{GNO}(\mathcal{P}) \subseteq \mathcal{AS}(P)$.

Proposition 13 *Let* $\mathcal{P} = (P, \emptyset)$ *be a logic program with preferences. Then* $\mathcal{PAS}_{GNO}(\mathcal{P}) = \mathcal{AS}(P)$.

Proposition 14 *Preferred answer sets as defined in Definition 13 satisfy Principle I.*

Proposition 15 *Let* $\mathcal{P}_1 = (P, <_1)$ *and* $\mathcal{P}_2 = (P, <_2)$ *be logic programs with preferences such that* $<_1 \subseteq <_2$*. Then* $\mathcal{PAS}_{GNO}(\mathcal{P}_2) \subseteq \mathcal{PAS}_{GNO}(\mathcal{P}_1)$.

The approach two is not equivalent to the answer set semantics for the subclass of stratified programs.

Proposition 16 *There is a logic program with preferences* $\mathcal{P} = (P, <)$ *where* P *is stratified and* $\mathcal{PAS}_{GNO}(\mathcal{P}) = \emptyset$.

Example 25 shows such a program. Example 20 and 25 illustrate the main difference between the two approaches. While $\mathcal{PAS}_G$ ignores preferences between non-conflicting rules, $\mathcal{PAS}_{GNO}$ is not always able to do so.

Example 25 *Consider again the program from Example 20:*

r_1: $a \leftarrow not\ b$
r_2: $b \leftarrow$

$r_2 < r_1$

The program is stratified and has a unique answer set $S = \{b\}$*. A unique generating set* $R = \{r_2\}$ *corresponds to the answer set* S.

We have that $T_{r_1}^R = \emptyset$*. The rule* r_2 *is not included as* $r_2 < r_1$*. Due to a simplicity of the approach, preference* $r_2 < r_1$ *is not ignored. Hence* $head(T_{r_1}^R) \cap body^-(r_1) = \emptyset$*, and* $r_1 \in \mathcal{P}^R$*. From that* $\mathcal{Q}(\mathcal{P}^R) \neq R$*, and* S *is not a preferred answer set.*

On the other hand the approach stays in the NP complexity class.

Proposition 17 *Deciding whether* $\mathcal{PAS}_{GNO}(\mathcal{P}) \neq \emptyset$ *for a logic program with preferences* $\mathcal{P}$ *is NP-complete.*

Proof: Membership: Using Proposition 11, we can reduce the decision problem $\mathcal{PAS}_{GNO}(\mathcal{P}) \neq \emptyset$ to the problem $\mathcal{AS}(t(\mathcal{P})) \neq \emptyset$ (in polynomial time), which is in NP. Hardness: Deciding $\mathcal{AS}(P) \neq \emptyset$ for a program P is NP-complete. Using Proposition 13 we can reduce it to the decision $\mathcal{PAS}_{GNO}((P, \emptyset)) \neq \emptyset$.

Relation between the Approaches of this Paper

It turns out that the approaches of this paper form a hierarchy, which does not collapse.

Notation 4 *Let* A *and* B *be names of semantics.*

We write $A \subseteq B$ *iff each preferred answer set according to* A *is a preferred answer set according to* B.

We write $A = B$ *iff* $A \subseteq B$ *and* $B \subseteq A$.

Proposition 18 $\mathcal{PAS}_{GNO} \subseteq \mathcal{PAS}_G \subseteq \mathcal{PAS}_D$

Proposition 19 $\mathcal{PAS}_D \not\subseteq \mathcal{PAS}_G$

Proposition 20 $\mathcal{PAS}_G \not\subseteq \mathcal{PAS}_{GNO}$

We interpret the results as follows. The semantics $\mathcal{PAS}_D$ is the reference semantics for the case of direct conflicts. The semantics $\mathcal{PAS}_{GNO}$ and $\mathcal{PAS}_G$ extend the semantics to the case of indirect conflicts. The semantics $\mathcal{PAS}_G$ ignores preferences between non-conflicting rules, e.g. it is equivalent to the answer set semantics for the subclass

of stratified programs (Stratified programs contain no conflicts). If an application domain allows it, we can drop the requirement for ignoring preferences between non-conflicting rules and use the semantics $\mathcal{PAS}_{GNO}$ that stays in the NP complexity class. The semantics $\mathcal{PAS}_{GNO}$ is sound w.r.t. $\mathcal{PAS}_G$ but it is not complete w.r.t. $\mathcal{PAS}_G$. Some preferred answer sets according to $\mathcal{PAS}_G$ are not preferred according to $\mathcal{PAS}_{GNO}$ due to preferences between non-conflicting rules.

Relation to Existing Approaches

Schaub and Wang (Schaub and Wang 2003) have shown that the approaches (Delgrande, Schaub, and Tompits 2003; Wang, Zhou, and Lin 2000; Brewka and Eiter 1999), referred here as $\mathcal{PAS}_{DST}$, $\mathcal{PAS}_{WZL}$, $\mathcal{PAS}_{BE}$ form a hierarchy.

Proposition 21 ((Schaub and Wang 2003)) $\mathcal{PAS}_{DST} \subseteq \mathcal{PAS}_{WZL} \subseteq \mathcal{PAS}_{BE}$

We have shown that our approach for direct conflicts continues in this hierarchy (Šimko 2013).

Proposition 22 ((Šimko 2013)) $\mathcal{PAS}_{BE} \subseteq \mathcal{PAS}_D$

The relations $\mathcal{PAS}_{DST} \subseteq \mathcal{PAS}_{GNO}$ and $\mathcal{PAS}_{WZL} \subseteq \mathcal{PAS}_G$ are the only subset relation between our semantics for general conflicts $\mathcal{PAS}_{GNO}$, $\mathcal{PAS}_G$ and $\mathcal{PAS}_{DST}$, $\mathcal{PAS}_{WZL}$ and $\mathcal{PAS}_{BE}$.

Proposition 23 $\mathcal{PAS}_{DST} \subseteq \mathcal{PAS}_{GNO}$.

Proposition 24 $\mathcal{PAS}_{WZL} \subseteq \mathcal{PAS}_G$.

Proposition 25 $\mathcal{PAS}_{GNO} \not\subseteq \mathcal{PAS}_{BE}$.

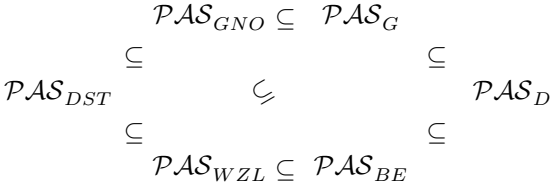
Corollary 1

- $\mathcal{PAS}_{GNO} \not\subseteq \mathcal{PAS}_{WZL}$, $\mathcal{PAS}_{GNO} \not\subseteq \mathcal{PAS}_{DST}$,
- $\mathcal{PAS}_G \not\subseteq \mathcal{PAS}_{BE}$, $\mathcal{PAS}_G \not\subseteq \mathcal{PAS}_{WZL}$, $\mathcal{PAS}_G \not\subseteq \mathcal{PAS}_{DST}$.

Proposition 26 $\mathcal{PAS}_{WZL} \not\subseteq \mathcal{PAS}_{GNO}$

The overall hierarchy of the approaches is depicted in Figure 1.

Figure 1: The hierarchy of the approaches.



An Example

In this section we show that the approaches of this paper handle correctly the program of Example 1 from Introduction. We remind that neither of the approaches $\mathcal{PAS}_{DST}$, $\mathcal{PAS}_{WZL}$ and $\mathcal{PAS}_{BE}$ provides intended preferred answer sets.

Example 26 *We recall the program:*

r_1: $nice(car_1) \leftarrow$
r_2: $safe(car_2) \leftarrow$

r_3: $rec(car_1) \leftarrow nice(car_1), not\ \neg rec(car_1)$
r_4: $rec(car_2) \leftarrow nice(car_2), not\ \neg rec(car_2)$

u_1: $\neg rec(car_2) \leftarrow rec(car_1)$
u_2: $\neg rec(car_1) \leftarrow rec(car_2)$

u_3: $rec(car_1) \leftarrow safe(car_1), not\ \neg rec(car_1)$
u_4: $rec(car_2) \leftarrow safe(car_2), not\ \neg rec(car_2)$

$r_i < u_j$ *for each* i *and* j.

The program has two answer sets $S_1 = \{rec(car_1), \neg rec(car_2)\} \cup F$ *and* $S_2 = \{\neg rec(car_1), rec(car_2)\} \cup F$ *where* $F = \{nice(car_1), safe(car_2)\}$. *As we mentioned in Introduction,* S_2 *is the intended unique preferred answer set.*

$\mathcal{PAS}_G$ *: We start by listing fragments of the program. We denote by* F_i *fragments formed by the facts. Let* $F_0 = \emptyset$, $F_1 = \{r_1\}$, $F_2 = \{r_2\}$, $F_3 = \{r_1, r_2\}$.

The rules r_3 *and* u_4 *are conflicting. We denote by* A_i *fragments containing the rule* r_3*:* $A_1 = \{r_1, r_3\}$, $A_2 = \{r_1, r_3, u_1\}$, $A_3 = \{r_1, r_2, r_3\}$, $A_4 = \{r_1, r_2, r_3, u_1\}$.

We denote by B_i *fragments containing the rule* u_4. *Let* $B_1 = \{r_2, u_4\}$, $B_2 = \{r_2, u_4, u_2\}$, $B_3 = \{r_1, r_2, u_4\}$, $B_4 = \{r_1, r_2, u_4, u_2\}$.

A stable fragment set $E_1 = \{F_0, F_1, F_2, F_3, A_1, A_2, A_3, A_4\}$ *corresponds to the answer set* S_1 *and a stable fragment set* $E_2 = \{F_0, F_1, F_2, F_3, B_1, B_2, B_3, B_4\}$ *corresponds to the answer set* S_2.

We have that B_3 *overrides both* A_2 *and* A_4. *Hence* $B_3 \in \mathcal{P}^{E_1}$, *and* $\mathcal{P}^{E_1} \neq E_1$. *Hence* S_1 *is not a preferred answer set.*

On the other hand $E_2 = \mathcal{P}^{E_2}$, *and* S_2 *is a preferred answer set.*

$\mathcal{PAS}_{GNO}$ *: A generating set* $R_1 = \{r_1, r_2, r_3, u_1\}$ *corresponds to the answer set* S_1, *and* $R_2 = \{r_1, r_2, u_4, u_2\}$ *corresponds to the answer set* S_2.

We have that $T_{u_4}^{R_1} = \{u_1\}$. *The rules* r_1, r_2, r_3 *are not included as they are less preferred that* u_4. *Hence* $body^-(u_4) \cap head(T_{u_4}^{R_1}) = \emptyset$. *Therefore* u_4 *cannot be defeated, i.e.* $u_4 \in \mathcal{P}^{R_1}$. *Hence* $R_1 \neq \mathcal{Q}(\mathcal{P}^{R_1})$, *and the answer set* S_1 *is not a preferred answer set.*

On the other hand $R_2 = \mathcal{Q}(\mathcal{P}^{R_2})$, *and the answer set* S_2 *is a preferred answer set.*

Conclusions

When dealing with preferences it is always important to remember what the abstract term "preferences" represents. In this paper we understand preferences as a mechanism for encoding exceptions. In case of conflicting rules, the preferred

rules define exceptions to less preferred ones, and not the other way around. For this interpretation of preferences, it is important that a semantics for preferred answer sets satisfies Brewka and Eiter's Principle I. All the existing approaches for logic programming with preferences on rules that satisfy the principle introduce an imperative feature into the language. Preferences are understood as the order in which the rules of a program are applied.

The goal of this paper was to develop a purely declarative approach to preference handling satisfying Principle I. We have developed two approaches $\mathcal{PAS}_G$ and $\mathcal{PAS}_{GNO}$. The first one is able to ignore preferences between non-conflicting rules. For example, it is equivalent with the answer set semantics on stratified programs. It is designed for situations, where developer does not have full control over preferences. An example is a situation where a user is able to write his/her own rules in order to override developer's rules. If the user's rules are not known until run-time of the system, we have to prefer all the user's rules over the developer's rules. To the best of our knowledge, no existing approach for logic programming with preferences satisfying Principle I is usable in this situation. On the other hand, in situations where we can drop the requirement for ignoring preferences between non-conflicting rules, e.g. if a developer has full control over the program, we can use $\mathcal{PAS}_{GNO}$ which is in the NP complexity class. Naturally, since the requirement for ignoring preferences between non-conflicting rules was dropped, there are stratified programs with answer sets and no preferred answer sets according to $\mathcal{PAS}_{GNO}$.

The two presented approaches are not independent. They form a hierarchy, a branch in the hierarchy of the approaches $\mathcal{PAS}_{DST}$, $\mathcal{PAS}_{WZL}$, $\mathcal{PAS}_{BE}$ and $\mathcal{PAS}_D$.

One of our future goals is to better understand the complexity of the decision problem $\mathcal{PAS}_G(\mathcal{P}) \neq \emptyset$. So far, we have Σ_3^P membership result. It is not immediately clear whether the problem is also Σ_3^P hard.

We also plan to investigate relation between $\mathcal{PAS}_G$ and argumentation, and to implement a prototype solver for the semantics using a meta-interpretation technique of (Eiter et al. 2003).

Acknowledgments

We would like to thank the anonymous reviewers for detailed and useful comments. This work was supported by the grant UK/276/2013 of Comenius University in Bratislava and 1/1333/12 of VEGA.

References

Brewka, G., and Eiter, T. 1999. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence* 109(1-2):297–356.

Brewka, G. 1996. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research* 4:19–36.

Delgrande, J. P.; Schaub, T.; Tompits, H.; and Wang, K. 2004. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence* 20(2):308–334.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2003. A Framework for Compiling Preferences in Logic Programs. *Theoretical Computer Science* 3(2):129–187.

Eiter, T.; Faber, W.; Leone, N.; and Pfeifer, G. 2003. Computing Preferred Answer Sets by Meta-Interpretation in Answer Set Programming. *Theoretical Computer Science* 3(4-5):463–498.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3-4):365–386.

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123(1-2):185–222.

Schaub, T., and Wang, K. 2002. Preferred well-founded semantics for logic programming by alternating fixpoints: Preliminary Report. In *9th International Workshop on Non-Monotonic Reasoning*, 238–246.

Schaub, T., and Wang, K. 2003. A semantic framework for preference handling in answer set programming. *Theoretical Computer Science* 3(4-5):569–607.

Šefránek, J. 2008. Preferred answer sets supported by arguments. In *Proceedings of Twelfth International Workshop on Non-Monotonic Reasoning*.

Šimko, A. 2013. Extension of Gelfond-Lifschitz Reduction for Preferred Answer Sets : Preliminary Report. In *Proceedings of 27th Workshop on Logic Programming (WLP2013)*, 2–16.

Šimko, A. 2014. Proofs for the Approaches to Preferred Answer Sets with General Conflicts. Technical report, Department of Applied Informatics, Comenius University in Bratislava. `http://dai.fmph.uniba.sk/~simko/nmr2014_proofs.pdf`.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The Well-founded Semantics for General Logic Programs. *Journal of the ACM*.

Wang, K.; Zhou, L.; and Lin, F. 2000. Alternating Fixpoint Theory for Logic Programs with Priority. In *Proceedings of the First International Conference on Computational Logic*, 164–178.

Zhang, Y., and Foo, N. Y. 1997. Answer Sets for Prioritized Logic Programs. In *Proceedings of the 1998 International Logic Programming Symposium*, 69–83.

KR^3: An Architecture for Knowledge Representation and Reasoning in Robotics

Shiqi Zhang
Department of Computer Science
Texas Tech University, USA
shiqi.zhang6@gmail.com

Mohan Sridharan
Department of Computer Science
Texas Tech University, USA
mohan.sridharan@ttu.edu

Michael Gelfond
Department of Computer Science
Texas Tech University, USA
michael.gelfond@ttu.edu

Jeremy Wyatt
School of Computer Science
University of Birmingham, UK
jlw@cs.bham.ac.uk

Abstract

This paper describes an architecture that combines the complementary strengths of declarative programming and probabilistic graphical models to enable robots to represent, reason with, and learn from, qualitative and quantitative descriptions of uncertainty and knowledge. An action language is used for the low-level (LL) and high-level (HL) system descriptions in the architecture, and the definition of recorded histories in the HL is expanded to allow prioritized defaults. For any given goal, tentative plans created in the HL using default knowledge and commonsense reasoning are implemented in the LL using probabilistic algorithms, with the corresponding observations used to update the HL history. Tight coupling between the two levels enables automatic selection of relevant variables and generation of suitable action policies in the LL for each HL action, and supports reasoning with violation of defaults, noisy observations and unreliable actions in large and complex domains. The architecture is evaluated in simulation and on physical robots transporting objects in indoor domains; the benefit on robots is a reduction in task execution time of 39% compared with a purely probabilistic, but still hierarchical, approach.

1 Introduction

Mobile robots deployed in complex domains receive far more raw data from sensors than is possible to process in real-time, and may have incomplete domain knowledge. Furthermore, the descriptions of knowledge and uncertainty obtained from different sources may complement or contradict each other, and may have different degrees of relevance to current or future tasks. Widespread use of robots thus poses fundamental knowledge representation and reasoning challenges—robots need to represent, learn from, and reason with, qualitative and quantitative descriptions of knowledge and uncertainty. Towards this objective, our architecture combines the knowledge representation and non-monotonic logical reasoning capabilities of declarative programming with the uncertainty modeling capabilities of probabilistic graphical models. The architecture consists of two tightly coupled levels and has the following key features:

1. An action language is used for the HL and LL system descriptions and the definition of recorded history is expanded in the HL to allow prioritized defaults.
2. For any assigned objective, tentative plans are created in the HL using default knowledge and commonsense reasoning, and implemented in the LL using probabilistic algorithms, with the corresponding observations adding suitable statements to the HL history.
3. For each HL action, abstraction and tight coupling between the LL and HL system descriptions enables automatic selection of relevant variables and generation of a suitable action policy in the LL.

In this paper, the HL domain representation is translated into an Answer Set Prolog (ASP) program, while the LL domain representation is translated into partially observable Markov decision processes (POMDPs). The novel contributions of the architecture, e.g., allowing histories with prioritized defaults, tight coupling between the two levels, and the resultant automatic selection of the relevant variables in the LL, support reasoning with violation of defaults, noisy observations and unreliable actions in large and complex domains. The architecture is grounded and evaluated in simulation and on physical robots moving objects in indoor domains.

2 Related Work

Probabilistic graphical models such as POMDPs have been used to represent knowledge and plan sensing, navigation and interaction for robots (Hoey et al. 2010; Rosenthal and Veloso 2012). However, these formulations (by themselves) make it difficult to perform commonsense reasoning, e.g., default reasoning and non-monotonic logical reasoning, especially with information not directly relevant to tasks at hand. In parallel, research in classical planning has provided many algorithms for knowledge representation and logical reasoning (Ghallab, Nau, and Traverso 2004), but these algorithms require substantial prior knowledge about the domain, task and the set of actions. Many of these algorithms also do not support merging of new, unreliable information from sensors and humans with the cur-

rent beliefs in a knowledge base. Answer Set Programming (ASP), a non-monotonic logic programming paradigm, is well-suited for representing and reasoning with commonsense knowledge (Gelfond 2008; Baral 2003). An international research community has been built around ASP, with applications such as reasoning in simulated robot housekeepers and for representing knowledge extracted from natural language human-robot interaction (Chen et al. 2012; Erdem, Aker, and Patoglu 2012). However, ASP does not support probabilistic analysis, whereas a lot of information available to robots is represented probabilistically to quantitatively model the uncertainty in sensor input processing and actuation in the real world.

Researchers have designed cognitive architectures (Laird, Newell, and Rosenbloom 1987; Langley and Choi 2006; Talamadupula et al. 2010), and developed algorithms that combine deterministic and probabilistic algorithms for task and motion planning on robots (Kaelbling and Lozano-Perez 2013; Hanheide et al. 2011). Recent work has also integrated ASP and POMDPs for non-monotonic logical inference and probabilistic planning on robots (Zhang, Sridharan, and Bao 2012). Some examples of principled algorithms developed to combine logical and probabilistic reasoning include probabilistic first-order logic (Halpern 2003), first-order relational POMDPs (Sanner and Kersting 2010), Markov logic network (Richardson and Domingos 2006), Bayesian logic (Milch et al. 2006), and a probabilistic extension to ASP (Baral, Gelfond, and Rushton 2009). However, algorithms based on first-order logic for probabilistically modeling uncertainty do not provide the desired expressiveness for capabilities such as default reasoning, e.g., it is not always possible to express uncertainty and degrees of belief quantitatively. Other algorithms based on logic programming that support probabilistic reasoning do not support one or more of the desired capabilities: reasoning as in causal Bayesian networks; incremental addition of probabilistic information; reasoning with large probabilistic components; and dynamic addition of variables with different ranges (Baral, Gelfond, and Rushton 2009). The architecture described in this paper is a step towards achieving these capabilities. It exploits the complementary strengths of declarative programming and probabilistic graphical models to represent, reason with, and learn from qualitative and quantitative descriptions of knowledge and uncertainty, enabling robots to automatically plan sensing and actuation in larger domains than was possible before.

3 KRR Architecture

This section describes our architecture's HL and LL domain representations. The syntax, semantics and representation of the corresponding transition diagrams are described in an *action language* AL (Gelfond and Kahl 2014). Action languages are formal models of parts of natural language used for describing transition diagrams. AL has a sorted signature containing three *sorts*: `statics`, `fluents` and `actions`. Statics are domain properties whose truth values cannot be changed by actions, while fluents are properties whose truth values are changed by actions. Actions are defined as a set of elementary actions that can be executed in parallel. A domain property `p` or its negation $\neg p$ is a domain literal. AL allows three types of statements:

a **causes** l_{in} **if** $p_0,\ldots,p_m$ (Causal law)
l **if** $p_0,\ldots,p_m$ (State constraint)
impossible $a_0,\ldots,a_k$ **if** $p_0,\ldots,p_m$ (Executability condition)

where a is an action, l is a literal, l_{in} is a inertial fluent literal, and $p_0,\ldots,p_m$ are domain literals. The causal law states, for instance, that action a causes inertial fluent literal l_{in} if the literals $p_0,\ldots,p_m$ hold true. A collection of statements of AL forms a system/domain description.

As an illustrative example used throughout this paper, we will consider a robot that has to move objects to specific places in an indoor domain. The domain contains four specific places: *office*, *main_library*, *aux_library*, and *kitchen*, and a number of specific objects of the sorts: *textbook*, *printer* and *kitchenware*.

3.1 HL domain representation

The HL domain representation consists of a system description $\mathscr{D}_H$ and histories with defaults $\mathscr{H}$. $\mathscr{D}_H$ consists of a sorted signature and axioms used to describe the HL transition diagram τ_H. The sorted signature: $\Sigma_H = \langle \mathscr{O}, \mathscr{F}, \mathscr{P} \rangle$ is a tuple that defines the names of objects, functions, and predicates available for use in the HL. The sorts in our example are: `place`, `thing`, `robot`, and `object`; `object` and `robot` are subsorts of `thing`. Robots can move on their own, but objects cannot move on their own. The sort `object` has subsorts such as `textbook`, `printer` and `kitchenware`. The fluents of the domain are defined in terms of their arguments:

$$loc(thing, place) \qquad (1)$$
$$in\_hand(robot, object)$$

The first predicate states the location of a thing; and the second predicate states that a robot has an object.These two predicates are *inertial fluents* subject to the law of inertia, which can be changed by an action. The *actions* in this domain include:

$$move(robot, place) \qquad (2)$$
$$grasp(robot, object)$$
$$putdown(robot, object)$$

The dynamics of the domain are defined using the following causal laws:

$$move(robot, Pl) \textbf{ causes } loc(robot, Pl) \qquad (3)$$
$$grasp(robot, Ob) \textbf{ causes } in\_hand(robot, Ob)$$
$$putdown(robot, Ob) \textbf{ causes } \neg in\_hand(robot, Ob)$$

state constraints:

$$loc(Ob, Pl) \textbf{ if } loc(robot, Pl),\ in\_hand(robot, Ob) \qquad (4)$$
$$\neg loc(Th, Pl_1) \textbf{ if } loc(Th, Pl_2),\ Pl_1 \neq Pl_2$$

and executability conditions:

$$\begin{array}{ll}
\textbf{impossible} & move(robot,Pl) \ \textbf{if} \ loc(robot,Pl) \\
\textbf{impossible} & A_1,\ A_2,\ \textbf{if}\ A_1 \neq A_2. \\
\textbf{impossible} & grasp(robot,Ob) \ \textbf{if} \ loc(robot,Pl1), \\
 & \quad loc(Ob,Pl2), Pl1 \neq Pl2 \\
\textbf{impossible} & grasp(robot,Ob) \ \textbf{if} \ in_hand(robot,Ob) \\
\textbf{impossible} & putdown(robot,Ob) \ \textbf{if} \ \neg in_hand(robot,Ob)
\end{array} \tag{5}$$

The top part of Figure 1 shows some state transitions in the HL; nodes include a subset of fluents (robot's position) and actions are the arcs between nodes. Although $\mathscr{D}_H$ does not include the costs of executing actions, these are included in the LL (see Section 3.2).

Histories with defaults A recorded history of a dynamic domain is usually defined as a collection of records of the form $obs(fluent, boolean, step)$ and $hpd(action, step)$. The former states that a specific fluent was observed to be true or false at a given step of the domain's trajectory, and the latter states that a specific action happened (or was executed by the robot) at that step. In this paper, *we expand on this view by allowing histories to contain (possibly prioritized) defaults describing the values of fluents in their initial states*. A default $d(X)$ stating that *in the typical initial state elements of class c satisfying property b also have property p* is represented as:

$$d(X) = \left\{ \begin{array}{l} default(d(X)) \\ head(d(X), p(X)) \\ body(d(X), c(X)) \\ body(d(X), b(X)) \end{array} \right. \tag{6}$$

where the literal in the "head" of the default, e.g., $p(X)$ is true if all the literals in the "body" of the default, e.g., $b(X)$ and $c(X)$, hold true; see (Gelfond and Kahl 2014) for formal semantics of defaults. In this paper, we abbreviate $obs(f, true, 0)$ and $obs(f, false, 0)$ as $init(f, true)$ and $init(f, false)$ respectively.

Example 1 *[Example of defaults]*
Consider the following statements about the locations of textbooks in the initial state in our illustrative example. *Textbooks are typically in the main library. If a textbook is not there, it is in the auxiliary library. If a textbook is checked out, it can be found in the office.* These defaults can be represented as:

$$\begin{array}{l} default(d_1(X)) \\ head(d_1(X), loc(X, main\_library)) \\ body(d_1(X), textbook(X)) \end{array} \tag{7}$$

$$\begin{array}{l} default(d_2(X)) \\ head(d_2(X), loc(X, aux\_library)) \\ body(d_2(X), textbook(X)) \\ body(d_2(X), \neg loc(X, main\_library)) \end{array} \tag{8}$$

$$\begin{array}{l} default(d_3(X)) \\ head(d_3(X), loc(X, office)) \\ body(d_3(X), textbook(X)) \\ body(d_3(X), \neg loc(X, main\_library)) \\ body(d_3(X), \neg loc(X, aux\_library)) \end{array} \tag{9}$$

A default such as "kitchenware are usually in the kitchen" may be represented in a similar manner. We first present multiple informal examples to illustrate reasoning with these defaults; Definition 3 (below) will formalize this reasoning. For textbook tb_1, history $\mathscr{H}_1$ containing the above statements should entail: $holds(loc(tb_1, main\_library), 0)$. A history $\mathscr{H}_2$ obtained from $\mathscr{H}_1$ by adding an observation: $init(loc(tb_1, main\_library), false)$ renders the first default inapplicable; hence $\mathscr{H}_2$ should entail: $holds(loc(tb_1, aux\_library), 0)$. A history $\mathscr{H}_3$ obtained from $\mathscr{H}_2$ by adding an observation: $init(loc(tb_1, aux\_library), false)$ entails: $holds(loc(tb_1, office), 0)$.

Consider history $\mathscr{H}_4$ obtained by adding observation: $obs(loc(tb_1, main\_library), false, 1)$ to $\mathscr{H}_1$. This observation should defeat the default d_1 in Equation 7 because if this default's conclusion were true in the initial state, it would also be true at step 1 (by inertia), which contradicts our observation. The book tb_1 is thus not in the main library initially. The second default will conclude that this book is initially in the auxiliary library—the inertia axiom will propagate this information and $\mathscr{H}_4$ will entail: $holds(loc(tb_1, aux\_library), 1)$.

The definition of entailment relation can now be given with respect to a fixed system description $\mathscr{D}_H$. We start with the notion of *a state of transition diagram τ_H of $\mathscr{D}_H$ compatible with a description $\mathscr{I}$ of the initial state of history $\mathscr{H}$*. We use the following terminology. We say that a set S of literals is *closed under a default d* if S contains the head of d whenever it contains all literals from the body of d and does not contain the literal contrary to d's head. S is *closed under a constraint* of $\mathscr{D}_H$ if S contains the constraint's head whenever it contains all literals from the constraint's body. Finally, we say that a set U of literals is the *closure of S* if $S \subseteq U$, U is closed under constraints of $\mathscr{D}_H$ and defaults of $\mathscr{H}$, and no proper subset of U satisfies these properties.

Definition 1 *[Compatible initial states]*
A state σ of τ_H is *compatible* with a description $\mathscr{I}$ of the initial state of history $\mathscr{H}$ if:

1. σ satisfies all observations of $\mathscr{I}$,
2. σ contains the closure of the union of statics of $\mathscr{D}_H$ and the set $\{f : init(f, true) \in \mathscr{I}\} \cup \{\neg f : init(f, false) \in \mathscr{I}\}$.

Let $\mathscr{I}_k$ be the description of the initial state of history $\mathscr{H}_k$. States in Example 1 compatible with $\mathscr{I}_1$, $\mathscr{I}_2$, $\mathscr{I}_3$ must then contain $\{loc(tb_1, main\_library)\}$, $\{loc(tb_1, aux\_library)\}$, and $\{loc(tb_1, office)\}$ respectively. There are multiple such states, which differ by the location of robot. Since $\mathscr{I}_1 = \mathscr{I}_4$ they have the same compatible states. Next, we define *models* of history $\mathscr{H}$, i.e., paths of the transition diagram τ_H of $\mathscr{D}_H$ compatible with $\mathscr{H}$.

Definition 2 *[Models]*
A path P of τ_H is a *model* of history $\mathscr{H}$ with description $\mathscr{I}$ of its initial state if there is a collection E of *init* statements such that:

1. If $init(f, true) \in E$ then $\neg f$ is the head of one of the defaults of $\mathscr{I}$. Similarly, for $init(f, false)$.

2. The initial state of P is compatible with the description: $\mathscr{I}_E = \mathscr{I} \cup E$.

3. Path P satisfies all observations in $\mathscr{H}$.

4. There is no collection E_0 of *init* statements which has less elements than E and satisfies the conditions above.

We will refer to E as an *explanation* of $\mathscr{H}$. Models of $\mathscr{H}_1$, $\mathscr{H}_2$, and $\mathscr{H}_3$ are paths consisting of initial states compatible with $\mathscr{I}_1$, $\mathscr{I}_2$, and $\mathscr{I}_3$—the corresponding explanations are empty. However, in the case of $\mathscr{H}_4$, the situation is different—the predicted location of tb_1 will be different from the observed one. The only explanation of this discrepancy is that tb_1 is an exception to the first default. Adding $E = \{init(loc(tb_1, main\_library), false)\}$ to $\mathscr{I}_4$ will resolve the problem.

Definition 3 *[Entailment and consistency]*

- Let $\mathscr{H}^n$ be a history of length n, f be a fluent, and $0 \leq i \leq n$ be a step of $\mathscr{H}^n$. We say that $\mathscr{H}^n$ *entails* a statement $Q = holds(f,i)$ ($\neg holds(f,i)$) if for every model P of $\mathscr{H}^n$, fluent literal f ($\neg f$) belongs to the ith state of P. We denote the entailment as $\mathscr{H}^n \models Q$.
- A history which has a model is said to be *consistent*.

It can be shown that histories from Example 1 are consistent and that our entailment captures the corresponding intuition.

Reasoning with HL domain representation The HL domain representation ($\mathscr{D}_H$ and $\mathscr{H}$) is translated into a program in CR-Prolog, which incorporates consistency restoring rules in ASP (Balduccini and Gelfond 2003; Gelfond and Kahl 2014); specifically, we use the knowledge representation language SPARC that expands CR-Prolog and provides explicit constructs to specify objects, relations, and their sorts (Balai, Gelfond, and Zhang 2013). ASP is a declarative language that can represent recursive definitions, defaults, causal relations, special forms of self-reference, and other language constructs that occur frequently in non-mathematical domains, and are difficult to express in classical logic formalisms (Baral 2003). ASP is based on the stable model semantics of logic programs, and builds on research in non-monotonic logics (Gelfond 2008). A CR-Prolog program is thus a collection of statements describing domain objects and relations between them. The ground literals in an *answer set* obtained by solving the program represent beliefs of an agent associated with the program[1]; program consequences are statements that are true in all such belief sets. Algorithms for computing the entailment relation of AL and related tasks such as planning and diagnostics are thus based on reducing these tasks to computing answer sets of programs in CR-Prolog. First, $\mathscr{D}_H$ and $\mathscr{H}$ are translated into an ASP program $\Pi(\mathscr{D}_H, \mathscr{H})$ consisting of direct translation of causal laws of $\mathscr{D}_H$, inertia axioms, closed world assumption for defined fluents, reality checks, records of observations, actions and defaults from $\mathscr{H}$, and special axioms for *init*:

$$holds(F,0) \leftarrow init(F,true) \qquad (10)$$
$$\neg holds(F,0) \leftarrow init(F,false)$$

[1] SPARC uses DLV (Leone et al. 2006) to generate answer sets.

In addition, every default of $\mathscr{I}$ is turned into an ASP rule:

$$holds(p(X),0) \leftarrow c(X),\ holds(b(X),0),\ not\ \neg holds(p(X),0) \qquad (11)$$

and a consistency-restoring rule:

$$\neg holds(p(X),0) \stackrel{+}{\leftarrow} c(X),\ holds(b(X),0) \qquad (12)$$

which states that to restore consistency of the program one may assume that the conclusion of the default is false. For more details about the translation, CR-rules and CR-Prolog, please see (Gelfond and Kahl 2014).

Proposition 1 *[Models and Answer Sets]*
A path $P = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_{n-1}, a_n \rangle$ of τ_H is a model of history $\mathscr{H}^n$ iff there is an answer set S of a program $\Pi(\mathscr{D}_H, \mathscr{H})$ such that:

1. *A fluent $f \in \sigma_i$ iff $holds(f,i) \in S$,*
2. *A fluent literal $\neg f \in \sigma_i$ iff $\neg holds(f,i) \in S$,*
3. *An action $e \in a_i$ iff $occurs(e,i) \in S$.*

The proposition reduces computation of models of $\mathscr{H}$ to computing answer sets of a CR-Prolog program. This proposition allows us to reduce the task of planning to computing answer sets of a program obtained from $\Pi(\mathscr{D}_H, \mathscr{H})$ by adding the definition of a goal, a constraint stating that the goal must be achieved, and a rule generating possible future actions of the robot.

3.2 LL domain representation

The LL system description $\mathscr{D}_L$ consists of a sorted signature and axioms that describe a transition diagram τ_L. The sorted signature Σ_L of action theory describing τ_L includes the sorts from signature Σ_H of HL with two additional sorts `room` and `cell`, which are subsorts of sort `place`. Their elements satisfy the static relation *part_of(cell, room)*. We also introduce the static *neighbor(cell, cell)* to describe neighborhood relation between cells. Fluents of Σ_L include those of Σ_H, an additional inertial fluent: *searched(cell, object)*—robot searched a cell for an object—and two defined fluents: *found(object, place)*—an object was found in a place—and *continue_search(room, object)*—the search for an object is continued in a room.

The actions of Σ_L include the HL actions that are viewed as being represented at a higher resolution, e.g., movement is possible to specific cells. The causal law describing the effect of *move* may be stated as:

$$move(robot,Y)\ \ \textbf{causes}\ \ \{loc(robot,Z) : neighbor(Z,Y)\} \qquad (13)$$

where Y, Z are cells. This causal law states that moving to a cell can cause the robot to be in one of the neighboring cells[2]. The LL includes an additional action *search* that enables robots to search for objects in cells; the corresponding

[2] This is a special case of a non-deterministic causal law defined in extensions of AL with non-boolean fluents, i.e., functions whose values can be elements of arbitrary finite domains.

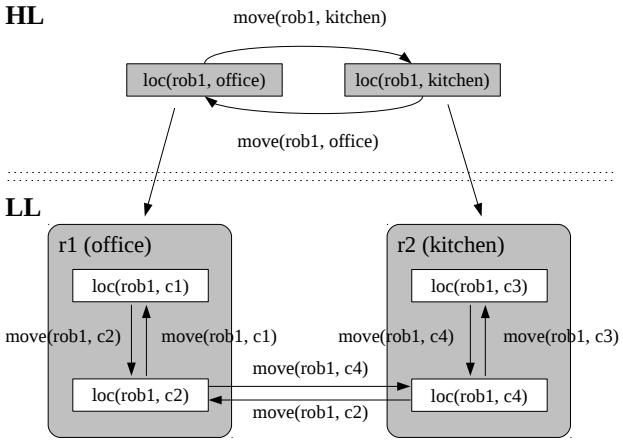


Figure 1: Illustrative example of state transitions in the HL and the LL of the architecture.

causal laws and constraints may be written as:

$$search(cell, object) \textbf{ causes } searched(cell, object) \quad (14)$$
$$found(object, cell) \textbf{ if } searched(cell, object), \; loc(object, cell)$$
$$found(object, room) \textbf{ if } part\_of(cell, room), \; found(object, cell)$$
$$continue\_search(room, object) \textbf{ if } \neg found(object, room), \; part\_of(cell, room), \neg searched(cell, object)$$

We also introduce a defined fluent *failure* that holds iff the object under consideration is not in the room that the robot is searching—this fluent is defined as:

$$failure(object, room) \textbf{ if } loc(robot, room), \quad (15)$$
$$\neg continue\_search(room, object), \neg found(object, room)$$

This completes the action theory that describes τ_L. The states of τ_L can be viewed as extensions of states of τ_H by physically possible fluents and statics defined in the language of LL. Moreover, for every HL state-action-state transition $\langle \sigma, a, \sigma' \rangle$ and every LL state s compatible with σ (i.e., $\sigma \subset s$), there is a path in the LL from s to some state compatible with σ'.

Unlike the HL system description in which effects of actions and results of observations are always accurate, the action effects and observations in the LL are only known with some degree of probability. The state transition function $T : S \times A \times S' \rightarrow [0,1]$ defines the probabilities of state transitions in the LL. Due to perceptual limitations of the robot, only a subset of the fluents are observable in the LL; we denote this set of fluents by Z. Observations are elements of Z associated with a probability, and are obtained by processing sensor inputs using probabilistic algorithms. The observation function $O : S \times Z \rightarrow [0,1]$ defines the probability of observing specific observable fluents in specific states. Functions T and O are computed using prior knowledge, or by observing the effects of specific actions in specific states (see Section 4.1).

States are partially observable in the LL, and we introduce (and reason with) *belief states*, probability distributions over the set of states. Functions T and O describe a probabilistic transition diagram defined over belief states. The initial belief state is represented by B_0, and is updated iteratively using Bayesian inference:

$$B_{t+1}(s_{t+1}) \propto O(s_{t+1}, o_{t+1}) \sum_s T(s, a_{t+1}, s_{t+1}) \cdot B_t(s) \quad (16)$$

The LL system description includes a reward specification $R : S \times A \times S' \rightarrow \Re$ that encodes the relative cost or *value* of taking specific actions in specific states. Planning in the LL then involves computing a *policy* that maximizes the reward over a planning horizon. This policy maps belief states to actions: $\pi : B_t \mapsto a_{t+1}$. We use a point-based approximate algorithm to compute this policy (Ong et al. 2010). In our illustrative example, an LL policy computed for HL action *move* is guaranteed to succeed, and that the LL policy computed for HL action *grasp* considers three LL actions: *move*, *search*, and *grasp*. Plan execution in the LL corresponds to using the computed policy to repeatedly choose an action in the current belief state, and updating the belief state after executing that action and receiving an observation. We henceforth refer to this algorithm as "POMDP-1".

Unlike the HL, history in the LL representation consists of observations and actions over one time step; the current belief state is assumed to be the result of all information obtained in previous time steps (first-order Markov assumption). In this paper, the LL domain representation is translated automatically into POMDP models, i.e., specific data structures for representing the components of $\mathscr{D}_L$ (described above) such that existing POMDP solvers can be used to obtain action policies.

We observe that the coupling between the LL and the HL has some key consequences. First, for any HL action, the relevant LL variables are identified automatically, improving the computational efficiency of computing the LL policies. Second, if LL actions cause different fluents, these fluents are independent. Finally, although defined fluents are crucial in determining what needs to be communicated between the levels of the architecture, they themselves need not be communicated.

3.3 Control loop

Algorithm 1 describes the architecture's control loop[3]. First, the LL observations obtained in the current location add statements to the HL history, and the HL initial state (s^H_{init}) is communicated to the LL (line 1). The assigned task determines the HL goal state (s^H_{goal}) for planning (line 2). Planning in the HL provides a sequence of actions with deterministic effects (line 3).

In some situations, planning in the HL may provide multiple plans, e.g., when the object that is to be grasped can be in one of multiple locations, tentative plans may be generated for the different hypotheses regarding the object's location. In such situations, all the HL plans are communicated to the

[3]We leave the proof of the correctness of this algorithm as future work.

Algorithm 1: Control loop of architecture

Input: The HL and LL domain representations, and the specific task for robot to perform.

1 LL observations reported to HL history; HL initial state (s^H_{init}) communicated to LL.
2 Assign goal state s^H_{goal} based on task.
3 Generate HL plan(s).
4 **if** *multiple HL plans exist* **then**
5 Send plans to the LL, select plan with lowest (expected) action cost and communicate to the HL.
6 **end**
7 **if** *HL plan exists* **then**
8 **for** $a^H_i \in$ *HL plan:* $i \in [1,n]$ **do**
9 Pass a^H_i and relevant fluents to LL.
10 Determine initial belief state over the relevant LL state space.
11 Generate LL action policy.
12 **while** a^H_i *not completed* ***and*** a^H_i *achievable* **do**
13 Execute an action based on LL action policy.
14 Make an LL observation and update belief state.
15 **end**
16 LL observations and action outcomes add statements to HL history.
17 **if** *results unexpected* **then**
18 Perform diagnostics in HL.
19 **end**
20 **if** *HL plan invalid* **then**
21 Replan in the HL (line 3).
22 **end**
23 **end**
24 **end**

LL and compared based on their costs, e.g., the expected time to execute the plans. The plan with the least expected cost is communicated to the HL (lines 4-6).

If an HL plan exists, actions are communicated one at a time to the LL along with the relevant fluents (line 9). For HL action a^H_i, the communicated fluents are used to automatically identify the relevant LL variables and set the initial belief state, e.g., a uniform distribution (line 10). An LL action policy is computed (line 11) and used to execute actions and update the belief state until a^H_i is achieved or inferred to be unachievable (lines 12-15). The outcome of executing the LL policy, and the LL observations, add to the HL history (line 16). For instance, if defined fluent *failure* is true for object ob_1 and room rm_1, the robot reports: $obs(loc(ob_1, rm_1), false)$ to the HL history. If the results are unexpected, diagnosis is performed in the HL (lines 17-19); we assume that the robot is capable of identifying these unexpected outcomes. If the HL plan is invalid, a new plan is generated (lines 20-22); else, the next action in the HL plan is executed.

4 Experimental setup and results

This section describes the experimental setup and results of evaluating the proposed architecture in indoor domains.

4.1 Experimental setup

The architecture was evaluated in simulation and on physical robots. To provide realistic observations in the simulator, we included object models that characterize objects using probabilistic functions of features extracted from images captured by a camera on physical robots (Li and Sridharan 2013). The simulator also uses action models that reflect the motion of the robot. Specific instances of objects of different classes were simulated in a set of rooms. The experimental setup also included an initial training phase in which the robot repeatedly executed the different movement actions and applied the visual input processing algorithms on images with known objects. A human participant provided some of the ground truth data, e.g., labels of objects in images. A comparison of the expected and actual outcomes was used to define the functions that describe the probabilistic transition diagram (T, O) in the LL, while the reward specification is defined by also considering the computational time required by different visual processing and navigation algorithms.

In each trial of the experimental results summarized below, the robot's goal is to move specific objects to specific places; the robot's location, target object, and locations of objects are chosen randomly in each trial. A sequence of actions extracted from an answer set obtained by solving the SPARC program of the HL domain representation provides an HL plan. If a robot (`robot1`) that is in the *office* is asked to fetch a textbook (`tb1`) from the *main_library*, the HL plan consists of the following sequence of actions:

$$move(robot1, main\_library)$$
$$grasp(robot1, tb1)$$
$$move(robot1, office)$$
$$putdown(robot1, tb1)$$

The LL action policies for each HL action are generated by solving the appropriate POMDP models using the APPL solver (Ong et al. 2010; Somani et al. 2013). In the LL, the location of an object is considered to be known with certainty if the belief (of the object's occurrence) in a grid cell exceeds a threshold (0.85).

We experimentally compared our architecture, with the control loop described in Algorithm 1, henceforth referred to as "PA", with two alternatives: (1) POMDP-1 (see Section 3.2); and (2) POMDP-2, which revises POMDP-1 by assigning high probability values to defaults to bias the initial belief states. These comparisons evaluate two hypotheses: (H1) PA enables a robot to achieve the assigned goals more reliably and efficiently than using POMDP-1; (H2) our representation of defaults improves reliability and efficiency in comparison with not using default knowledge or assigning high probability values to defaults.

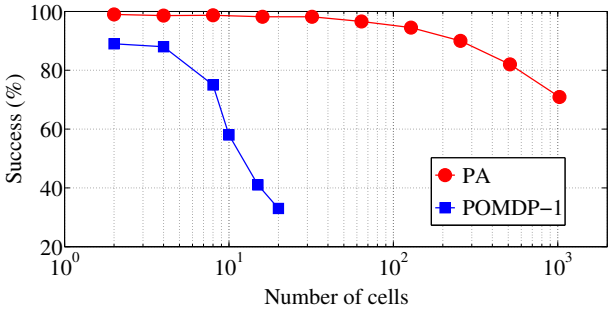


Figure 2: Ability to successfully achieve the assigned goal, as a function of the number of cells in the domain; with a limit on the time to compute policies PA significantly increases accuracy in comparison with just POMDP-1 as the number of cells in the domain increases.

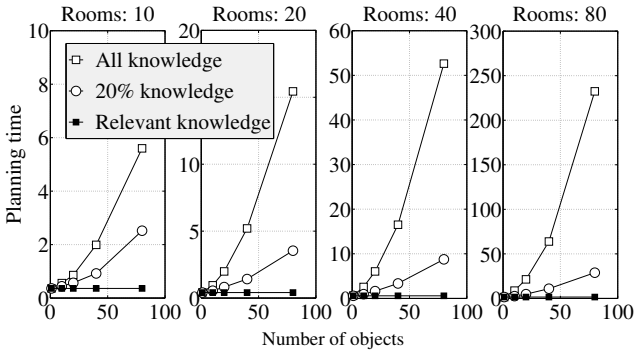


Figure 3: Planning time as a function of the number of rooms and the number of objects in the domain—*PA* scales to larger number of rooms and objects.

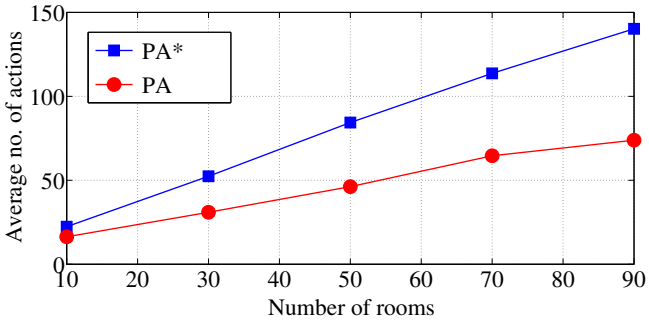


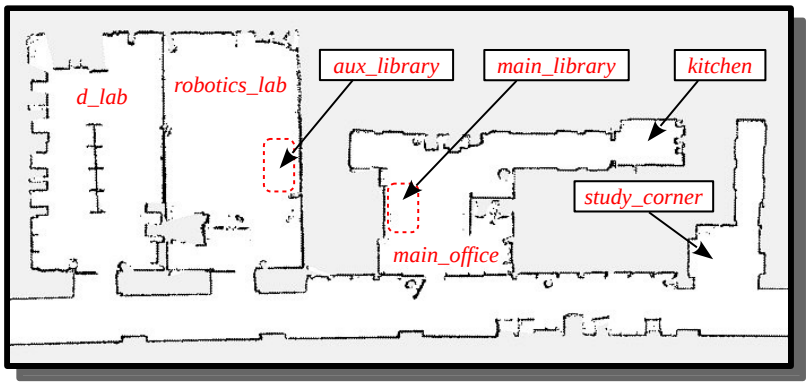
Figure 4: Effect of using default knowledge—principled representation of defaults significantly reduces the number of actions (and thus time) for achieving assigned goal.

4.2 Experimental Results

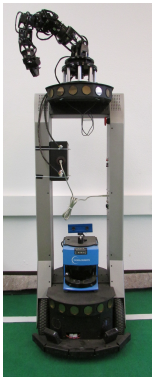
To evaluate H1, we first compared PA with POMDP-1 in a set of trials in which the robot's initial position is known but the position of the object to be moved is unknown. The solver used in POMDP-1 is given a fixed amount of time to compute action policies. Figure 2 summarizes the ability to successfully achieve the assigned goal, as a function of the number of cells in the domain. Each point in Figure 2 is the average of 1000 trials, and we set (for ease of interpretation) each room to have four cells. PA significantly improves the robot's ability to achieve the assigned goal in comparison with POMDP-1. As the number of cells (i.e., size of the domain) increases, it becomes computationally difficult to generate good POMDP action policies which, in conjunction with incorrect observations (e.g., false positive sightings of objects) significantly impacts the ability to successfully complete the trials. PA, on the other hand, focuses the robot's attention on relevant regions of the domain (e.g., specific rooms and cells). As the size of the domain increases, a large number of plans of similar cost may still be generated which, in conjunction with incorrect observations, may affect the robot's ability to successfully complete the trials—the impact is, however, much less pronounced.

Next, we computed the time taken by PA to generate a plan as the size of the domain increases. Domain size is characterized based on the number of rooms and the number of objects in the domain. We conducted three sets of experiments in which the robot reasons with: (1) all available knowledge of domain objects and rooms; (2) only knowledge relevant to the assigned goal—e.g., if the robot knows an object's default location, it need not reason about other objects and rooms in the domain to locate this object; and (3) relevant knowledge and knowledge of an additional 20% of randomly selected domain objects and rooms. Figure 3 summarizes these results. We observe that PA supports the generation of appropriate plans for domains with a large number of rooms and objects. We also observe that using only the knowledge relevant to the goal significantly reduces the planning time—such knowledge can be automatically selected using the relationships included in the HL system description. Furthermore, if we only use a probabilistic approach (POMDP-1), it soon becomes computationally intractable to generate a plan for domains with many objects and rooms; these results are not shown in Figure 3—see (Sridharan, Wyatt, and Dearden 2010; Zhang, Sridharan, and Washington 2013).

To evaluate H2, we first conducted multiple trials in which PA was compared with PA^*, a version that does not include any default knowledge. Figure 4 summarizes the average number of actions executed per trial as a function of the number of rooms in the domain—each sample point is the average of 10000 trials. The goal in each trial is (as before) to move a specific object to a specific place. We observe that the principled use of default knowledge significantly reduces the number of actions (and thus time) required to achieve the assigned goal. Next PA was compared with POMDP-2, which assigns high probability values to default information and suitably revises the initial belief state. We observe that the effect of assigning a probability value to defaults is arbitrary depending on multiple factors: (a) the numerical value chosen; and (b) whether the ground truth matches the default



(a) Domain map



(b) Robot platform

Figure 5: Subset of the map of the second floor of our department; specific places are labeled as shown, and used during planning to achieve the assigned goals. The robot platform used in the experimental trials is also shown.

information. For instance, *if a large probability is assigned to the default knowledge that books are typically in the library, but the book the robot has to move is an exception to the default (e.g., a cookbook), it takes a significantly large amount of time for POMDP-2 to revise (and recover from) the initial belief.* PA, on the other hand, enables the robot to revise initial defaults and encode exceptions to defaults.

Robot Experiments: In addition to the trials in simulated domains, we compared PA with POMDP-1 on a wheeled robot over 50 trials conducted on two floors of our department building. This domain includes places in addition to those included in our illustrative example, e.g., Figure 5(a) shows a subset of the domain map of the third floor of our department, and Figure 5(b) shows the wheeled robot platform. Such domain maps are learned by the robot using laser range finder data, and revised incrementally over time. Manipulation by physical robots is not a focus of this work. Therefore, once the robot is next to the desired object, it currently asks for the object to be placed in the extended gripper; future work will include existing probabilistic algorithms for manipulation in the LL.

For experimental trials on the third floor, we considered 15 rooms, which includes faculty offices, research labs, common areas and a corridor. To make it feasible to use POMDP-1 in such large domains, we used our prior work on a hierarchical decomposition of POMDPs for visual sensing and information processing that supports automatic belief propagation across the levels of the hierarchy and model generation in each level of the hierarchy (Sridharan, Wyatt, and Dearden 2010; Zhang, Sridharan, and Washington 2013). The experiments included paired trials, e.g., over 15 trials (each), POMDP-1 takes 1.64 as much time as PA (on average) to move specific objects to specific places. For these paired trials, this 39% reduction in execution time provided by PA is statistically significant: $p\text{-}value = 0.0023$ at the 95% significance level.

Consider a trial in which the robot's objective is to bring a specific textbook to the place named *study_corner*. The robot uses default knowledge to create an HL plan that causes the robot to move to and search for the textbook in the *main_library*. When the robot does not find this textbook in the *main_library* after searching using a suitable LL policy, replanning in the HL causes the robot to investigate the *aux_library*. The robot finds the desired textbook in the *aux_library* and moves it to the target location. A video of such an experimental trial can be viewed online:
`http://youtu.be/8zL4R8te6wg`

5 Conclusions

This paper described a knowledge representation and reasoning architecture for robots that integrates the complementary strengths of declarative programming and probabilistic graphical models. The system descriptions of the tightly coupled high-level (HL) and low-level (LL) domain representations are provided using an action language, and the HL definition of recorded history is expanded to allow prioritized defaults. Tentative plans created in the HL using defaults and commonsense reasoning are implemented in the LL using probabilistic algorithms, generating observations that add suitable statements to the HL history. In the context of robots moving objects to specific places in indoor domains, experimental results indicate that the architecture supports knowledge representation, non-monotonic logical inference and probabilistic planning with qualitative and quantitative descriptions of knowledge and uncertainty, and scales well as the domain becomes more complex. Future work will further explore the relationship between the HL and LL transition diagrams, and investigate a tighter coupling of declarative logic programming and probabilistic reasoning for robots.

Acknowledgments

The authors thank Evgenii Balai for making modifications to SPARC to support some of the experiments reported in this paper. This research was supported in part by the U.S. Office of Naval Research (ONR) Science of Autonomy Award

N00014-13-1-0766. Opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the ONR.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

Balduccini, M., and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *Logical Formalization of Commonsense Reasoning, AAAI Spring Symposium Series*, 9–18.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

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

Chen, X.; Xie, J.; Ji, J.; and Sui, Z. 2012. Toward Open Knowledge Enabling for Human-Robot Interaction. *Journal of Human-Robot Interaction* 1(2):100–117.

Erdem, E.; Aker, E.; and Patoglu, V. 2012. Answer Set Programming for Collaborative Housekeeping Robotics: Representation, Reasoning, and Execution. *Intelligent Service Robotics* 5(4).

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gelfond, M. 2008. Answer Sets. In Frank van Harmelen and Vladimir Lifschitz and Bruce Porter., ed., *Handbook of Knowledge Representation*. Elsevier Science. 285–316.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco, USA: Morgan Kaufmann.

Halpern, J. 2003. *Reasoning about Uncertainty*. MIT Press.

Hanheide, M.; Gretton, C.; Dearden, R.; Hawes, N.; Wyatt, J.; Pronobis, A.; Aydemir, A.; Gobelbecker, M.; and Zender, H. 2011. Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *International Joint Conference on Artificial Intelligence*.

Hoey, J.; Poupart, P.; Bertoldi, A.; Craig, T.; Boutilier, C.; and Mihailidis, A. 2010. Automated Handwashing Assistance for Persons with Dementia using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding* 114(5):503–519.

Kaelbling, L., and Lozano-Perez, T. 2013. Integrated Task and Motion Planning in Belief Space. *International Journal of Robotics Research* 32(9-10).

Laird, J. E.; Newell, A.; and Rosenbloom, P. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33(3).

Langley, P., and Choi, D. 2006. An Unified Cognitive Architecture for Physical Agents. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Li, X., and Sridharan, M. 2013. Move and the Robot will Learn: Vision-based Autonomous Learning of Object Models. In *International Conference on Advanced Robotics*.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2006. BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Ong, S. C.; Png, S. W.; Hsu, D.; and Lee, W. S. 2010. Planning under Uncertainty for Robotic Tasks with Mixed Observability. *International Journal of Robotics Research* 29(8):1053–1068.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine learning* 62(1).

Rosenthal, S., and Veloso, M. 2012. Mobile Robot Planning to Seek Help with Spatially Situated Tasks. In *National Conference on Artificial Intelligence*.

Sanner, S., and Kersting, K. 2010. Symbolic Dynamic Programming for First-order POMDPs. In *National Conference on Artificial Intelligence (AAAI)*.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP Planning with Regularization. In *Advances in Neural Information Processing Systems (NIPS)*.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2010. Planning to See: A Hierarchical Aprroach to Planning Visual Actions on a Robot using POMDPs. *Artificial Intelligence* 174:704–725.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for Human-Robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology* 1(2):14:1–14:24.

Zhang, S.; Sridharan, M.; and Bao, F. S. 2012. ASP+POMDP: Integrating Non-monotonic Logical Reasoning and Probabilistic Planning on Robots. In *International Joint Conference on Development and Learning and on Epigenetic Robotics*.

Zhang, S.; Sridharan, M.; and Washington, C. 2013. Active Visual Planning for Mobile Robot Teams using Hierarchical POMDPs. *IEEE Transactions on Robotics* 29(4).

An ASP-Based Architecture for Autonomous UAVs in Dynamic Environments: Progress Report

Marcello Balduccini and **William C. Regli** and **Duc N. Nguyen**
Applied Informatics Group
Drexel University
Philadelphia, PA, USA

Abstract

Traditional AI reasoning techniques have been used successfully in many domains, including logistics, scheduling and game playing. This paper is part of a project aimed at investigating how such techniques can be extended to coordinate teams of unmanned aerial vehicles (UAVs) in dynamic environments. Specifically challenging are real-world environments where UAVs and other network-enabled devices must communicate to coordinate—and communication actions are neither reliable nor free. Such network-centric environments are common in military, public safety and commercial applications, yet most research (even multi-agent planning) usually takes communications among distributed agents as a given. We address this challenge by developing an agent architecture and reasoning algorithms based on Answer Set Programming (ASP). ASP has been chosen for this task because it enables high flexibility of representation, both of knowledge and of reasoning tasks. Although ASP has been used successfully in a number of applications, and ASP-based architectures have been studied for about a decade, to the best of our knowledge this is the first practical application of a complete ASP-based agent architecture. It is also the first practical application of ASP involving a combination of centralized reasoning, decentralized reasoning, execution monitoring, and reasoning about network communications. This work has been empirically validated using a distributed network-centric software evaluation testbed and the results provide guidance to designers in how to understand and control intelligent systems that operate in these environments.

Introduction

Unmanned Aerial Vehicles (UAVs) promise to revolutionize the way in which we use our airspace. From talk of automating the navigation for major shipping companies to the use of small helicopters as "deliverymen" that drop your packages at the door, it is clear that our airspaces will become increasingly crowded in the near future. This increased utilization and congestion has created the need for new and different methods of coordinating assets using the airspace. Currently, airspace management is the job for mostly human controllers. As the number of entities using the airspace vastly increases—many of which are autonomous—the need for improved autonomy techniques becomes evident.

The challenge in an environment full of UAVs is that the world is highly dynamic and the communications environment is uncertain, making coordination difficult. Communicative actions in such setting are neither reliable nor free.

The work discussed here is in the context of the development of a novel application of network-aware reasoning and of an intelligent mission-aware network layer to the problem of UAV coordination. Typically, AI reasoning techniques do not consider realistic network models, nor does the network layer reason dynamically about the needs of the mission plan. With network-aware reasoning (Figure 1a), a reasoner (either centralized or decentralized) factors in the communications network and its conditions, while with mission-aware networking, an intelligent network middleware service considers the mission and network state, and dynamically infers quality of service (QoS) requirements for mission execution.

In this paper we provide a general overview of the approach, and then focus on the aspect of network-aware reasoning. We address this challenge by developing an agent architecture and reasoning algorithms based on Answer Set Programming (ASP, (Gelfond and Lifschitz 1991; Marek and Truszczynski 1999; Baral 2003)). ASP has been chosen for this task because it enables high flexibility of representation, both of knowledge and of reasoning tasks. Although ASP has been used successfully in a number of applications, and ASP-based architectures have been studied for about a decade, to the best of our knowledge this is the first practical application of a complete ASP-based agent architecture. It is also the first practical application of ASP involving a combination of centralized reasoning, decentralized reasoning, execution monitoring, and reasoning about network communications. This work has been empirically validated using a distributed network-centric software evaluation testbed and the results provide guidance to designers in how to understand and control intelligent systems that operate in these environments.

The next section describes relevant systems and reasoning techniques, and is followed by a motivating scenario that applies to UAV coordination. The Technical Approach section describes network-aware reasoning and demonstrates the level of sophistication of the behavior exhibited by the UAVs using example problem instances. Next is a description of the network-centric evaluation testbed used for sim-

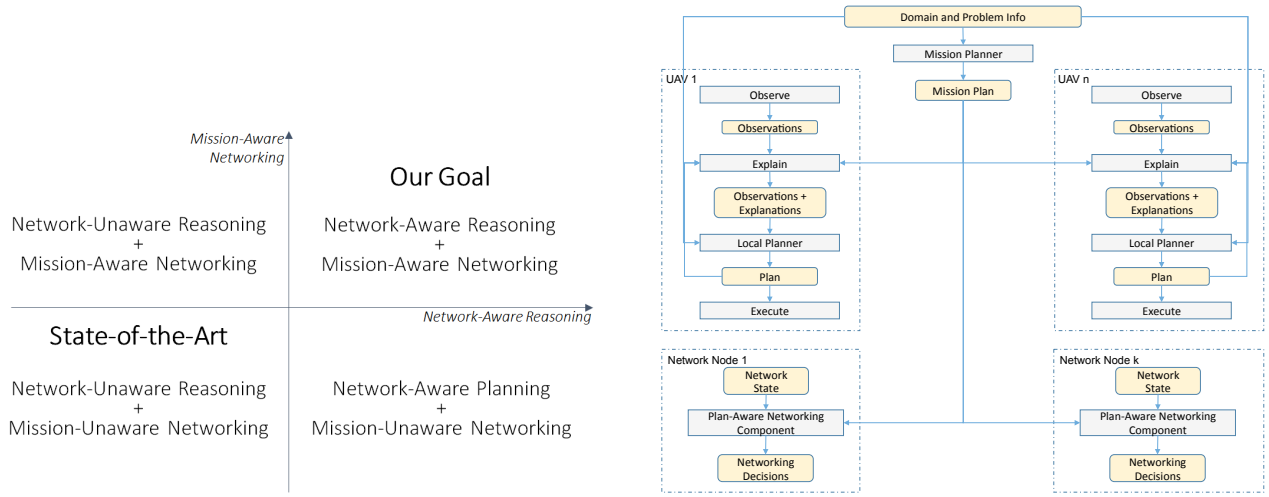


Figure 1: (a) The current state of reasoning and networking (lower-left) vs our goal combination (top-right); (b) Information flow in our framework.

ulations. Finally, we draw conclusions and discuss future work.

Related Work

Incorporating network properties into planning and decision-making has been investigated in (Usbeck, Cleveland, and Regli 2012). The authors' results indicate that plan execution effectiveness and performance is increased with the increased network-awareness during the planning phase. The UAV coordination approach in this current work combines network-awareness during the reasoning processes with a plan-aware network layer.

The problem of mission planning for UAVs under communication constraints has been addressed in (Kopeikin et al. 2013), where an ad-hoc task allocation process is employed to engage under-utilized UAVs as communication relays. In our work, we do not separate planning from the engagement of under-utilized UAVs, and do not rely on ad-hoc, hard-wired behaviors. Our approach gives the planner more flexibility and finer-grained control of the actions that occur in the plans, and allows for the emergence of sophisticated behaviors without the need to pre-specify them.

The architecture adopted in this work is an evolution of (Balduccini and Gelfond 2008), which can be viewed as an instantiation of the BDI agent model (Rao and Georgeff 1991; Wooldridge 2000). Here, the architecture has been extended to include a centralized mission planning phase, and to reason about other agents' behavior. Recent related work on logical theories of intentions (Blount, Gelfond, and Balduccini 2014) can be further integrated into our approach to allow for a more systematic hierarchical characterization of actions, which is likely to increase performance.

Traditionally, AI planning techniques have been used (to great success) to perform multi-agent teaming, and UAV coordination. Multi-agent teamwork decision frameworks such as the ones described in (Pynadath and Tambe 2002) may factor communication costs into the decision-making. However, the agents do not actively reason about other agent's observed behavior, nor about the communication process. Moreover, policies are used as opposed to reasoning from models of domains and of agent behavior.

The reasoning techniques used in the present work have already been successfully applied to domains ranging from complex cyber-physical systems to workforce scheduling. To the best of our knowledge, however, they have never been applied to domains combining realistic communications and multiple agents.

Finally, high-fidelity multi-agent simulators (e.g., AgentFly (David Sislak and Pechoucek 2012)) do not account for network dynamism nor provide a realistic network model. For this reason, we base our simulator on the Common Open Research Emulator (CORE) (Ahrenholz 2010). CORE provides network models in which communications are neither reliable nor free.

Motivating Scenario

To motivate the need for network-aware reasoning and mission-aware networking, consider a simple UAV coordination problem, depicted in Figure 4a, in which two UAVs are tasked with taking pictures of a set of three targets, and with relaying the information to a home base.

Fixed relay access points extend the communications range of the home base. The UAVs can share images of the targets with each other and with the relays when they are within radio range. The simplest solution to this problem consists in entirely disregarding the networking component of the scenario, and generating a mission plan in which each UAV flies to a different set of targets, takes pictures of them, and flies back to the home base, where the pictures are transferred. This solution, however, is not satisfactory. First of all, it is inefficient, because it requires that the UAVs fly all the way back to the home base before the images can be used. The time it takes for the UAVs to fly back may easily render the images too outdated to be useful. Secondly, disregarding

the network during the reasoning process may lead to mission failure — especially in the case of unexpected events, such as enemy forces blocking transit to and from the home base after a UAV has reached a target. Even if the UAVs are capable of autonomous behavior, they will not be able to complete the mission unless they take advantage of the network.

Another common solution consists of acknowledging the availability of the network, and assuming that the network is constantly available throughout plan execution. A corresponding mission plan would instruct each UAV to fly to a different set of targets, and take pictures of them, while the network relays the data back to the home base. This solution is *optimistic* in that it assumes that the radio range is sufficient to reach the area where the targets are located, and that the relays will work correctly throughout the execution of the mission plan.

This optimistic solution is more efficient than the previous one, since the pictures are received by the home base soon after they are taken. Under realistic conditions, however, the strong assumptions it relies upon may easily lead to mission failure—for example, if the radio range does not reach the area where the targets are located.

In this work, the reasoning processes take into account not only the presence of the network, but also its configuration and characteristics, taking advantage of available resources whenever possible. The mission planner is given information about the radio range of the relays and determines, for example, that the targets are out of range. A possible mission plan constructed by this information into account consists in having one UAV fly to the targets and take pictures, while the other UAV remains in a position to act as a network bridge between the relays and the UAV that is taking pictures. This solution is as efficient as the optimistic solution presented earlier, but is more robust, because it does not rely on the same strong assumptions.

Conversely, when given a mission plan, an intelligent network middleware service capable of sensing conditions and modifying network parameters (e.g., modify network routes, limit bandwidth to certain applications, and prioritize network traffic) is able to adapt the network to provide optimal communications needed during plan execution. A relay or UAV running such a middleware is able to interrupt or limit bandwidth given to other applications to allow the other UAV to transfer images and information toward home base. Without this traffic prioritization, network capacity could be reached prohibiting image transfer.

Technical Approach

In this section, we formulate the problem in more details; provide technical background; discuss the design of the agent architecture and of the reasoning modules; and demonstrate the sophistication of the resulting behavior of the agents in two scenarios.

Problem Formulation

A problem instance for coordinating UAVs to observe targets and deliver information (e.g., images) to a home base is defined by a set of UAVs, $u_1, u_2, \ldots$, a set of targets, $t_1, t_2, \ldots$, a (possibly empty) set of fixed radio relays, $r_1, r_2, \ldots$, and a home base. The UAVs, the relays, and the home base are called radio nodes (or network nodes). Two nodes are in radio contact if they are within a distance ρ from each other, called radio range[1], or if they can relay information to each other through intermediary radio nodes that are themselves within radio range. The UAVs are expected to travel from the home base to the targets to take pictures of the targets and deliver them to the home base. A UAV will automatically take a picture when it reaches a target. If a UAV is within radio range of a radio node, the pictures are automatically shared. From the UAVs' perspective, the environment is only partially observable. Features of the domain that are observable to a UAV u are (1) which radio nodes u can and cannot communicate with by means of the network, and (2) the position of any UAV that near u.

The goal is to have the UAVs take a picture of each of the targets so that (1) the task is accomplished as quickly as possible, and (2) the total "staleness" of the pictures is as small as possible. Staleness is defined as the time elapsed from the moment a picture is taken, to the moment it is received by the home base. While the UAVs carry on their tasks, the relays are expected to actively prioritize traffic over the network in order to ensure mission success and further reduce staleness.

Answer Set Programming

In this section we provide a definition of the syntax of ASP and of its informal semantics. We refer the reader to (Gelfond and Lifschitz 1991; Niemelä and Simons 2000; Baral 2003) for a specification of the formal semantics. Let Σ be a signature containing constant, function and predicate symbols. Terms and atoms are formed as usual in first-order logic. A (basic) literal is either an atom a or its strong (also called classical or epistemic) negation $\neg a$. A *rule* is a statement of the form:

$$h_1 \text{ OR } \ldots \text{ OR } h_k \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n$$

where h_i's and l_i's are ground literals and *not* is the so-called *default negation*. The intuitive meaning of the rule is that a reasoner who believes $\{l_1, \ldots, l_m\}$ and has no reason to believe $\{l_{m+1}, \ldots, l_n\}$, must believe one of h_i's. Symbol $\leftarrow$ can be omitted if no l_i's are specified. Often, rules of the form $h \leftarrow \text{not } h, l_1, \ldots, \text{not } l_n$ are abbreviated into $\leftarrow l_1, \ldots, \text{not } l_n$, and called *constraints*. The intuitive meaning of a constraint is that $\{l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n\}$ must not be satisfied. A rule containing variables is interpreted as the shorthand for the set of rules obtained by replacing the variables with all the possible ground terms. A *program* is a pair $\langle \Sigma, \Pi \rangle$, where Σ is a signature and Π is a set of rules over Σ. We often denote programs just by the second element of the pair, and let the signature be defined implicitly. Finally, the *answer set* (or *model*) of a program Π is the collection of its consequences under the answer set

[1]For simplicity, we assume that all the radio nodes use comparable network devices, and that thus ρ is unique throughout the environment.

semantics. Notice that the semantics of ASP is defined in such a way that programs may have multiple answer sets, intuitively corresponding to alternative solutions satisfying the specification given by the program. The semantics of default negation provides a simple way of encoding choices. For example, the set of rules $\{p \leftarrow \text{not } q.\ \ q \leftarrow \text{not } p.\}$ intuitively states that either p or q may hold, and the corresponding program has two answer sets, $\{p\}$, $\{q\}$. The language of ASP has been extended with *constraint literals* (Niemelä and Simons 2000), which are expressions of the form $m\{l_1, l_2, \ldots, l_k\}n$, where m, n are arithmetic expressions and l_i's are basic literals as defined above. A constraint literal is satisfied whenever the number of literals that hold from $\{l_1, \ldots, l_k\}$ is between m and n, inclusive. Using constraint literals, the choice between p and q, under some set of conditions Γ, can be compactly encoded by the rule $1\{p, q\}1 \leftarrow \Gamma$. A rule of this kind is called *choice rule*. To further increase flexibility, the set $\{l_1, \ldots, l_k\}$ can also be specified as $\{l(\vec{X}) : d(\vec{X})\}$, where $\vec{X}$ is a list of variables. Such an expression intuitively stands for the set of all $l(\vec{x})$ such that $d(\vec{x})$ holds. We refer the reader to (Niemelä and Simons 2000) for a more detailed definition of the syntax of constraint literals and of the corresponding extended rules.

Agent Architecture

The architecture used in this project follows the BDI agent model (Rao and Georgeff 1991; Wooldridge 2000), which provides a good foundation because of its logical underpinning, clear structure and flexibility. In particular, we build upon ASP-based instances of this model (Baral and Gelfond 2000; Balduccini and Gelfond 2008) because they employ directly-executable logical languages featuring good computational properties while at the same time ensuring elaboration tolerance (McCarthy 1998) and elegant handling of incomplete information, non-monotonicity, and dynamic domains.

A sketch of the information flow throughout the system is shown in Figure 1b.[2] Initially, a centralized *mission planner* is given a description of the domain and of the problem instance, and finds a plan that uses the available UAVs to achieve the goal.

Next, each UAV receives the plan and begins executing it individually. As plan execution unfolds, the communication state changes, potentially affecting network connectivity. For example, the UAVs may move in and out of range of each other and of the other network nodes. Unexpected events, such as relays failing or temporarily becoming disconnected, may also affect network connectivity. When that happens, each UAV reasons in a decentralized, autonomous fashion to overcome the issues. As mentioned earlier, the key to taking into account, and hopefully compensating for, any unexpected circumstances is to actively employ, in the reasoning processes, realistic and up-to-date information about the communications state.

The control loop used by each UAV is shown in Figure 2a. In line with (Gelfond and Lifschitz 1991; Marek and Truszczynski 1999; Baral 2003), the loop and the I/O functions are implemented procedurally, while the reasoning functions ($Goal\_Achieved$, $Unexpected\_Observations$, $Explain\_Observations$, $Compute\_Plan$) are implemented in ASP. The loop takes in input the mission goal and the mission plan, which potentially includes courses of actions for multiple UAVs. Functions New_Observations, Next_Action, Tail, Execute, Record_Execution perform basic manipulations of data structures, and interface the agent with the execution and perception layers. Functions $Next\_Action$ and Tail are assumed to be capable of identifying the portions of the mission plan that are relevant to the UAV executing the loop. The remaining functions occurring in the control loop implement the reasoning tasks. Central to the architecture is the maintenance of a history of past observations and actions executed by the agent. Such history is stored in variable H and updated by the agent when it gathers observations about its environment and when it performs actions. It is important to note that variable $H$is local to the specific agent executing the loop, rather than shared among the UAVs (which would be highly unrealistic in a communication-constrained environment). Thus, different agents will develop differing views of the history of the environment as execution unfolds. At a minimum, the difference will be due to the fact that agents cannot observe each other's actions directly, but only their consequences, and even those are affected by the partial observability of the environment.

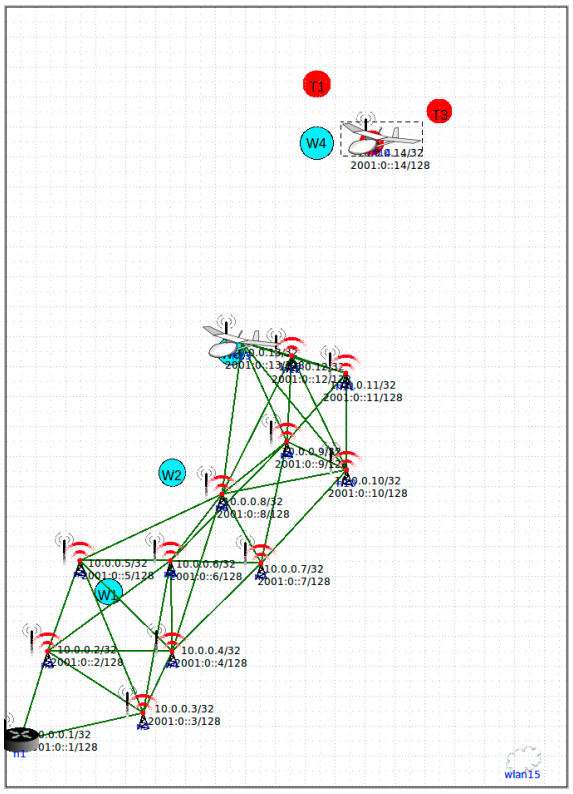
Details on the control loop can be found in (Balduccini and Gelfond 2008). With respect to that version of the loop, the control loop used in the present work does not allow for the selection of a new goal at run-time, but it extends the earlier control loop with the ability to deal with, and reason about, an externally-provided, multi-agent plan, and to reason about other agents' behavior. We do not expect run-time selection of goals to be difficult to embed in the control loop presented here, but doing so is out of the scope of the current phase of the project.

Network-Aware Reasoning

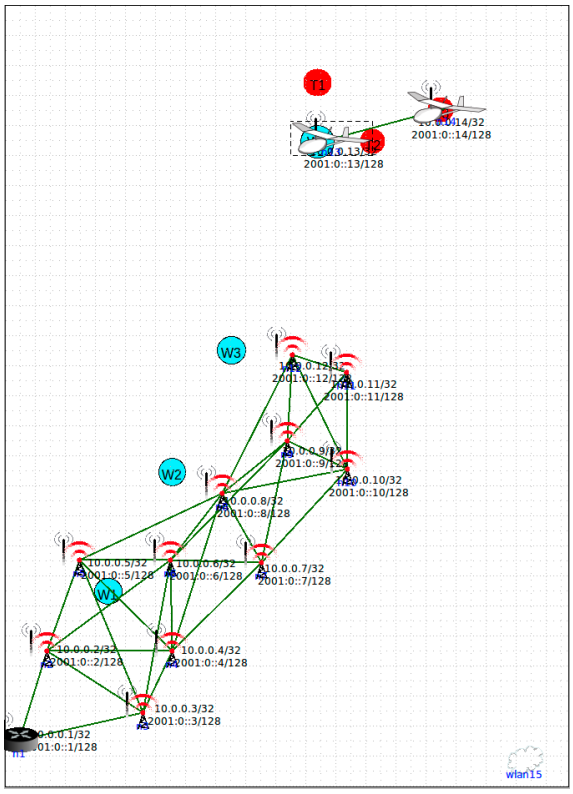
The major reasoning tasks (centralized mission planning, as well as anomaly detection, explanation and planning within each agent) are reduced to finding models of answer-set based formalizations of the corresponding problems. Central to all the reasoning tasks is the ability to represent the evolution of the environment over time. Such evolution is conceptualized into a *transition diagram* (Gelfond and Lifschitz 1993), a graph whose nodes correspond to states of the environment, and whose arcs describe state transitions due to the execution of actions. Let $\mathcal{F}$ be a collection of *fluents*, expressions representing relevant properties of the domain that may change over time, and let $\mathcal{A}$ be a collection of *actions*. A *fluent literal* l is a fluent $f \in \mathcal{F}$ or its negation $\neg f$. A *state* σ is a complete and consistent set of fluent literals.

The transition diagram is formalized in ASP by rules describing the direct effects of actions, their executability conditions, and their indirect effects (also called state constraints). The succession of moments in the evolution of the

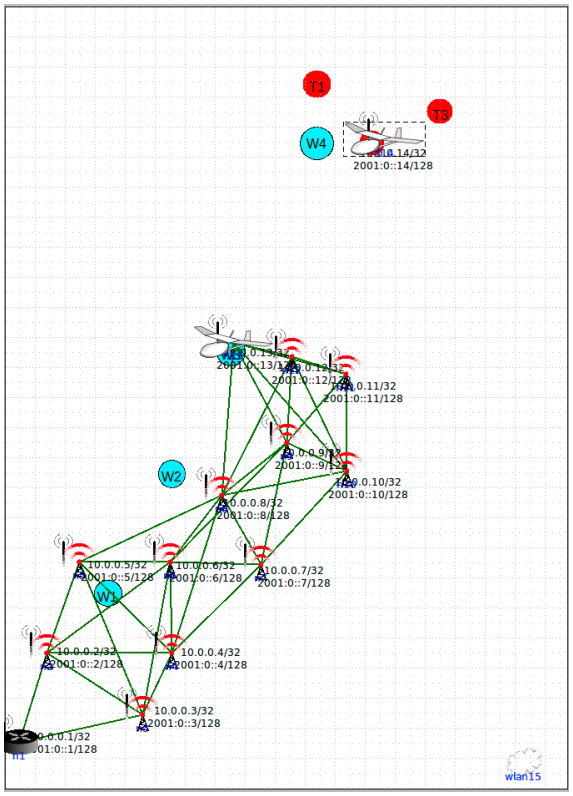
[2] The tasks in the various boxes are executed only when necessary.



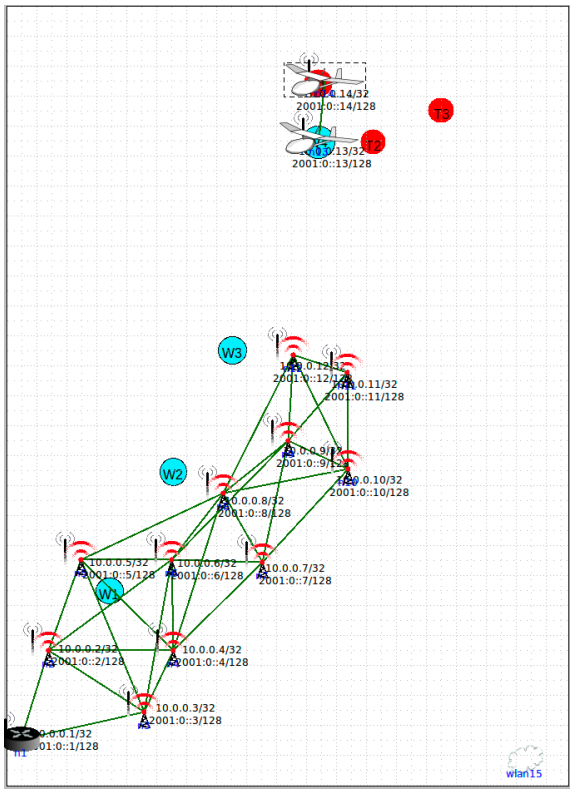
(a) Step 5: u_1 is disconnected from home base.



(b) Step 6: u_2 connects with u_1 and transfers images t_2 and t_3.



(c) Step 7: u_2 reconnects with relays, transfers images to the home base.



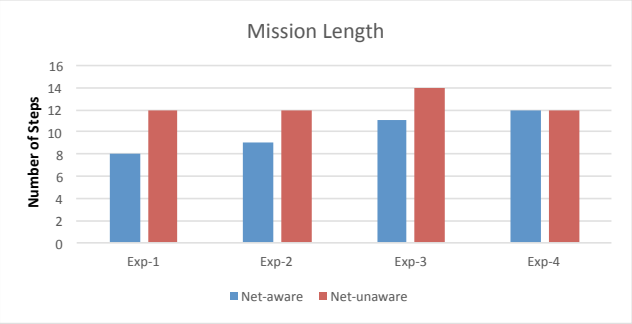
(d) Step 8: u_2 reconnects with u_1 to relay images of t_1.

Figure 4: Example instance 1 illustrating "data mule" information relaying between u_1 and u_2.

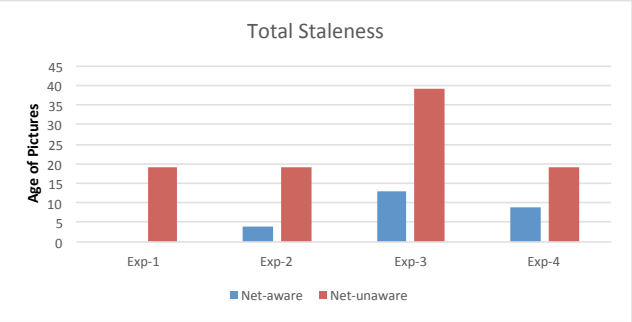
Input: M: mission plan;
G: mission goal;
Vars: H : history;
P: current plan;

$P := M$;
$H :=$ New_Observations();
while $\neg$Goal_Achieved(H, G) **do**
  if Unexpected_Observations(H) **then**
    $H :=$ Explain_Observations(H);
    $P :=$ Compute_Plan(G, H, P);
  end if
  $A :=$ Next_Action(P);
  $P :=$ Tail(P);
  Execute(A);
  $H :=$ Record_Execution(H, A);
  $H := H \cup$ New_Observations();
loop

Figure 2: Agent Control Loop.



(a) Length of the mission in time steps for the example instances.



(b) The total staleness of the image transfers.

Figure 3: Performance comparison.

environment is characterized by discrete *steps*, associated with non-negative integers. The fact that a certain fluent f is true at a step s is encoded by an atom $h(f, s)$. If f is false, this is expressed by $\neg h(f, s)$. The occurrence of an action $a \in \mathcal{A}$ at step s is represented as $o(a, s)$.

The history of the environment is formalized in ASP by two types of statements: $obs(f, true, s)$ states that f was observed to be true at step s (respectively, $obs(f, false, s)$ states that f was false); $hpd(a, s)$ states that a was observed to occur at s. Because in the this paper other agents' actions are not observable, the latter expression is used only to record an agent's own actions.

Objects in the UAV domain discussed in this paper are the home base, a set of fixed relays, a set of UAVs, a set of targets, and a set of waypoints. The waypoints are used to simplify the path-planning task, which we do not consider in the present work. The locations that the UAVs can occupy and travel to are the home base, the waypoints, and the locations of targets and fixed relays. The current location, l, of UAV u is represented by a fluent $at(u, l)$. For each location, the collection of its neighbors is defined by relation $next(l, l')$. UAV motion is restricted to occur only from a location to a neighboring one. The direct effect of action $move(u, l)$, intuitively stating that UAV u moves to location l, is described by the rule:

$$h(at(U, L2), S+1) \leftarrow o(move(U, L2), S), h(at(U, L1), S), next(L1, L2).$$

The fact that two radio nodes are in radio contact is encoded by fluent $in\_contact(r_1, r_2)$. The next two rules provide a recursive definition of the fluent, represented by means of state constraints:

$$h(in\_contact(R1, R2), S) \leftarrow R1 \neq R2, \neg h(down(R1), S), \neg h(down(R2), S), h(at(R1, L1), S), h(at(R2, L2), S), range(Rg), dist2(L1, L2, D), D \leq Rg^2.$$

$$h(in\_contact(R1, R3), S) \leftarrow R1 \neq R2, R2 \neq R3, R1 \neq R3, \neg h(down(R1), S), \neg h(down(R2), S). h(at(R1, L1), S), h(at(R2, L2), S), range(Rg), dist2(L1, L2, D), D \leq Rg^2, h(in\_contact(R2, R3), S),$$

The first rule defines the base case of two radio nodes that are directly in range of each other. Relation $dist2(l_1, l_2, d)$ calculates the square of the distance between two locations. Fluent $down(r)$ holds if radio r is known to be out-of-order, and a suitable axiom (not shown) defines the closed-world assumption on it. In the formalization, $in\_contact(R1, R2)$ is a *defined positive fluent*, i.e., a fluent whose truth value, in each state, is completely defined by the current value of other fluents, and is not subject to inertia. The formalization of $in\_contact(R1, R2)$ is thus completed by a rule capturing

the closed-world assumption on it:

$$\neg h(in\_contact(R1, R2), S) \leftarrow \\ R1 \neq R2, \\ \text{not } h(in\_contact(R1, R2), S).$$

Functions Goal_Achieved and Unexpected_Observations, in Figure 2a, respectively check if the goal has been achieved, and whether the history observed by the agent contains any unexpected observations. Following the definitions from (Balduccini and Gelfond 2003), observations are unexpected if they contradict the agent's expectations about the corresponding state of the environment. This definition is captured by the *reality-check axiom*, consisting of the constraints:

$$\leftarrow obs(F, true, S),\ \neg h(F, S).$$
$$\leftarrow obs(F, false, S),\ h(F, S).$$

Function Explain_Observations uses a diagnostic process along the lines of (Balduccini and Gelfond 2003) to identify a set of exogenous actions (actions beyond the control of the agent that may occur unobserved), whose occurrence explains the observations. To deal with the complexities of reasoning in a dynamic, multi-agent domain, the present work extends the previous results on diagnosis by considering multiple types of exogenous actions, and preferences on the resulting explanations. The simplest type of exogenous action is $break(r)$, which occurs when radio node r breaks. This action causes fluent $down(r)$ to become true. Actions of this kind may be used to explain unexpected observations about the lack of radio contact. However, the agent must also be able to cope with the limited observability of the position and motion of the other agents. This is accomplished by encoding commonsensical statements (encoding omitted) about the behavior of other agents, and about the factors that may affect it. The first such statement says that *a UAV will normally perform the mission plan, and will stop performing actions when its portion of the mission plan is complete.* Notice that a mission plan is simply a sequence of actions. There is no need to include pre-conditions for the execution of the actions it contains, because those can be easily identified by each agent, at execution time, from the formalization of the domain.

The agent is allowed to hypothesize that *a UAV may have stopped executing the mission plan* (for example, if the UAV malfunctions or is destroyed). *Normally, the reasoning agent will expect a UAV that aborts execution to remain in its latest location.* In certain circumstances, however, a UAV may need to deviate completely from the mission plan. To accommodate for this situation, the agent may hypothesize that *a UAV began behaving in an unpredictable way (from the agent's point of view) after aborting plan execution*. The following choice rule allows an agent to consider all of the possible explanations:

$$\{\ hpd(break(R), S), hpd(aborted(U, S)), \\ hpd(unpredictable(U, S))\ \}.$$

A constraint ensures that unpredictable behavior can be considered only if a UAV is believed to have aborted the plan. If that happens, the following choice rule is used to consider all possible courses of actions from the moment the UAV became unpredictable to the current time step.

$$\{hpd(move(U, L), S') : S' \geq S : S' < currstep\} \leftarrow \\ hpd(unpredictable(U, S)).$$

In practice, such a thought process is important to enable coordination with other UAVs when communications between them are impossible, and to determine the side-effects of the inferred courses of actions and potentially take advantage of them (e.g., "the UAV must have flown by target t_3. Hence, it is no longer necessary to take a picture of t_3"). A *minimize* statement ensures that only cardinality-minimal diagnoses are found:

$$\#minimize[hpd(break(R), S), \\ hpd(aborted(U, S)), \\ hpd(unpredictable(U, S))].$$

An additional effect of this statement is that the reasoning agent will prefer simpler explanations, which assume that a UAV aborted the execution of the mission plan and stopped, over those hypothesizing that the UAV engaged in an unpredictable course of actions.

Function Compute_Plan, as well as the mission planner, compute a new plan using a rather traditional approach, which relies on a choice rule for generation of candidate sequences of actions, constraints to ensure the goal is achieved, and *minimize* statements to ensure optimality of the plan with respect to the given metrics.

The next paragraphs outline two experiments, in increasing order of sophistication, which demonstrate the features of our approach, including non-trivial emerging interactions between the UAVs and the ability to work around unexpected problems autonomously.

Example Instance 1. Consider the environment shown in in Figure 4. Two UAVs, u_1 and u_2 are initially located at the home base in the lower left corner. The home base, relays and targets are positioned as shown in the figure, and the radio range is set to 7 grid units.

The mission planner finds a plan in which the UAVs begin by traveling toward the targets. While u_1 visits the first two targets, u_2 positions itself so as to be in radio contact with u_1 (Figures 4a and 4b). Upon receipt of the pictures, u_2 moves to within range of the relays to transmit the pictures to the home base (Figure 4c). At the same time, u_1 flies toward the final target. UAV u_2, after transmitting pictures to home base, moves to re-establish radio contact with u_1 and to receive the picture of t_3 (Figure 4d). Finally, u_2 moves within range of the relays to transmit picture of t_3 to the home base.

Remarkably, in this problem instance the plan establishes u_2 as a "data mule" in order to cope with the network limits. The "data mule" behavior is well-known in sensor network applications (Shah et al. 2003; Jea, Somasundara, and Srivastava 2005); however, no description of such behavior is included in our planner. Rather, the behavior emerges as a result of the reasoning process. The data-mule behavior is adopted by the planner because it optimizes the evaluation metrics (mission length and total staleness).

Example Instance 2. Now consider a more challenging and realistic example (Figure 5), in which the UAVs must cope

with unexpected events occurring during mission execution. Environment and mission goals are as above.

The mission planner produces the same plan described earlier[3], in which u_2 acts as a "data mule." The execution of the plan begins as expected, with u_1 reaching the area of the targets and u_2 staying in radio contact with it in order to receive the pictures of the first two targets (Figure 5a). When u_2 flies back to re-connect with the relays, however, it observes ("Observe" step of the architecture from Figure 1b) that the home base is unexpectedly not in radio contact. Hence, u_2 uses the available observations to determine plausible causes ("Explain" step of the architecture). In this instance, u_2 observes that relays r_5, r_6, r_7 and all the network nodes South of them are not reachable via the network. Based on knowledge of the layout of the network, u_2 determines that the simplest plausible explanation is that those three relays must have stopped working while u_2 was out of radio contact (e.g., started malfunctioning or have been destroyed).[4] Next, u_2 replans ("Local Planner" step of the architecture). *The plan is created based on the assumption that u_1 will continue executing the mission plan. This assumption can be later withdrawn if observations prove it false.* Following the new plan, u_2 moves further South towards the home base (Figure 5c). Simultaneously, u_1 continues with the execution of the mission plan, unaware that the connectivity has changed and that u_2 has deviated from the mission plan. After successfully relaying the pictures to the home base, u_2 moves back towards u_1. UAV u_1, on the other hand, reaches the expected rendezvous point, and observes that u_2 is not where expected (Figure 5d). UAV u_1 does not know the actual position of u_2, but its absence is evidence that u_2 must have deviated from the plan at some point in time. Thus, u_1's must now replan. Not knowing u_2's state, u_1's plan is to fly South to relay the missing picture to the home base on its own. This plan still does not deal with the unavailability of r_5, r_6, r_7, since u_1 has not yet had a chance to get in radio contact with the relays and observe the current network connectivity state. The two UAVs continue with the execution of their new plans and eventually meet, unexpectedly for both (Figure 5e). At that point, they automatically share the final picture. Both now determine that the mission can be completed by flying South past the failed relays, and execute the corresponding actions.

Experimental Comparison. As mentioned earlier, we believe that our network-aware approach to reasoning provides advantages over the state-of-the-art techniques that either disregard the network, or assume perfect communications. Figure 3b provides an overview of a quantitative experimental demonstration of such advantages. The figure compares our approach with the one in which the network is disregarded, in terms of mission length and total staleness.[5] The optimistic approach is not considered, because its brittleness makes it not viable for actual applications. The comparison includes the two example instances discussed earlier (labeled Exp-2 and Exp-4). Of the other two experiments, Exp-1 is a variant of Exp-2 that can be solved with the data-mule in a static position, while Exp-3 is a variant of Exp-2 with 5 targets. As can be seen, the network-aware approach is always superior. In Exp-1, the UAV acting as a data-mule extends the range of the network so that all the pictures are instantly relayed to the home base, reducing total staleness to 0. In Exp-4, it is worth stressing that the network, which the UAVs rely upon when using our approach, suddenly fails. One would expect the network-*unaware* approach to have an advantage under these circumstances, but, as demonstrated by the experimental results, our approach still achieves a lower total staleness of the pictures thanks to its ability to identify the network issues and to work around them.

From a practical perspective, the execution times of the various reasoning tasks have been extremely satisfactory, taking only fractions of a second on a modern desktop computer running the CLASP solver (Gebser, Kaufmann, and Schaub 2009), even in the most challenging cases.

Simulation and Experimental Setup

The simulation for the experimental component of this work was built using the Common Open Research Emulator (CORE) (Ahrenholz 2010). CORE is a real-time network emulator that allows users to create lightweight virtual nodes with full-fledged network communications stack. CORE virtual nodes can run unmodified Linux applications in real-time. The CORE GUI incorporates a basic range-based model to emulate networks typical in mobile ad-hoc network (MANET) environments. CORE provides an interface for creating complex network topologies, node mobility in an environment, and access to the lower-level network conditions, e.g., network connectivity. Using CORE as a real-time simulation environment allows agents, represented as CORE nodes, to execute mission plans in realistic radio environments. For this work, CORE router nodes represent the home base, relays, and UAVs. The nodes are interconnected via an ad-hoc wireless network. As the UAVs move in the environment, CORE updates the connectivity between other UAVs and relays based on the range dictated by the built-in wireless model. The radio network model has limited range and bandwidth capacity. Each node runs the Optimized Link-State Routing protocol (OLSR) (Jacquet et al. 2001), a unicast MANET routing algorithm, which maintains the routing tables across the nodes. The routing table makes it possible to determine if a UAV can exchange information with other radio nodes at any given moment. Using CORE allows us to account for realistic communications in ways not possible with multi-agent simulators such as AgentFly (David Sislak and Pechoucek 2012).

[3]The careful reader may notice from the figures that the trajectory used to visit the targets is the mirror image of the one from the previous example. The corresponding plans are equivalent from the point of view of all the metrics, and the specific selection of one over the other is due to randomization used in the search process.

[4]As shown in Figure 5b this is indeed the case in our experimental set-up, although it need not be. Our architecture is capable of operating under the *assumption* that its hypotheses are correct, and later re-evaluate the situation based on further observations, and correct its hypotheses and re-plan if needed.

[5]For simplicity we measure mission length and staleness in time steps, but it is not difficult to add action durations.

Conclusion and Future Work

This paper discussed a novel application of an ASP-based intelligent agent architecture to the problem of UAV coordination. The UAV scenarios considered in this paper are bound to be increasingly common as more levels autonomy are required to create large-scale systems. Prior work on distributed coordination and planning has mostly overlooked or simplified communications dynamics, at best treating communications as a resource or other planning constraint.

Our work demonstrates the reliability and performance gains deriving from network-aware reasoning. In our experimental evaluation, our approach yielded a reduction in mission length of up to 30% and in total staleness between 50% and 100%. We expect that, in more complex scenarios, the advantage of a realistic networking model will be even more evident. In our experiments, execution time was always satisfactory, and we believe that several techniques from the state-of-the-art can be applied to curb the increase in execution time as the scenarios become more complex. For the future, we intend to extend the mission-aware networking layer with advanced reasoning capabilities, integrate network-aware reasoning and mission-aware networking tightly, and execute experiments demonstrating the advantages of such a tight integration.

References

Ahrenholz, J. 2010. Comparison of CORE network emulation platforms. In *IEEE Military Communications Conf.*

Balduccini, M., and Gelfond, M. 2003. Diagnostic reasoning with A-Prolog. *Journal of Theory and Practice of Logic Programming (TPLP)* 3(4–5):425–461.

Balduccini, M., and Gelfond, M. 2008. The AAA Architecture: An Overview. In *AAAI Spring Symp.: Architectures for Intelligent Theory-Based Agents*.

Baral, C., and Gelfond, M. 2000. Reasoning Agents In Dynamic Domains. In *Workshop on Logic-Based Artificial Intelligence*, 257–279. Kluwer Academic Publishers.

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

Blount, J.; Gelfond, M.; and Balduccini, M. 2014. Towards a Theory of Intentional Agents. In *Knowledge Representation and Reasoning in Robotics*, AAAI Spring Symp. Series.

David Sislak, Premysl Volf, S. K., and Pechoucek, M. 2012. *AgentFly: Scalable, High-Fidelity Framework for Simulation, Planning and Collision Avoidance of Multiple UAVs*. Wiley Inc. chapter 9, 235–264.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. The Conflict-Driven Answer Set Solver clasp: Progress Report. In *Logic Programming and Nonmonotonic Reasoning*.

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(2–4):301–321.

Jacquet, P.; Muhlethaler, P.; Clausen, T.; Laouiti, A.; Qayyum, A.; and Viennot, L. 2001. Optimized link state routing protocol for ad hoc networks. In *IEEE INMIC: Technology for the 21st Century*.

Jea, D.; Somasundara, A.; and Srivastava, M. 2005. Multiple controlled mobile elements (data mules) for data collection in sensor networks. *Distr. Computing in Sensor Sys.*

Kopeikin, A. N.; Ponda, S. S.; Johnson, L. B.; and How, J. P. 2013. Dynamic Mission Planning for Communication Control in Multiple Unmanned Aircraft Teams. *Unmanned Systems* 1(1):41–58.

Marek, V. W., and Truszczynski, M. 1999. *The Logic Programming Paradigm: a 25-Year Perspective*. Springer Verlag, Berlin. chapter Stable Models and an Alternative Logic Programming Paradigm, 375–398.

McCarthy, J. 1998. Elaboration Tolerance.

Niemelä, I., and Simons, P. 2000. *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers. chapter Extending the Smodels System with Cardinality and Weight Constraints.

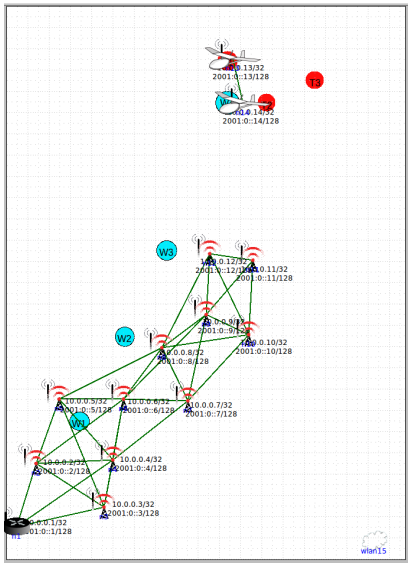
Pynadath, D. V., and Tambe, M. 2002. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *JAIR* 16:389–423.

Rao, A. S., and Georgeff, M. P. 1991. Modeling Rational Agents within a BDI-Architecture. In *Proc. of the Int'l Conf. on Principles of Knowledge Representation and Reasoning*.

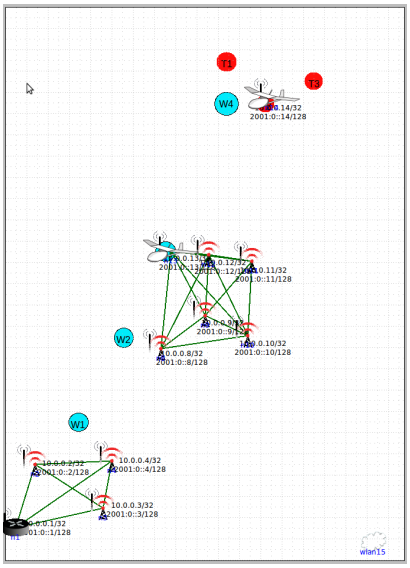
Shah, R. C.; Roy, S.; Jain, S.; and Brunette, W. 2003. Data MULEs: modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks* 1(2-3).

Usbeck, K.; Cleveland, J.; and Regli, W. C. 2012. Network-centric ied detection planning. *IJIDSS* 5(1):44–74.

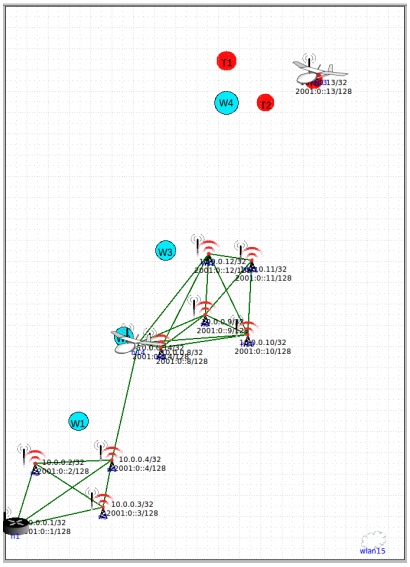
Wooldridge, M. 2000. *Reasoning about Rational Agents*. MIT Press.



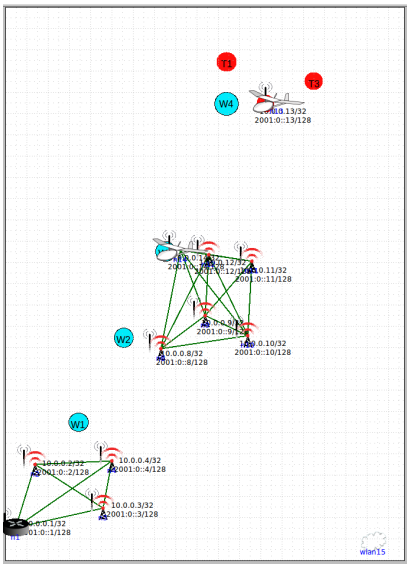
(a) Step 5: u_1 is transmitting images to u_2.



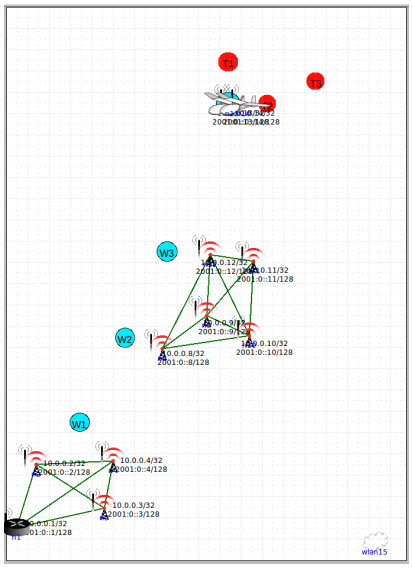
(b) Step 6: u_2 moves toward relays. Relay nodes 5, 6, and 7 have failed.



(c) Step 7: u_2 re-plans and moves closer to home base.



(d) Step 8: u_2 moves toward u_1.



(e) Step 9: u_2 and u_1 reconnect and move back toward home base.

Figure 5: Example instance 2 illustrates re-planning after relay node failure between steps 5 and 6 forcing the UAVs to re-plan.

Implementing Default and Autoepistemic Logics via the Logic of GK

Jianmin Ji
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China

Hannes Strass
Computer Science Institute
Leipzig University
Leipzig, Germany

Abstract

The logic of knowledge and justified assumptions, also known as the logic of grounded knowledge (GK), was proposed by Lin and Shoham as a general logic for nonmonotonic reasoning. To date, it has been used to embed in it default logic (propositional case), autoepistemic logic, Turner's logic of universal causation, and general logic programming under stable model semantics. Besides showing the generality of GK as a logic for nonmonotonic reasoning, these embeddings shed light on the relationships among these other logics. In this paper, for the first time, we show how the logic of GK can be embedded into disjunctive logic programming in a polynomial but non-modular translation with new variables. The result can then be used to compute the extension/expansion semantics of default logic, autoepistemic logic and Turner's logic of universal causation by disjunctive ASP solvers such as GNT, cmodels, DLV, and claspD(-2).

Introduction

Lin and Shoham [1992] proposed a logic with two modal operators $\mathbf{K}$ and $\mathbf{A}$, standing for knowledge and assumption, respectively. The idea is that one starts with a set of assumptions (those true under the modal operator $\mathbf{A}$), computes the minimal knowledge under this set of assumptions, and then checks to see if the assumptions were justified in that they agree with the resulting minimal knowledge. For instance, consider the GK formula $\mathbf{A}p \supset \mathbf{K}p$. If we assume p, then we can conclude that we know p, thus the assumption that p holds is justified, and we get a GK model where both $\mathbf{A}p$ and $\mathbf{K}p$ are true. (There is another GK model where we do not assume p and hence do not know p.) However, there is no GK model of $\neg\mathbf{A}p \supset \mathbf{K}p$: if we do not assume p, we are forced to conclude $\mathbf{K}p$, but then knowledge and assumptions do not coincide; if we do assume p, we cannot conclude that we know p and thus assuming p was not justified.

To date, there have been embeddings from default logic [Reiter, 1980] and autoepistemic logic [Moore, 1985] to the logic of GK [Lin and Shoham, 1992], from Turner's logic of universal causation [Turner, 1999] to the logic of GK [Ji and Lin, 2012], as well as from general logic programs [Ferraris, 2005] to the logic of GK [Lin and Zhou, 2011]. Among other things, these embeddings shed new light on nonmonotonic reasoning, and have led to an interesting characterization of strong equivalence in logic programming [Lin, 2002; Lin and Zhou, 2011], and helped relate logic programming to circumscription [Lin and Shoham, 1992] as the semantics of GK is just a minimization (of knowledge) together with an identity check (of assumptions and knowledge) after the minimization.

In this paper, for the first time, we consider computing models of GK theories by disjunctive logic programs. We shall propose a polynomial translation from a (pure) GK theory to a disjunctive logic program such that there is a one-to-one correspondence between GK models of the GK theory and answer sets of the resulting disjunctive logic program. The result can then be used to compute the extension/expansion semantics of default logic, autoepistemic logic and Turner's logic of universal causation by disjunctive ASP solvers such as GNT [Janhunen and Niemelä, 2004], cmodels [Giunchiglia, Lierler, and Maratea, 2006], DLV [Leone et al., 2006], claspD [Drescher et al., 2008] and claspD-2 [Gebser, Kaufmann, and Schaub, 2013]. In particular, the recent advances in disjunctive answer set solving [Gebser, Kaufmann, and Schaub, 2013] open up promising research avenues towards applications of expressive nonmonotonic knowledge representation languages.

To substantiate this claim, we have implemented the translation and report on some preliminary experiments that we conducted on the special case of computing extensions for Reiter's default logic [Reiter, 1980]. The implementation, called gk2dlp, is available for download from the second author's home page.[1]

Providing implementations for theoretical formalisms has a long tradition in nonmonotonic reasoning, for an overview see [Dix, Furbach, and Niemelä, 2001]. In fact, nonmonotonic reasoning itself originated from a desire to more accurately model the way humans reason, and was since its conception driven by applications in commonsense reasoning [McCarthy, 1980, 1986]. Today, thanks to extensive research efforts, we know how closely interrelated the different formalisms for nonmonotonic reasoning are, and can use this knowledge to improve the scope of implementations.

This paper is organized as follows. Section 2 reviews logic programs, the logic of GK and default and autoepistemic logics. Section 3 presents our main result, the map-

[1] http://informatik.uni-leipzig.de/~strass/gk2dlp/

ping from GK to disjunctive logic programming. Section 4 presents our prototypical implementation, several experiments we conducted to analyze the translation, possible applications for it, and a comparison with previous and related work. Section 5 concludes with ideas for future work.

Preliminaries

We assume a propositional language with two zero-place logical connectives $\top$ for tautology and $\bot$ for contradiction. We denote by $Atom$ the set of atoms, the signature of our language, and Lit the set of literals: $Lit = Atom \cup \{\neg p \mid p \in Atom\}$. A set I of literals is called *complete* if for each atom p, exactly one of $\{p, \neg p\}$ is in I.

In this paper, we identify an interpretation with a complete set of literals. If I is a complete set of literals, we use it as an interpretation when we say that it is a model of a formula, and we use it as a set of literals when we say that it entails a formula. In particular, we denote by $Th(I)$ the logical closure of I (considered to be a set of literals).

Logic Programming

A *nested expression* is built from literals using the 0-place connectives $\top$ and $\bot$, the unary connective "not" and the binary connectives "," and ";" for conjunction and disjunction. A *logic program* with nested expressions is a finite set of rules of the form $F \leftarrow G$, where F and G are nested expressions. The *answer set* of a logic program with nested expressions is defined as in [Lifschitz, Tang, and Turner, 1999]. Given a nested expression F and a set S of literals, we define when S satisfies F, written $S \models F$ below, recursively as follows (l is a literal):

- $S \models l$ if $l \in S$,
- $S \models \top$ and $S \not\models \bot$,
- $S \models not\, F$ if $S \not\models F$,
- $S \models F, G$ if $S \models F$ and $S \models G$, and
- $S \models F; G$ if $S \models F$ or $S \models G$.

S satisfies a rule $F \leftarrow G$ if $S \models F$ whenever $S \models G$. S satisfies a logic program P, written $S \models P$, if S satisfies all rules in P.

The *reduct* P^S of P related to S is the result of replacing every maximal subexpression of P that has the form $not\, F$ with $\bot$ if $S \models F$, and with $\top$ otherwise. For a logic program P without not, the *answer set* of P is any minimal consistent subset S of Lit that satisfies P. We use $\Gamma_P(S)$ to denote the set of answer sets of P^S. Now a consistent set S of literals is an *answer set* of P iff $S \in \Gamma_P(S)$. Every logic program with nested expressions can be equivalently translated to disjunctive logic programs with disjunctive rules of the form

$$l_1; \cdots; l_k \leftarrow l_{k+1}, \ldots, l_t, not\, l_{t+1}, \ldots, not\, l_m, \\ not\, not\, l_{m+1}, \ldots, not\, not\, l_n$$

where $n \geq m \geq t \geq k \geq 0$ and $l_1, \ldots, l_n$ are propositional literals.

Default Logic

Default logic [Reiter, 1980] is for making and withdrawing assumptions in the light of incomplete knowledge. This is done by *defaults*, that allow to express rules of thumb such as "birds usually fly" and "tools usually work." For a given logical language, a *default* is any expression of the form $\phi : \psi_1, \ldots, \psi_n / \varphi$ where $\phi, \psi_1, \ldots, \psi_n, \varphi$ are formulas of the underlying language. A *default theory* is a pair (W, D), where W is a set of formulas and D is a set of defaults. The meaning of default theories is given through the notion of *extensions*. An extension of a default theory (W, D) is "interpreted as an acceptable set of beliefs that one may hold about the incompletely specified world W" [Reiter, 1980]. For a default theory (W, D) and any set S of formulas let $\Gamma(S)$ be the smallest set satisfying (1) $W \subseteq \Gamma(S)$, (2) $Th(\Gamma(S)) = \Gamma(S)$, (3) If $\phi : \psi_1, \ldots, \psi_n / \varphi \in D$, $\phi \in \Gamma(S)$ and $\neg\psi_1, \ldots, \neg\psi_n \notin S$, then $\varphi \in \Gamma(S)$. A set E of formulas is called an *extension for* (W, D) iff $\Gamma(E) = E$.

Autoepistemic Logic

Moore [1985] strives to formalize an ideally rational agent reasoning about its own beliefs. He uses a belief modality L to explicitly refer to the agent's belief within the language. Given a set A of formulas (the initial beliefs), a set T is an *expansion* of A if it coincides with the deductive closure of the set $A \cup \{L\varphi \mid \varphi \in T\} \cup \{\neg L\varphi \mid \varphi \notin T\}$. In words, T is an expansion if it equals what can be derived using the initial beliefs A and positive and negative introspection with respect to T itself. It was later discovered that this definition of expansions allows unfounded, self-justifying beliefs. Such beliefs are however not always desirable when representing the knowledge of agents.

The Logic of GK

The language of GK proposed by Lin and Shoham [1992] is a modal propositional language with two modal operators, $\mathbf{K}$, for knowledge, and $\mathbf{A}$, for assumption. GK *formulas* φ are propositional formulas with $\mathbf{K}$ and $\mathbf{A}$, that is,

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{K}\varphi \mid \mathbf{A}\varphi$$

where p is an atom. A GK *theory* is a set of GK formulas.

GK is a nonmonotonic logic, and its semantics is defined using the standard Kripke possible world interpretations. Informally speaking, a GK model is a Kripke interpretation where what is true under $\mathbf{K}$ is minimal and exactly the same as what is true under $\mathbf{A}$. The intuition here is that given a GK formula, one first makes some assumptions (those true under $\mathbf{A}$), then one minimizes the knowledge thus entailed, and finally checks to make sure that the initial assumption is justified in the sense that the minimal knowledge is the same as the initial assumption.

Formally, a *Kripke interpretation* M is a tuple $\langle W, \pi, R_K, R_A, s\rangle$, where W is a nonempty set of *possible worlds*, π a function that maps a possible world to an interpretation, R_K and R_A binary relations over W representing the accessibility relations for $\mathbf{K}$ and $\mathbf{A}$, respectively, and $s \in W$, called the *actual world* of M. The *satisfaction relation* $\models$ between a Kripke interpretation

$M = \langle W, \pi, R_K, R_A, s\rangle$ and a GK formula φ is defined in a standard way:

- $M \not\models \bot$,
- $M \models p$ iff $p \in \pi(s)$, where p is an atom,
- $M \models \neg\varphi$ iff $M \not\models \varphi$,
- $M \models \varphi \wedge \psi$ iff $M \models \varphi$ and $M \models \psi$,
- $M \models \varphi \vee \psi$ iff $M \models \varphi$ or $M \models \psi$,
- $M \models \mathbf{K}\varphi$ iff $\langle W, \pi, R_K, R_A, w\rangle \models \varphi$ for any $w \in W$ such that $(s, w) \in R_K$,
- $M \models \mathbf{A}\varphi$ iff $\langle W, \pi, R_K, R_A, w\rangle \models \varphi$ for any $w \in W$ such that $(s, w) \in R_A$.

Note that for any $w \in W$, $\pi(w)$ is an interpretation. We say that a Kripke interpretation M is a *model* of a GK formula φ if M satisfies φ, M is a *model* of a GK theory T if M satisfies every GK formula in T. In the following, given a Kripke interpretation M, we let

$$\mathbf{K}(M) = \{\, \phi \mid \phi \text{ is a propositional formula and } M \models \mathbf{K}\phi \,\},$$
$$\mathbf{A}(M) = \{\, \phi \mid \phi \text{ is a propositional formula and } M \models \mathbf{A}\phi \,\}.$$

Notice that $\mathbf{K}(M)$ and $\mathbf{A}(M)$ are always closed under classical logical entailment – they are propositional theories.

Given a GK formula T, a Kripke interpretation M is a minimal model of T if M is a model of T and there does not exist another model M_1 of T such that $\mathbf{A}(M_1) = \mathbf{A}(M)$ and $\mathbf{K}(M_1) \subsetneq \mathbf{K}(M)$. We say that M is a *GK model* of T if M is a minimal model of T and $\mathbf{K}(M) = \mathbf{A}(M)$.

In this paper, we consider only GK formulas that do not contain nested occurrences of modal operators. Specifically, an $\mathbf{A}$-*atom* is a formula of the form $\mathbf{A}\phi$ and a $\mathbf{K}$-*atom* is a formula of the form $\mathbf{K}\phi$, where ϕ is a propositional formula. A GK formula is called a *pure GK formula* if it is formed from $\mathbf{A}$-atoms, $\mathbf{K}$-atoms and propositional connectives. Similarly, a *pure GK theory* is a set of pure GK formulas. Given a pure GK formula F, we denote

$$Atom_{\mathbf{K}}(F) = \{\, \phi \mid \mathbf{K}\phi \text{ is a } \mathbf{K}\text{-atom occurring in } F \,\},$$
$$Atom_{\mathbf{A}}(F) = \{\, \phi \mid \mathbf{A}\phi \text{ is an } \mathbf{A}\text{-atom occurring in } F \,\}.$$

For a pure GK theory T, we use $Atom_{\mathbf{K}}(T) = \bigcup_{F\in T} Atom_{\mathbf{K}}(F)$ and $Atom_{\mathbf{A}}(T) = \bigcup_{F\in T} Atom_{\mathbf{A}}(F)$ to denote their modal atoms.

So far, the applications of the logic of GK only ever use pure GK formulas. We now present some embeddings of well-known nonmonotonic knowledge representation languages into the logic of GK.

Default logic A (propositional) default theory $\Delta = (W, D)$ (under extension semantics) is translated into pure GK formulas in the following way: (1) Translate each $\phi \in W$ to $\mathbf{K}\phi$; (2) translate each $(\phi : \psi_1, \ldots, \psi_n/\varphi) \in D$ to $\mathbf{K}\phi \wedge \neg\mathbf{A}\neg\psi_1 \wedge \cdots \wedge \neg\mathbf{A}\neg\psi_n \supset \mathbf{K}\varphi$. For the weak extension semantics, a default $(\phi : \psi_1, \ldots, \psi_n/\varphi) \in D$ is translated to $\mathbf{A}\phi \wedge \neg\mathbf{A}\neg\psi_1 \wedge \cdots \wedge \neg\mathbf{A}\neg\psi_n \supset \mathbf{K}\varphi$.

Autoepistemic logic An L-sentence of autoepistemic logic that is in normal form [Konolige, 1988], that is, a disjunction of the form $\neg L\phi \vee L\psi_1 \vee \cdots \vee L\psi_n \vee \varphi$, is (under expansion semantics) expressed as $\mathbf{A}\phi \wedge \neg\mathbf{A}\psi_1 \wedge \cdots \wedge \neg\mathbf{A}\psi_n \supset \mathbf{K}\varphi$. For strong expansion semantics, it becomes $\mathbf{K}\phi \wedge \neg\mathbf{A}\psi_1 \wedge \cdots \wedge \neg\mathbf{A}\psi_n \supset \mathbf{K}\varphi$.

Notice that the translation of default and autoepistemic theories into the logic of GK is compatible with Konolige's translation from default logic into autoepistemic logic [Konolige, 1988]. Indeed, Konolige's translation perfectly aligns the weak extension semantics of default logic with expansion semantics for autoepistemic logic, and likewise for extension and strong expansion semantics [Denecker, Marek, and Truszczyński, 2003].

Logic of universal causation The logic of universal causation is a nonmonotonic propositional modal logic with one modality $\mathbf{C}$ [Turner, 1999]. A formula of this logic is translated to the pure logic of GK by replacing every occurrence of $\mathbf{C}$ by $\mathbf{K}$, adding $\mathbf{A}$ before each atom which is not in the range of $\mathbf{C}$ in it, and adding $\mathbf{A}p \vee \mathbf{A}\neg p$ for each atom p. For example, if a UCL formula is $(p \wedge \neg q) \supset \mathbf{C}(p \wedge \neg q)$ and $Atom = \{p, q\}$, then the corresponding pure GK formula is $((\mathbf{A}p \wedge \neg\mathbf{A}q) \supset \mathbf{K}(p \wedge \neg q)) \wedge (\mathbf{A}p \vee \mathbf{A}\neg p) \wedge (\mathbf{A}q \vee \mathbf{A}\neg q)$.

Disjunctive logic programs A disjunctive LP rule

$$p_1 \vee \cdots \vee p_k \leftarrow p_{k+1}, \ldots, p_l, not\, p_{l+1}, \ldots, not\, p_m,$$

where p's are atoms, corresponds to the pure GK formula:

$$\mathbf{K}p_{k+1} \wedge \cdots \wedge \mathbf{K}p_l \wedge \neg\mathbf{A}p_{l+1} \wedge \cdots \wedge \neg\mathbf{A}p_m \supset \mathbf{K}p_1 \vee \cdots \vee \mathbf{K}p_k$$

Main Result: From Pure GK to Disjunctive ASP

Before presenting the translation, we introduce some notations. Let F be a pure GK formula, we use $tr_p(F)$ to denote the propositional formula obtained from F by replacing each occurrence of a $\mathbf{K}$-atom $\mathbf{K}\phi$ by k_ϕ and each occurrence of an $\mathbf{A}$-atom $\mathbf{A}\psi$ by a_ψ, where k_ϕ and a_ψ are new atoms with respect to ϕ and ψ respectively. For a pure GK theory T, we define $tr_p(T) = \bigwedge_{F\in T} tr_p(F)$. To illustrate these and the definitions that follow, we use a running example.

Example 1 (Normal Reiter default) *Consider the pure GK theory $\{F\}$ with $F = \neg\mathbf{A}\neg p \supset \mathbf{K}p$ corresponding to the default $\top : p/p$, and another pure GK theory $\{F, G\}$ with $G = \mathbf{K}\neg p$ corresponding to the default $\top : \top/\neg p$. Then $tr_p(\{F\}) = \neg a_{\neg p} \supset k_p$ and $tr_p(\{F, G\}) = (\neg a_{\neg p} \supset k_p) \wedge k_{\neg p}$, where $a_{\neg p}$, k_p, and $k_{\neg p}$ are new atoms.*

Here we introduce a set of new atoms k_ϕ and a_ψ for each formula $\phi \in Atom_{\mathbf{K}}(T)$ and $\psi \in Atom_{\mathbf{A}}(T)$. Intuitively, the new atom k_ϕ (resp. a_ψ) will be used to encode containment of the formula ϕ in $\mathbf{K}(M)$ (resp. $\mathbf{A}(M)$) of a GK model M for T.

Given a propositional formula ϕ and an atom a, we use ϕ^a to denote the propositional formula obtained from ϕ by replacing each occurrence of an atom p with a new atom p^a with respect to a. These formulas and new atoms will later be used in our main translation to perform the minimality check of the logic of GK's semantics.

We now stepwise work our way towards the main result. We start out with a result that relates a pure GK theory to

a propositional formula that will later reappear in our main translation.

Proposition 1 *Let T be a pure GK theory. A Kripke interpretation M is a model of T if and only if there exists a model I^* of the propositional formula Φ_T where*

$$\Phi_T = tr_p(T) \wedge \Phi_{snd} \wedge \Phi^{\mathbf{K}}_{wit} \wedge \Phi^{\mathbf{A}}_{wit} \text{ with}$$

$$\Phi_{snd} = \bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \phi^k) \wedge \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \phi^a)$$

$$\Phi^{\mathbf{K}}_{wit} = \bigwedge_{\psi \in Atom_{\mathbf{K}}(T)} \left(\neg k_\psi \supset \Phi^{\mathbf{K}}_\psi\right)$$

$$\Phi^{\mathbf{A}}_{wit} = \bigwedge_{\psi \in Atom_{\mathbf{A}}(T)} \left(\neg a_\psi \supset \Phi^{\mathbf{A}}_\psi\right)$$

$$\Phi^{\mathbf{K}}_\psi = \neg\psi^{k_\psi} \wedge \bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \phi^{k_\psi})$$

$$\Phi^{\mathbf{A}}_\psi = \neg\psi^{a_\psi} \wedge \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \phi^{a_\psi})$$

such that

- $\mathbf{K}(M) \cap Atom_{\mathbf{K}}(T) = \{\phi \mid \phi \in Atom_{\mathbf{K}}(T),\ I^* \models k_\phi\}$;
- $\mathbf{A}(M) \cap Atom_{\mathbf{A}}(T) = \{\phi \mid \phi \in Atom_{\mathbf{A}}(T),\ I^* \models a_\phi\}$.

The proposition examines the relationship between models of a pure GK theory and particular models of the propositional formula Φ_T. The first conjunct $tr_p(T)$ of the formula Φ_T indicates that the k-atoms and a-atoms in it can be interpreted in accordance with $\mathbf{K}(M)$ and $\mathbf{A}(M)$ such that $I^* \models tr_p(T)$ iff M is a model of T. The soundness formula Φ_{snd} achieves that the sets $\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\}$ and $\{\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } I^* \models a_\phi\}$ are consistent. The witness formulas Φ_{wit} indicate that, if $I^* \models \neg k_\psi$ for some $\psi \in Atom_{\mathbf{K}}(T)$ (resp. $\psi \in Atom_{\mathbf{A}}(T)$) then there exists a model I' of $\mathbf{K}(M)$ (resp. $\mathbf{A}(M)$) such that $I' \models \neg\psi$, where I' is explicitly indicated by newly introduced p^{k_ψ} (resp. p^{a_ψ}) atoms. So intuitively, if a formula is not known (or not assumed), then there must be a witness for that. This condition is necessary: for instance, the set $\{k_p, k_q, \neg k_{p\wedge q}\}$ satisfies the formula $(k_{p\wedge q} \supset k_p) \wedge (k_{p\wedge q} \supset k_q)$, however, since $\mathbf{K}(M)$ is a theory there does not exist a Kripke interpretation M such that $p \in \mathbf{K}(M), q \in \mathbf{K}(M)$ and $p \wedge q \notin \mathbf{K}(M)$.

Example 1 (Continued) *Formula $\Phi_{\{F\}}$ is given by:*

$$tr_p(\{F\}) = \neg a_{\neg p} \supset k_p$$

$$\Phi_{snd}(\{F\}) = (k_p \supset p^k) \wedge (a_{\neg p} \supset \neg p^a)$$

$$\Phi^{\mathbf{K}}_{wit}(\{F\}) = \neg k_p \supset (\neg p^{k_p} \wedge (k_p \supset p^{k_p}))$$

$$\Phi^{\mathbf{A}}_{wit}(\{F\}) = \neg a_{\neg p} \supset (\neg\neg p^{a_{\neg p}} \wedge (a_{\neg p} \supset \neg p^{a_{\neg p}}))$$

Formula $\Phi_{\{F,G\}}$ is given by:

$$tr_p(\{F,G\}) = (\neg a_{\neg p} \supset k_p) \wedge k_{\neg p}$$

$$\Phi_{snd}(\{F,G\}) = \Phi_{snd}(\{F\}) \wedge (k_{\neg p} \supset \neg p^k)$$

$$\Phi^{\mathbf{K}}_{wit}(\{F,G\}) = (\neg k_p \supset \Phi^{\mathbf{K}}_p) \wedge (\neg k_{\neg p} \supset \Phi^{\mathbf{K}}_{\neg p})$$

$$\Phi^{\mathbf{A}}_{wit}(\{F,G\}) = \Phi^{\mathbf{A}}_{wit}(\{F\})$$

$$\Phi^{\mathbf{K}}_p = \neg p^{k_p} \wedge (k_p \supset p^{k_p}) \wedge (k_{\neg p} \supset \neg p^{k_p})$$

$$\Phi^{\mathbf{K}}_{\neg p} = \neg\neg p^{k_{\neg p}} \wedge (k_p \supset p^{k_{\neg p}}) \wedge (k_{\neg p} \supset \neg p^{k_{\neg p}})$$

where p^k, p^a, p^{k_p}, $p^{a_{\neg p}}$, and $p^{k_{\neg p}}$ are new atoms. Note that formula $\Phi_{snd}(\{F,G\})$ prevents a model that satisfies both k_p and $k_{\neg p}$.

While Proposition 1 aligns Krikpe models and propositional models of the translation, there is yet no mention of GK's typical minimization step. This is the task of the next result, which extends the above relationship to GK models.

Proposition 2 *Let T be a pure GK theory. A Kripke interpretation M is a GK model of T if and only if there exists a model I^* of the propositional formula Φ_T such that*

- $\mathbf{K}(M) = \mathbf{A}(M) = Th\,(\,\{\phi \mid \phi \in Atom_{\mathbf{K}}(T), I^* \models k_\phi\}\,)$;
- *for each $\psi \in Atom_{\mathbf{A}}(T)$,*

  $I^* \models a_\psi$ *iff* $\psi \in Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T)$ *and* $I^* \models k_\phi\})$

- *there does not exist another model $I^{*\prime}$ such that*

$$I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\},$$
$$I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}.$$

Example 1 (Continued) *Clearly the intended reading of our running example $\{F\}$ is that there is no reason to assume that p is false, and the default lets us conclude that we know p. This is testified by the partial interpretation $I^* = \{\neg a_{\neg p}, k_p, p^k, p^{a_{\neg p}}\}$ (the remaining atoms are not relevant). It is easy to see that I^* is a model for $\Phi_{\{F\}}$ and there is no model $I^{*\prime}$ with the properties above. Now $k_p \in I^*$ shows that p is known in the corresponding GK model.*

Similarly, G provides a reason to assume that p is false and $\{F,G\}$ concludes that we know $\neg p$. Consider the partial interpretation $I^* = \{a_{\neg p}, \neg k_p, k_{\neg p}, \neg p^k, \neg p^a, \neg p^{k_p}\}$, it specifies a model for $\Phi_{\{F,G\}}$ and there is no model $I^{*\prime}$ with the properties above. In particular, $k_{\neg p} \in I^*$ shows that $\neg p$ is known in the corresponding GK model.

In Proposition 2, we only need to consider a Kripke interpretation M such that $\mathbf{A}(M) \cup \mathbf{K}(M)$ is consistent. This means that formula Φ_T can be modified to Ψ_T where

$$\Psi_T = tr_p(T) \wedge \Psi_{snd} \wedge \Psi^{\mathbf{K}}_{wit} \wedge \Psi^{\mathbf{A}}_{wit} \text{ with}$$

$$\Psi_{snd} = \bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \phi) \wedge \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \phi)$$

$$\Psi^{\mathbf{K}}_{wit} = \bigwedge_{\psi \in Atom_{\mathbf{K}}(T)} \left(\neg k_\psi \supset \Psi^{\mathbf{K}}_\psi\right)$$

$$\Psi^{\mathbf{A}}_{wit} = \bigwedge_{\psi \in Atom_{\mathbf{A}}(T)} \left(\neg a_\psi \supset \Psi^{\mathbf{A}}_\psi\right)$$

$$\Psi^{\mathbf{K}}_\psi = \neg\psi^{k_\psi} \wedge \bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \phi^{k_\psi}) \wedge \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \phi^{k_\psi})$$

$$\Psi^{\mathbf{A}}_\psi = \neg\psi^{a_\psi} \wedge \bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \phi^{a_\psi}) \wedge \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \phi^{a_\psi})$$

So the soundness formula Ψ_{snd} actually becomes easier, since soundness of knowledge and assumptions is enforced for one and the same vocabulary (the one from the original theory). The witness formulas become somewhat more complicated, as the witnesses have to respect both the knowledge as well as the assumptions of the theory. This is best explained by consulting our running example again.

Example 1 (Continued) *While F's propositionalization $tr_p(\{F\})$ stays the same, the soundness and witness formulas change in the step from formula $\Phi_{\{F\}}$ to formula $\Psi_{\{F\}}$. We only show the first conjunct of the witness formula Ψ_{wit}, which is given by*

$$\neg k_p \supset \left(\neg p^{k_p} \wedge \left(k_p \supset p^{k_p}\right) \wedge \left(a_{\neg p} \supset \neg p^{k_p}\right)\right)$$

Intuitively, the formula expresses that whenever p is not known, then there must be a witness, that is, an interpretation where p is false. Since the witnessing interpretations could in principle be distinct for each $\mathbf{K}$-atom, they have to be indexed by the respective $\mathbf{K}$-atom they refer to, as in p^{k_p}. Of course, the witnesses have to obey all that is known and assumed, which is guaranteed in the last two conjuncts.

Using this new formula, the result of Proposition 2 can be restated.

Proposition 3 *Let T be a pure GK theory. A Kripke interpretation M is a GK model of T if and only if there exists a model I^* of the propositional formula Ψ_T such that*

- $\mathbf{K}(M) = \mathbf{A}(M) = Th\left(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T), I^* \models k_\phi\}\right)$;
- *for each $\psi \in Atom_{\mathbf{A}}(T)$, we have that $I^* \models a_\psi$ implies*

$$\psi \in Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$$

- *there does not exist another model $I^{*\prime}$ of Φ_T such that*

$$I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}$$
$$I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}$$

We are now ready for our main result, translating a pure GK theory to a disjunctive logic program. First, we introduce some notations. Let T be a pure GK theory, we use $tr_{ne}(T)$ to denote the nested expression obtained from Ψ_T by first converting it to negation normal form[2], then replacing "$\wedge$" by "," and "$\vee$" by ";". A propositional formula ϕ can be equivalently translated to conjunctive normal form (involving at most linear blowup)

$$(p_1 \vee \cdots \vee p_t \vee \neg p_{t+1} \vee \cdots \vee \neg p_m) \wedge \ldots \\ \wedge (q_1 \vee \cdots \vee q_k \vee \neg q_{k+1} \vee \cdots \vee \neg q_n)$$

where p's and q's are atoms; we use $tr_c(\phi)$ to denote the set of rules

$$p_1; \ldots; p_t \leftarrow p_{t+1}, \ldots, p_m \quad \ldots \quad q_1; \ldots; q_k \leftarrow q_{k+1}, \ldots, q_n$$

We use $\widehat{\phi}$ to denote the propositional formula obtained from ϕ by replacing each occurrence of an atom p by a new atom $\hat{p}$.

[2]A propositional formula is in Negation Normal Form (NNF) if negation occurs only immediately above atoms, and $\{\bot, \top, \neg, \wedge, \vee\}$ are the only allowed connectives.

We use T^* to denote the propositional formula obtained from the formula Φ_T by replacing each occurrence of an atom p (except atoms in $\{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}$) by a new atom p^*. Intuitively, each atom that is not an a-atom is replaced by a new atom.

Notice that $tr_{ne}(T)$ is obtained from Ψ_T while T^* is obtained from Φ_T. Intuitively, by Proposition 3, $tr_{ne}(T)$ is used to restrict interpretations for introduced k-atoms and a-atoms so that these interpretations serve as candidates for GK models, and by Proposition 1, T^* constructs possible models of the GK theory which are later used to test whether these models prevent the candidate to be a GK model.

Inspired by the linear translation from parallel circumscription into disjunctive logic programs by Janhunen and Oikarinen [2004], we have the following theorem.

Theorem 1 *Let T be a pure GK theory. A Kripke interpretation M is a GK model of T if and only if there exists an answer set S of the logic program $tr_{lp}(T)$ in Figure 1 with $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } k_\phi \in S\})$.*

The intuition behind the construction is as follows:

- (1) and (2) in $tr_{lp}(T)$: I^* is a model of the formula Ψ_T.
- (3–8): if there exists a model $I^{*\prime}$ of the formula Φ_T with

$$I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}$$
$$I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\},$$

  then there exists a set S^* constructed from new atoms in $tr_c(T^*)$ (which is a copy of the formula Φ_T with same a_ϕ for each $\phi \in Atom_{\mathbf{A}}(T)$) and c_ϕ for some $\phi \in Atom_{\mathbf{K}}(T)$ such that S^* satisfies rules (3) to (8) and $u \notin S^*$.
- (9) and (10): if there is such a set S^* then it is the least set containing u, all p^*'s and c-atoms.
- (11): such a set S^* should not exist. (See item 3 in Proposition 3.)
- (12) and (13): if there exists a model of the formula $\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)}(k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)}(a_\phi \supset \widehat{\phi})$, then v should not occur in the minimal model of the program.
- (14): $\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)}(k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)}(a_\phi \supset \widehat{\phi})$ should not be consistent. (This is necessary by item 2 in Proposition 3.)

Given a model S of the logic program $tr_{lp}(T)$, the new atom u is used to indicate that the model I^* of Ψ_T w.r.t. S (specified by (1) and (2)) satisfies item 3 in Proposition 3. Specifically, if I^* does not satisfy item 3, then there exists a subset S^* of p^*'s and c-atoms that satisfies (3) to (8). If in addition $u \notin S^*$, then there exists a subset of S that satisfies all rules in $tr_{lp}(T)$ except (11), thus S cannot be an answer set of $tr_{lp}(T)$. Similarly, v is used to indicate that I^* satisfies item 2 in Proposition 3. Specifically, if I^* does not satisfy item 2, then the propositional formula $\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)}(k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)}(a_\phi \supset \widehat{\phi})$ is satisfiable, thus there exists a subset $\widehat{S}$ of $\hat{p}$'s that satisfies (12). If in addition $v \notin \widehat{S}$, then there exists a subset of S that satisfies all rules in $tr_{lp}(T)$ except (14), thus S cannot be an answer set of $tr_{lp}(T)$.

$$
\begin{array}{lrl}
(1) & \bot \leftarrow \mathit{not}\ tr_{ne}(T) & \\
(2) & p'; \neg p' \leftarrow \top & \textit{(for each atom } p' \textit{ occurring in } tr_{ne}(T)\textit{)} \\
(3) & u; A \leftarrow B & \textit{(for each rule } A \leftarrow B \textit{ in } tr_c(T^*)\textit{)} \\
(4) & u; c_{\phi_1}; \cdots; c_{\phi_m} \leftarrow \top & (\{\phi_1, \ldots, \phi_m\} = Atom_{\mathbf{K}}(T)) \\
(5) & u \leftarrow c_\phi, \mathit{not}\ k_\phi & \textit{(for each } \phi \in Atom_{\mathbf{K}}(T)\textit{)} \\
(6) & u \leftarrow k^*_\phi, \mathit{not}\ k_\phi & \textit{(for each } \phi \in Atom_{\mathbf{K}}(T)\textit{)} \\
(7) & u \leftarrow c_\phi, k^*_\phi, \mathit{not}\ \neg k_\phi & \textit{(for each } \phi \in Atom_{\mathbf{K}}(T)\textit{)} \\
(8) & u; c_\phi; k^*_\phi \leftarrow \mathit{not}\ \neg k_\phi & \textit{(for each } \phi \in Atom_{\mathbf{K}}(T)\textit{)} \\
(9) & p^* \leftarrow u & \textit{(for each new atom } p^* \textit{ occurring in } tr_c(T^*)\textit{)} \\
(10) & c_\phi \leftarrow u & \textit{(for each } \phi \in Atom_{\mathbf{K}}(T)\textit{)} \\
(11) & \bot \leftarrow \mathit{not}\ u & \\
(12) & v; A \leftarrow B & \textit{(for each rule } A \leftarrow B \textit{ in } tr_c\left(\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \widehat{\phi})\right)\textit{)} \\
(13) & \hat{p} \leftarrow v & \textit{(for each atom } \hat{p} \textit{ except } k\textit{-atoms and } a\textit{-atoms occurring in } tr_c\left(\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)} (k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\phi \in Atom_{\mathbf{A}}(T)} (a_\phi \supset \widehat{\phi})\right)\textit{)} \\
(14) & \bot \leftarrow \mathit{not}\ v &
\end{array}
$$

Figure 1: Translation from pure GK theory T to disjunctive logic program $tr_{lp}(T)$ used in Theorem 1, where u, v, and c_ϕ (for each $\phi \in Atom_{\mathbf{K}}(T)$) are new atoms.

Example 1 (Continued) *For our running example theory $\{F\}$ with $F = \neg\mathbf{A}\neg p \supset \mathbf{K}p$, we find that the logic program translation $tr_{lp}(\{F\})$ has a single answer set S with $k_p \in S$ Thus by Theorem 1 we can conclude that the GK theory $\{F\}$ has a single GK model M in which $\mathbf{K}(M) = Th(\{p\})$. Likewise, the logic program $tr_{lp}(\{F, G\})$ has a single answer set S' with $k_{\neg p} \in S'$, whence $\{F, G\}$ has a single GK model M' in which $\mathbf{K}(M') = Th(\{\neg p\})$.*

Computational complexity We have seen in the preliminaries section that disjunctive logic programs can be modularly and equivalently translated into pure formulas of the logic of GK. Conversely, Theorem 1 shows that pure GK formulas can be equivalently translated into disjunctive logic programs. Eiter and Gottlob showed that the problem of deciding whether a disjunctive logic program has an answer set is Σ^P_2-complete [Eiter and Gottlob, 1995]. In combination, these results yield the following straightforward complexity result for the satisfiability of pure GK.

Proposition 4 *Let T be a pure GK theory. The problem of deciding whether T has a GK model is Σ^P_2-complete.*

We remark that the hardness of disjunctive logic programs stems from so-called head cycles (at least two atoms that mutually depend on each other and occur jointly in some rule head). It is straightforwardly checked that our encoding creates such head cycles, for example the head of rule (8) contains the cycle induced by rules (7) and (10).

Implementation

We have implemented the translation of Theorem 1 into a working prototype gk2dlp. The program is written in Prolog and uses the disjunctive ASP solver claspD-2 [Gebser, Kaufmann, and Schaub, 2013], which was ranked first place in the 2013 ASP competition.[3]

Our prototype is the first implementation of the (pure) logic of GK to date. The restriction to pure formulas seems harmless since all known applications of the logic of GK use only pure formulas. We remark that gk2dlp implements default and autoepistemic logics such that input and target language are of the same complexity.

Evaluation To have a scalable problem domain and inspired by dl2asp [Chen et al., 2010], we chose the fair division problem [Bouveret and Lang, 2008] for experimental evaluation. An instance of the fair division problem consists of a set of agents, a set of goods, and for each agent a set of constraints that intuitively express which sets of goods the agent is willing to accept. A solution is then an assignment of goods to agents that is a partition of all goods and satisfies all agents' constraints. Bouveret and Lang [2008] showed that the problem is Σ^P_2-complete, and can be naturally encoded in default logic.

[3] `http://www.mat.unical.it/ianni/storage/aspcomp-2013-lpnmrtalk.pdf`

We created random instances of the fair division problem with increasing numbers of agents and goods. We then applied the translation of [Bouveret and Lang, 2008], furthermore the translation from default logic into the logic of GK, then invoked gk2dlp to produce logic programs and finally used gringo 3.0.3 and claspD version 2 (revision 6814) to compute all answer sets of these programs, thus all extensions of the original default theory corresponding to all solutions of the problem instance. The experiments were conducted on a Lenovo laptop with an Intel Core i3 processor with 4 cores and 4GB of RAM running Ubuntu 12.04. We recorded the size of the default theory, the size of the translated logic program, the translation time and the solving time, as well as the number of solutions obtained. We started out with 2 agents and 2 goods, and stepwise increased these numbers towards 6. For each combination in $(a, g) \in \{2, \ldots, 6\} \times \{2, \ldots, 6\}$, we tested 20 randomly generated instances. Random generation here means that we create agents' preferences by iteratively drawing random subsets of goods to add to an agent's acceptable subsets with probability P, where P is initialized with 1 and discounted by the factor $\frac{g-1}{g}$ for each subset that has been drawn.

In accordance with our theoretical predictions, we observed that the increase in size from GK formula to logic program is indeed polynomial (albeit with a low exponent). The plot on the right (Figure 2) shows the solving time in relation to the size of the default theory, where the time axis is logarithmic. We can see that the runtime behavior of gk2dlp is satisfactory. We acknowledge however that the runtimes we measured are not competitive with those reported by Chen et al. [2010] for dl2asp. However, a direct comparison of the two systems is problematic for a number of reasons. First of all, the system dl2asp is not publicly available to the best of our knowledge. Furthermore, Chen et al. [2010] do not describe how they create random instances of the fair division problem, so we cannot compare the runtimes they report and the ones we measured. Finally, dl2asp is especially engineered for default logic, and it is not clear how their approach can be generalized to other languages, for example Turner's logic of universal causation. In general, the approaches to translation that are followed by dl2asp and gk2dlp are completely different: dl2asp translates a Σ_2^P-complete problem to an NP-complete problem using a translation in Δ_2^P. Our system gk2dlp translates a Σ_2^P-complete problem into another Σ_2^P-complete problem using a translation that can be computed in polynomial time.

Applications We see immediate applicability of the translation of the present paper to several areas. Reiter [1987] provided a theory of diagnosis from first principles, and showed how default logic can be used as an implementation device. Cadoli, Eiter, and Gottlob [1994] proposed to use default logic as an expressive query language on top of relational databases, and gave an example of achieving strategic behavior in an economic setting. In reasoning about actions, Thielscher [1996] used default logic to solve the qualification problem of dealing with unexpected action failures. Martin and Thielscher [2001] later provided an implementation of that approach where extensions are enumerated in Prolog. Recently, Baumann et al. [2010] introduced a method for default reasoning in action theories, that is, an approach to the question what normally holds in a dynamic domain. Our translation yields an implementation of their approach, something that they stated as future work and later achieved to a limited extent (for a restricted sublanguage of their framework [Strass, 2012]). In a similar vein, Pagnucco et al. [2013] looked at belief change in the situation calculus and proposed an implementation based on default logic with preferences [Brewka, 1994; Delgrande and Schaub, 2000].

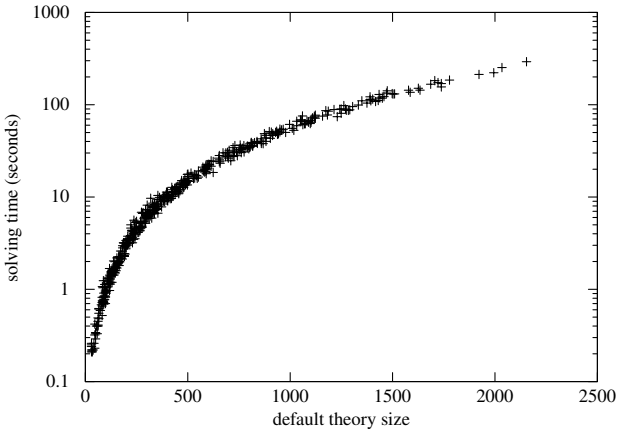


Figure 2: Solving time (log scale) with respect to default theory size.

Related work The translation presented in this paper is a generalization of the one presented for Turner's logic of universal causation by Ji and Lin [2013]. We chose the logic of GK as general nonmonotonic language, we could also have chosen the logic of minimal belief and negation as failure [Lifschitz, 1994], the logic of here-and-there [Heyting, 1930] or the nonmonotonic modal logic S4F [Schwarz and Truszczynski, 1994]. In terms of implementations, there are few approaches that treat as broad a range of propositional nonmonotonic knowledge representation languages as gk2dlp. Notable exceptions are the works of Junker and Konolige [1990], who implemented both autoepistemic and default logics by translating them to truth maintenance systems; Niemelä [1995], who provides a decision procedure for autoepistemic logic which also incorporates extension semantics for default logics; and Rosati [1999], who provides algorithms for Lifschitz' logic of minimal belief and negation as failure [1994]. Other approaches are restricted to specific languages, where default logic seems to be most popular. The recent system dl2asp [Chen et al., 2010] translates default theories to normal (non-disjunctive) logic programs; the translation figures out all implication relations between formulas occurring in the default theory, just as Junker and Konolige [1990] did. The authors of dl2asp [Chen et al., 2010] already observed that default logic and disjunctive logic programs are of the same complexity; they even stated the search for a polynomial translation from the former to the latter (that we achieved in this paper) as future work. Gadel [Nicolas, Saubion, and Stéphan,

2000] uses a genetic algorithm to compute extensions of a default theory; likewise the system DeReS [Cholewiński et al., 1999] is not translation-based but directly searches for extensions; similarly the XRay system [Schaub and Nicolas, 1997] implements local query-answering in default logics. Risch and Schwind [1994] describe a tableaux-based algorithm for computing all extensions of general default theories, but do not report runtimes for their Prolog-based implementation. For autoepistemic logic, Marek and Truszczyński [1991] investigate sceptical reasoning with respect to Moore's expansion semantics.

Discussion

We have presented the first translation of pure formulas of the logic of GK to disjunctive answer set programming. Among other things, this directly leads to implementations of Turner's logic of universal causation as well as implementations of default and autoepistemic logics under different semantics. We have prototypically implemented the translation and experimentally analysed its performance, which we found to be satisfactory given the system's generality.

In the future, we plan to integrate further nonmonotonic reasoning formalisms. This is more or less straightforward due to the generality of this work: to implement a language, it suffices to provide a translation into pure formulas of GK, then Theorem 1 of this paper does the rest. Particular formalism we want to look at are default logics with preferences [Brewka, 1994; Delgrande and Schaub, 2000] and the logic of only-knowing [Lakemeyer and Levesque, 2005]. It also seems worthwhile to check whether our translation can be adapted to the nonmonotonic modal logic S4F [Schwarz and Truszczynski, 1994; Truszczyński, 2007], that has only one modality instead of two. We finally plan to study the approaches mentioned as applications in the previous section to try out our translation and implementation on agent-oriented AI problems.

References

Baumann, R.; Brewka, G.; Strass, H.; Thielscher, M.; and Zaslawski, V. 2010. State Defaults and Ramifications in the Unifying Action Calculus. In *KR*, 435–444.

Bouveret, S., and Lang, J. 2008. Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. *JAIR* 32:525–564.

Brewka, G. 1994. Adding Priorities and Specificity to Default Logic. In *JELIA*, 247–260.

Cadoli, M.; Eiter, T.; and Gottlob, G. 1994. Default logic as a query language. In *KR*, 99–108.

Chen, Y.; Wan, H.; Zhang, Y.; and Zhou, Y. 2010. dl2asp: Implementing Default Logic via Answer Set Programming. In *JELIA*, volume 6341, 104–116.

Cholewiński, P.; Marek, V. W.; Truszczyński, M.; and Mikitiuk, A. 1999. Computing with default logic. *AIJ* 112(1):105–146.

Delgrande, J. P., and Schaub, T. 2000. Expressing Preferences in Default Logic. *AIJ* 123(1–2):41–87.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2003. Uniform Semantic Treatment of Default and Autoepistemic Logics. *AIJ* 143(1):79–122.

Dix, J.; Furbach, U.; and Niemelä, I. 2001. Nonmonotonic reasoning: Towards efficient calculi and implementations. *Handbook of Automated Reasoning* 2(18):1121–1234.

Drescher, C.; Gebser, M.; Grote, T.; Kaufmann, B.; König, A.; Ostrowski, M.; and Schaub, T. 2008. Conflict-Driven Disjunctive Answer Set Solving. In *KR*, 422–432.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *AMAI* 15(3–4):289–323.

Ferraris, P. 2005. Answer sets for propositional theories. In *LPNMR*, 119–131.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2013. Advanced conflict-driven disjunctive answer set solving. In *IJCAI*.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer Set Programming Based on Propositional Satisfiability. *J. Autom. Reasoning* 36(4):345–377.

Heyting, A. 1930. Die formalen Regeln der intuitionistischen Logik. In *Sitzungsberichte der preußischen Akademie der Wissenschaften*, 42–65, 57–71, 158–169. Physikalisch-mathematische Klasse.

Janhunen, T., and Niemelä, I. 2004. GnT – A Solver for Disjunctive Logic Programs. In *LPNMR*, 331–335.

Janhunen, T., and Oikarinen, E. 2004. Capturing parallel circumscription with disjunctive logic programs. In *Logics in Artificial Intelligence*. 134–146.

Ji, J., and Lin, F. 2012. From Turner's Logic of Universal Causation to the Logic of GK. In *Correct Reasoning*, volume 7265, 380–385.

Ji, J., and Lin, F. 2013. Turner's logic of universal causation, propositional logic, and logic programming. In *LPNMR*, 401–413.

Junker, U., and Konolige, K. 1990. Computing the Extensions of Autoepistemic and Default Logics with a Truth Maintenance System. In *AAAI*, 278–283.

Konolige, K. 1988. On the Relation Between Default and Autoepistemic Logic. *AIJ* 35(3):343–382.

Lakemeyer, G., and Levesque, H. J. 2005. Only-knowing: Taking it beyond autoepistemic reasoning. In *AAAI*, 633–638.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7(3):499–562.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *AMAI* 25(3-4):369–389.

Lifschitz, V. 1994. Minimal belief and negation as failure. *AIJ* 70(1–2):53–72.

Lin, F., and Shoham, Y. 1992. A logic of knowledge and justified assumptions. *AIJ* 57(2-3):271–289.

Lin, F., and Zhou, Y. 2011. From answer set logic programming to circumscription via logic of GK. *AIJ* 175(1):264–277.

Lin, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *KR*, 170–176.

Marek, V. W., and Truszczyński, M. 1991. Computing intersection of autoepistemic expansions. In *LPNMR*, 37–50.

Martin, Y., and Thielscher, M. 2001. Addressing the Qualification Problem in FLUX. In *KI/ÖGAI*, 290–304.

McCarthy, J. 1980. Circumscription – a form of non-monotonic reasoning. *AIJ* 13:295–323.

McCarthy, J. 1986. Applications of circumscription to formalizing commonsense knowledge. *AIJ* 28:89–118.

Moore, R. 1985. Semantical considerations on nonmonotonic logic. *AIJ* 25(1):75–94.

Nicolas, P.; Saubion, F.; and Stéphan, I. 2000. Gadel: a genetic algorithm to compute default logic extensions. In *ECAI*, 484–490.

Niemelä, I. 1995. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. *J. Autom. Reasoning* 14(1):3–42.

Pagnucco, M.; Rajaratnam, D.; Strass, H.; and Thielscher, M. 2013. Implementing Belief Change in the Situation Calculus and an Application. In *LPNMR*, volume 8148, 439–451.

Reiter, R. 1980. A logic for default reasoning. *AIJ* 13(1-2):81–132.

Reiter, R. 1987. A theory of diagnosis from first principles. *AIJ* 32(1):57–95.

Risch, V., and Schwind, C. 1994. Tableaux-based characterization and theorem proving for default logic. *J. Autom. Reasoning* 13(2):223–242.

Rosati, R. 1999. Reasoning about minimal belief and negation as failure. *JAIR* 11:277–300.

Schaub, T., and Nicolas, P. 1997. An implementation platform for query-answering in default logics: The XRay system, its implementation and evaluation. In *LPNMR*. 441–452.

Schwarz, G., and Truszczynski, M. 1994. Minimal knowledge problem: A new approach. *AIJ* 67(1):113–141.

Strass, H. 2012. The draculasp system: Default reasoning about actions and change using logic and answer set programming. In *NMR*.

Thielscher, M. 1996. Causality and the Qualification Problem. In *KR*, 51–62.

Truszczyński, M. 2007. The modal logic S4F, the default logic, and the logic here-and-there. In *AAAI*, 508–514.

Turner, H. 1999. Logic of universal causation. *AIJ* 113(1):87–123.

Appendix

Proof of Proposition 1:

$\Rightarrow$: Let M be a model of T, $I_1 \subseteq Lit$ a model of $\mathbf{K}(M)$, and $I_2 \subseteq Lit$ a model of $\mathbf{A}(M)$. Clearly, for each $\phi \in Atom_{\mathbf{K}}(T)$, if $\phi \in \mathbf{K}(M)$ then $I_1 \models \phi$; if $\phi \notin \mathbf{K}(M)$ then there exists a model I' of $\mathbf{K}(M)$ such that $I' \models \neg\phi$. Same results are established for each $\phi \in Atom_{\mathbf{A}}(T)$.

Then, we can create an interpretation I^* such that

$$
\begin{aligned}
I^* = &\{l^k \mid l \in I_1\} \cup \{l^a \mid l \in I_2\} \\
&\cup \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \cap \mathbf{K}(M)\} \\
&\cup \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \cap \mathbf{A}(M)\} \\
&\cup \{\neg k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } \phi \notin \mathbf{K}(M)\} \\
&\cup \{\neg a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } \phi \notin \mathbf{A}(M)\} \\
\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{K}}(T) \\ \psi \in \mathbf{K}(M)}} &\{l^{k_\psi} \mid l \in I_1\} \cup \bigcup_{\substack{\psi \in Atom_{\mathbf{A}}(T) \\ \psi \in \mathbf{A}(M)}} \{l^{a_\psi} \mid l \in I_2\} \\
\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{K}}(T) \\ \psi \notin \mathbf{K}(M)}} &\{l^{k_\psi} \mid l \in I', I' \text{ is a model of } \mathbf{K}(M) \cup \{\neg\psi\}\} \\
\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{A}}(T) \\ \psi \notin \mathbf{A}(M)}} &\{l^{a_\psi} \mid l \in I', I' \text{ is a model of } \mathbf{A}(M) \cup \{\neg\psi\}\}.
\end{aligned}
$$

It is easy to verify that I^* is a model of Φ_T and

- $\mathbf{K}(M) \cap Atom_{\mathbf{K}}(T) = \{\phi \mid \phi \in Atom_{\mathbf{K}}(T),\ I^* \models k_\phi\}$;
- $\mathbf{A}(M) \cap Atom_{\mathbf{A}}(T) = \{\phi \mid \phi \in Atom_{\mathbf{A}}(T),\ I^* \models a_\phi\}$.

$\Leftarrow$: Let I^* be a model of Φ_T. We can create a Kripke interpretation M such that

- $\mathbf{K}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$;
- $\mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } I^* \models a_\phi\})$.

Note that, $\{l \in Lit \mid I^* \models l^k\}$ is a model of $\mathbf{K}(M)$ and $\{l \in Lit \mid I^* \models l^a\}$ is a model of $\mathbf{A}(M)$, then both $\mathbf{K}(M)$ and $\mathbf{A}(M)$ are consistent.

For each $\phi \in Atom_{\mathbf{K}}(T)$, if $I^* \models k_\phi$ then $\phi \in \mathbf{K}(M)$; if $I^* \models \neg k_\phi$ then there exists a model $I' = \{l \in Lit \mid I^* \models l^{k_\phi}\}$ such that I' is a model of $\mathbf{K}(M)$ and $I' \models \neg\phi$, thus $\phi \notin \mathbf{K}(M)$. So $I^* \models k_\phi$ iff $\phi \in \mathbf{K}(M)$. The same result is established for each $\phi \in Atom_{\mathbf{A}}(T)$. Note that, $I^* \models tr_p(T)$ then M is a model of T. ■

Proof of Proposition 2:

$\Rightarrow$: Let M be a GK model of T. From the proof of Proposition 1, we can create a model I^* of Φ_T. Now we want to prove that I^* satisfies all conditions in the proposition.

From Theorem 3.5 in [Lin and Shoham, 1992], $\mathbf{K}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \cap \mathbf{K}(M)\})$, then $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$.

Assume that there exists another model $I^{*\prime}$ of Φ_T with

$$I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}$$
$$I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}$$

Then, from Proposition 1, there exists a Kripke interpretation M' such that $\mathbf{K}(M') = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^{*\prime} \models k_\phi\})$, $\mathbf{A}(M') = \mathbf{A}(M)$, and M' is a model of T. Note that, for each $\phi \in Atom_{\mathbf{K}}(T)$, $I^{*\prime} \models \neg k_\phi$ implies $\mathbf{K}(M') \not\models \phi$, then $\mathbf{K}(M') \subsetneq \mathbf{K}(M)$. From the definition of GK models, there does not exist such a model M', which conflicts to the assumption, then there does not exist such a model $I^{*\prime}$.

From the construction of I^*, for each $\psi \in Atom_{\mathbf{A}}(T)$, $I^* \models a_\psi$ iff $\psi \in \mathbf{A}(M)$. Note that, $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$, then $I^* \models a_\psi$ iff $\psi \in Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$.

So I^* is a model of Φ_T which satisfies all conditions in the proposition.

$\Leftarrow$: Let I^* be a model of Φ_T which satisfies corresponding conditions in the proposition. We can create a Kripke interpretation M such that $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$.

From the third condition in the proposition, $I^* \models a_\phi$ iff $\phi \in \mathbf{K}(M)$ for each $\phi \in Atom_{\mathbf{A}}(T)$. Then $\mathbf{A}(M) \cap Atom_{\mathbf{A}}(T) = \{\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } I^* \models a_\phi\}$. From the proof of Proposition 1, M is a model of T and $I^* \models k_\phi$ (resp. $I^* \models a_\phi$) iff $\phi \in \mathbf{K}(M)$ for each $\phi \in Atom_{\mathbf{K}}(T)$ (resp. $\phi \in Atom_{\mathbf{A}}(T)$). Now we want to prove that M is a GK model of T.

Assume that there exists another model M' of T such that $\mathbf{A}(M') = \mathbf{A}(M)$ and $\mathbf{K}(M') \subsetneq \mathbf{K}(M)$. Note that $\mathbf{K}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$, then $\mathbf{K}(M') \cap Atom_{\mathbf{K}}(T) \subsetneq \mathbf{K}(M) \cap Atom_{\mathbf{K}}(T)$.

Let $I = I^* \cap \{l^k \mid l \in Lit\}$, clearly, I is a model of $\mathbf{K}(M)$, $\mathbf{A}(M)$, and $\mathbf{K}(M')$. We can construct another model $I^{*\prime}$ of Φ_T as

$$\begin{aligned}
I^{*\prime} = &\{l^k \mid l \in I\} \cup \{l^a \mid l \in I\} \\
&\cup \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \cap \mathbf{K}(M')\} \\
&\cup \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \cap \mathbf{A}(M)\} \\
&\cup \{\neg k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } \phi \notin \mathbf{K}(M')\} \\
&\cup \{\neg a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } \phi \notin \mathbf{A}(M)\} \\
&\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{K}}(T) \\ \psi \in \mathbf{K}(M')}} \{l^{k_\psi} \mid l \in I\} \\
&\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{A}}(T) \\ \psi \in \mathbf{A}(M)}} \{l^{a_\psi} \mid l \in I\} \\
&\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{K}}(T) \\ \psi \notin \mathbf{K}(M')}} \{l^{k_\psi} \mid l \in I', I' \text{ is a model of } \\
&\qquad\qquad \mathbf{K}(M') \cup \{\neg\psi\}\} \\
&\cup \bigcup_{\substack{\psi \in Atom_{\mathbf{A}}(T) \\ \psi \notin \mathbf{A}(M)}} \{l^{a_\psi} \mid l \in I', I' \text{ is a model of } \\
&\qquad\qquad \mathbf{A}(M) \cup \{\neg\psi\}\}.
\end{aligned}$$

From the proof of Proposition 1, $I^{*\prime}$ is a model of Φ_T, and

$$\begin{aligned}
&I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} \\
&I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}.
\end{aligned}$$

This conflicts to the second condition in the proposition, then the assumption is not valid. So there does not exist another model M' of T such that $\mathbf{A}(M') = \mathbf{A}(M)$ and $\mathbf{K}(M') \subsetneq \mathbf{K}(M)$, thus M is a GK model of T. ■

Proof of Theorem 1:

$\Rightarrow$: Let M be a GK model of T. From Proposition 3, there exists a model I^* of Ψ_T such that $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$. We can create a set S of literals as $S = I^* \cup \{u, v\} \cup$

$\{p^* \mid \text{for each new atom } p^* \text{ occurring in } tr_c(T^*)\} \cup$
$\{c_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \cup \{\hat{p} \mid p \in Atom\}$.

Clearly, S satisfies each rule in $tr_{lp}(T)$. Now we want to prove that S is an answer set of the program.

Assume that S is not an answer set of $tr_{lp}(T)$, then there exists another set $S' \subsetneq S$ such that S' satisfies each rule in the reduct $tr_{lp}(T)^S$. Note that, $I^* \subseteq S'$, u implies $\{p^* \mid \text{for each new atom } p^* \text{ occurring in } tr_c(T^*)\} \cup \{c_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}$ and v implies $\{\hat{p} \mid p \in Atom\}$. Then there are only two possible cases: $u \notin S'$ or $v \notin S'$.

Case 1: $u \notin S'$, then there exists a set

$$T = S' \cap \big(\{p^* \mid p^* \text{ is a new atom occurring in } tr_c(T^*)\} \cup \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}\big)$$

such that T satisfies $tr_c(T^*)$. For each $\phi \in Atom_{\mathbf{K}}(T)$,

- by the rule $u \leftarrow c_\phi, not\ k_\phi$, $I^* \models \neg k_\phi$ implies $c_\phi \notin S'$;
- by the rule $u \leftarrow k^*_\phi, not\ k_\phi$, $I^* \models \neg k_\phi$ implies $k^*_\phi \notin S'$;
- by rules $u \leftarrow c_\phi, k^*_\phi, not\ \neg k_\phi$ and $u; c_\phi; k^*_\phi \leftarrow not\ \neg k_\phi$, $I^* \models k_\phi$ implies either c_ϕ or k^*_ϕ is in S' but not both;
- by the rule $u; c_{\phi_1}; \cdots; c_{\phi_m} \leftarrow \top$, there exists $c_\psi \in S'$ for some $\psi \in Atom_{\mathbf{K}}(T)$.

So there exists $\psi \in Atom_{\mathbf{K}}(T)$ such that $k_\psi \in S'$, $c_\psi \in S'$ and $k^*_\psi \notin S'$. Then we could create an interpretation $I^{*\prime}$ as

$$\begin{aligned}
I^{*\prime} = &\{p \mid p \in Atom \text{ and } p^* \in S'\} \\
&\cup \{\neg p \mid p \in Atom \text{ and } p^* \notin S'\} \\
&\cup \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } k^*_\phi \in S'\} \\
&\cup \{\neg k_\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } k^*_\phi \notin S'\} \\
&\cup \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } a_\phi \in S'\} \\
&\cup \{\neg a_\phi \mid \phi \in Atom_{\mathbf{A}}(T) \text{ and } a_\phi \notin S'\} \\
&\cup \bigcup_{\psi \in Atom_{\mathbf{K}}(T)} \{p^{k_\psi} \mid p^{k_\psi *} \in S'\} \\
&\cup \bigcup_{\psi \in Atom_{\mathbf{K}}(T)} \{\neg p^{k_\psi} \mid p^{k_\psi *} \notin S'\} \\
&\cup \bigcup_{\psi \in Atom_{\mathbf{A}}(T)} \{p^{a_\psi} \mid p^{a_\psi *} \in S'\} \\
&\cup \bigcup_{\psi \in Atom_{\mathbf{A}}(T)} \{\neg p^{a_\psi} \mid p^{a_\psi *} \notin S'\}.
\end{aligned}$$

Clearly, $I^{*\prime}$ is a model of Ψ_T. From the above results,

- $I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}$, and
- $I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}$.

From Proposition 3, such $I^{*\prime}$ does not exist. This conflicts to the assumption, then Case 1 is impossible.

Case 2: $v \notin S'$, then there exists a set

$$U = S' \cap \big(\{\hat{a} \mid a \in Atom\} \cup \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \cup \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\}\big)$$

such that U satisfies each rule in $tr_c(\bigwedge_{\phi \in Atom_{\mathbf{K}}(T)}(k_\phi \supset \widehat{\phi}) \wedge \neg \bigwedge_{\psi \in Atom_{\mathbf{A}}(T)}(a_\psi \supset \widehat{\psi}))$.

Then there exists $\psi \in Atom_{\mathbf{A}}(T)$ such that $I^* \models a_\psi$ and there exists an interpretation $I \subseteq Lit$ such that $I \models \bigwedge_{\phi \in Atom_{\mathbf{K}}(T), I^* \models k_\phi} \phi \wedge \neg\psi$, thus $\psi \notin Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T) \text{ and } I^* \models k_\phi\})$. From Proposition 3, such ψ does not exist. This conflicts to the assumption, then Case 2 is impossible. So both cases are impossible, then S' does not exist and S is an answer set of $tr_{lp}(T)$.

$\Leftarrow$: Let S be an answer set of $tr_{lp}(T)$. We can create an interpretation I^* as the intersection of S with the set of atoms occurring in Ψ_T. Clearly, I^* is a model of Ψ_T.

Similar to the above proof: If there exists another model $I^{*\prime}$ of Ψ_T such that

$$\begin{aligned}
&I^{*\prime} \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} = I^* \cap \{a_\phi \mid \phi \in Atom_{\mathbf{A}}(T)\} \\
&I^{*\prime} \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\} \subsetneq I^* \cap \{k_\phi \mid \phi \in Atom_{\mathbf{K}}(T)\}
\end{aligned}$$

then there exists another set S' such that S' satisfies each rule in the reduct $tr_{lp}(T)^S$ and $u \notin S'$, thus $S' \subsetneq S$. This conflicts to the precondition that S is an answer set, then such a model $I^{*\prime}$ does not exist.

If there exists $\psi \in Atom_{\mathbf{A}}(T)$ such that $I^* \models a_\psi$ and $\psi \notin Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T)$ and $I^* \models k_\phi\})$, then there exists another set S' such that S' satisfies each rule in the reduct $tr_{lp}(T)^S$ and $v \notin S'$, thus $S' \subsetneq S$. This conflicts to the precondition that S is an answer set, then such ψ does not exist.

From Proposition 3, a Kripke interpretation M such that $\mathbf{K}(M) = \mathbf{A}(M) = Th(\{\phi \mid \phi \in Atom_{\mathbf{K}}(T)$ and $k_\phi \in S\})$ is a GK models of T. ■

Compact Argumentation Frameworks*

Ringo Baumann and **Hannes Strass**
Leipzig University, Germany

Wolfgang Dvořák
University of Vienna, Austria

Thomas Linsbichler and **Stefan Woltran**
Vienna University of Technology, Austria

Abstract

Abstract argumentation frameworks (AFs) are one of the most studied formalisms in AI. In this work, we introduce a certain subclass of AFs which we call compact. Given an extension-based semantics, the corresponding compact AFs are characterized by the feature that each argument of the AF occurs in at least one extension. This not only guarantees a certain notion of fairness; compact AFs are thus also minimal in the sense that no argument can be removed without changing the outcome. We address the following questions in the paper: (1) How are the classes of compact AFs related for different semantics? (2) Under which circumstances can AFs be transformed into equivalent compact ones? (3) Finally, we show that compact AFs are indeed a non-trivial subclass, since the verification problem remains coNP-hard for certain semantics.

1 Introduction

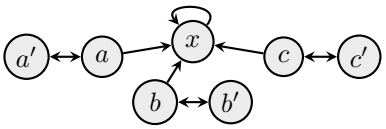
In recent years, *argumentation* has become a major concept in AI research (Bench-Capon & Dunne 2007; Rahwan & Simari 2009). In particular, Dung's well-studied *abstract argumentation frameworks* (AFs) (Dung 1995) are a simple, yet powerful formalism for modeling and deciding argumentation problems. Over the years, various *semantics* have been proposed, which may yield different results (so called *extensions*) when evaluating an AF (Dung 1995; Verheij 1996; Caminada, Carnielli, & Dunne 2012; Baroni, Caminada, & Giacomin 2011). Also, some subclasses of AFs such as acyclic, symmetric, odd-cycle-free or bipartite AFs, have been considered, where for some of these classes different semantics collapse (Coste-Marquis, Devred, & Marquis 2005; Dunne 2007).

In this work we introduce a further class, which to the best of our knowledge has not received attention in the literature, albeit the idea is simple. We will call an AF *compact* (with respect to a semantics σ), if each of its arguments appears in at least one extension under σ. Thus, compact AFs yield a "semantic" subclass since its definition is based on the notion of extensions. Another example of such a semantic subclass are coherent AFs (Dunne & Bench-Capon 2002); there are further examples in (Baroni & Giacomin 2008; Dvořák *et al.* 2014).

*This research has been supported by DFG (project BR 1817/7-1) and FWF (projects I1102 and P25518).

Importance of compact AFs mainly stems from the following two aspects. First, compact AFs possess a certain fairness behavior in the sense that each argument has the chance to be accepted. This might be a desired feature in some of the application areas such as decision support (Amgoud, Dimopoulos, & Moraitis 2008), where AFs are employed for a comparative evaluation of different options. Given that each argument appears in some extension ensures that the model is well-formed in the sense that it does not contain impossible options. The second and more concrete aspect is the issue of normal-forms of AFs. Indeed, compact AFs are attractive for such a normal-form, since none of the arguments can be removed without changing the extensions.

Following this idea we are interested in the question whether an arbitrary AF can be transformed into a compact AF without changing the outcome under the considered semantics. It is rather easy to see that under the *naive* semantics, which is defined as maximal conflict-free sets, any AF can be transformed into an equivalent compact AF. However, as has already been observed by Dunne *et al.* (2013), this is not true for other semantics. As an example consider the following AF F_1, where nodes represent arguments and directed edges represent attacks.



The *stable* extensions (conflict-free sets attacking all other arguments) of F_1 are $\{a, b, c\}$, $\{a, b', c'\}$, $\{a', b, c'\}$, $\{a', b', c\}$, $\{a, b, c'\}$, $\{a', b, c\}$, and $\{a, b', c\}$. It was shown in (Dunne *et al.* 2013) that there is no compact AF (in this case an F_1' not using argument x) which yields the same stable extensions as F_1. By the necessity of conflict-freeness any such compact AF would only allow conflicts between arguments a and a', b and b', and c and c', respectively. Moreover, there must be attacks in both directions for each of these conflicts in order to ensure stability. Hence any compact AF having the same stable extensions as F_1 necessarily yields $\{a', b', c'\}$ in addition. As we will see, all semantics under consideration share certain criteria which guarantee impossibility of a translation to a compact AF.

Like other subclasses, compact AFs decrease complexity of certain decision problems. This is obvious by the defini-

tion for credulous acceptance (does an argument occur in at least one extension). For skeptical acceptance (does an argument a occur in all extensions) in compact AFs this problem reduces to checking whether a is isolated. If yes, it is skeptically accepted; if no, a is connected to at least one further argument which has to be credulously accepted by the definition of compact AFs. But then, it is the case for any semantics which is based on conflict-free sets that a cannot be skeptically accepted, since it will not appear together with b in an extension. However, as we will see, the problem of verification (does a given set of arguments form an extension) remains coNP-hard for certain semantics, hence enumerating all extensions of an AF remains non-trivial.

An exact characterization of the collection of all sets of extensions which can be achieved by a compact AF under a given semantics σ seems rather challenging. We illustrate this on the example of stable semantics. Interestingly, we can provide an exact characterization under the condition that a certain conjecture holds: Given an AF F and two arguments which do not appear jointly in an extension of F, one can always add an attack between these two arguments (and potentially adapt other attacks in the AF) without changing the stable extensions. This conjecture is important for our work, but also an interesting question in and of itself.

To summarize, the main contributions of our work are:

- We define the classes of compact AFs for some of the most prominent semantics (namely naive, stable, stage, semi-stable and preferred) and provide a full picture of the relations between these classes. Then we show that the verification problem is still intractable for stage, semi-stable and preferred semantics.
- Moreover we use and extend recent results on maximal numbers of extensions (Baumann & Strass 2014) to give some impossibility-results for *compact realizability*. That is, we provide conditions under which for an AF with a certain number of extensions no translation to an equivalent (in terms of extensions) compact AF exists.
- Finally, we study *signatures* (Dunne *et al.* 2014) for compact AFs exemplified on the stable semantics. An exact characterization relies on the open explicit-conflict conjecture mentioned above. However, we give some sufficient conditions for an extension-set to be expressed as a stable-compact AF. For example, it holds that any AF with at most three stable extensions possesses an equivalent compact AF.

2 Preliminaries

In what follows, we recall the necessary background on abstract argumentation. For an excellent overview, we refer to (Baroni, Caminada, & Giacomin 2011).

Throughout the paper we assume a countably infinite domain $\mathfrak{A}$ of arguments. An *argumentation framework* (AF) is a pair $F = (A, R)$ where $A \subseteq \mathfrak{A}$ is a non-empty, finite set of arguments and $R \subseteq A \times A$ is the attack relation. The collection of all AFs is given as $AF_{\mathfrak{A}}$. For an AF $F = (B, S)$ we use A_F and R_F to refer to B and S, respectively. We write $a \mapsto_F b$ for $(a, b) \in R_F$ and $S \mapsto_F a$ (resp. $a \mapsto_F S$) if $\exists s \in S$ such that $s \mapsto_F a$ (resp. $a \mapsto_F s$). For $S \subseteq A$, the *range* of S (wrt. F), denoted S_F^+, is the set $S \cup \{b \mid S \mapsto_F b\}$.

Given $F = (A, R)$, an argument $a \in A$ is *defended* (in F) by $S \subseteq A$ if for each $b \in A$, such that $b \mapsto_F a$, also $S \mapsto_F b$. A set T of arguments is defended (in F) by S if each $a \in T$ is defended by S (in F). A set $S \subseteq A$ is *conflict-free* (in F), if there are no arguments $a, b \in S$, such that $(a, b) \in R$. $cf(F)$ denotes the set of all conflict-free sets in F. $S \in cf(F)$ is called *admissible* (in F) if S defends itself. $adm(F)$ denotes the set of admissible sets in F.

The semantics we study in this work are the naive, stable, preferred, stage, and semi-stable extensions. Given $F = (A, R)$ they are defined as subsets of $cf(F)$ as follows:

- $S \in naive(F)$, if there is no $T \in cf(F)$ with $T \supset S$
- $S \in stb(F)$, if $S \mapsto_F a$ for all $a \in A \setminus S$
- $S \in pref(F)$, if $S \in adm(F)$ and $\nexists T \in adm(F)$ s.t. $T \supset S$
- $S \in stage(F)$, if $\nexists T \in cf(F)$ with $T_F^+ \supset S_F^+$
- $S \in sem(F)$, if $S \in adm(F)$ and $\nexists T \in adm(F)$ s.t. $T_F^+ \supset S_F^+$

We will make frequent use of the following concepts.

Definition 1. Given $\mathbb{S} \subseteq 2^{\mathfrak{A}}$, $Arg_{\mathbb{S}}$ denotes $\bigcup_{S \in \mathbb{S}} S$ and $Pairs_{\mathbb{S}}$ denotes $\{(a, b) \mid \exists S \in \mathbb{S} : \{a, b\} \subseteq S\}$. $\mathbb{S}$ is called an *extension-set* (over $\mathfrak{A}$) if $Arg_{\mathbb{S}}$ is finite.

As is easily observed, for all considered semantics σ, $\sigma(F)$ is an extension-set for any AF F.

3 Compact Argumentation Frameworks

Definition 2. Given a semantics σ the set of *compact argumentation frameworks* under σ is defined as $CAF_\sigma = \{F \in AF_{\mathfrak{A}} \mid Arg_{\sigma(F)} = A_F\}$. We call an AF $F \in CAF_\sigma$ just σ-compact.

Of course the contents of CAF_σ differ with respect to the semantics σ. Concerning relations between the classes of compact AFs note that if for two semantics σ and θ it holds that $\sigma(F) \subseteq \theta(F)$ for any AF F, then also $CAF_\sigma \subseteq CAF_\theta$. Our first important result provides a full picture of the relations between classes of compact AFs under the semantics we consider.

Proposition 1. *1.* $CAF_{sem} \subset CAF_{pref}$*;*

2. $CAF_{stb} \subset CAF_\sigma \subset CAF_{naive}$ *for* $\sigma \in \{pref, sem, stage\}$*;*

3. $CAF_\theta \not\subseteq CAF_{stage}$ *and* $CAF_{stage} \not\subseteq CAF_\theta$ *for* $\theta \in \{pref, sem\}$.

Proof. (1) $CAF_{sem} \subseteq CAF_{pref}$ is by the fact that, in any AF F, $sem(F) \subseteq pref(F)$. Properness follows from the AF F' in Figure 1 (including the dotted part)[1]. Here $pref(F') = \{\{z\}, \{x_1, a_1\}, \{x_2, a_2\}, \{x_3, a_3\}, \{y_1, b_1\}, \{y_2, b_2\}, \{y_3, b_3\}\}$, but $sem(F') = (pref(F') \setminus \{\{z\}\})$, hence $F' \in CAF_{pref}$, but $F' \notin CAF_{sem}$.
(2) Let $\sigma \in \{pref, sem, stage\}$. The $\subseteq$-relations follow from the fact that, in any AF F, $stb(F) \subseteq \sigma(F)$ and each σ-extension is, by being conflict-free, part of some naive extension. The AF $(\{a, b\}, \{(a, b)\})$, which is compact under

[1] The construct in the lower part of the figure represents symmetric attacks between each pair of arguments.

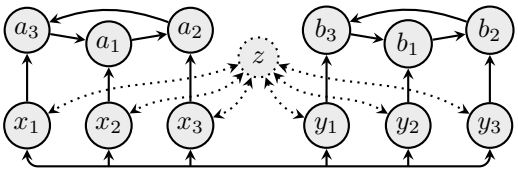
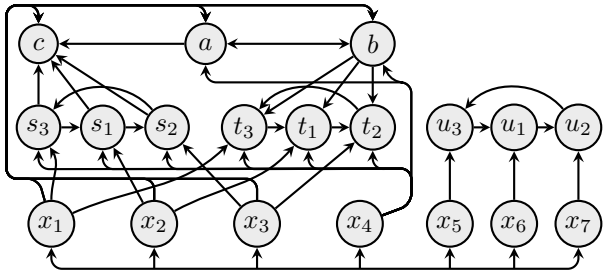


Figure 1: AFs illustrating the relations between various semantics.

naive but not under σ, and AF F from Figure 1 (now without the dotted part), which is compact under σ but not under stable, show that the relations are proper.
(3) The fact that F' from Figure 1 (again including the dotted part) is also not *stage*-compact shows $CAF_{pref} \not\subseteq CAF_{stage}$. Likewise, the AF G depicted below is *sem*-compact, but not *stage*-compact.



The reason for this is that argument a does not occur in any stage extension. Although $\{a, u_1, x_5\}$, $\{a, u_2, x_6\}$, $\{a, u_3, x_7\} \in sem(G)$, the range of any conflict-free set containing a is a proper subset of the range of every stable extension of G. $stage(G) = \{\{c, u_i, x_4\} \mid i \in \{1,2,3\}\} \cup \{\{b, u_i, s_j, x_{i+4}\} \mid i,j \in \{1,2,3\}\} \cup \{\{t_i, u_j, s_i, x_i\} \mid i,j \in \{1,2,3\}\}$. Hence $CAF_{sem} \not\subseteq CAF_{stage}$.
Finally, the AF $(\{a,b,c\}, \{(a,b),(b,c),(c,a)\})$ shows $CAF_{stage} \not\subseteq CAF_{\theta}$ for $\theta \in \{pref, sem\}$. □

Considering compact AFs obviously has effects on the computational complexity of reasoning. While credulous and skeptical acceptance are now easy (as discussed in the introduction) the next theorem shows that verifying extensions is still as hard as in general AFs.

Theorem 2. *For $\sigma \in \{pref, sem, stage\}$, AF $F = (A, R) \in CAF_\sigma$ and $E \subseteq A$, it is* coNP*-complete to decide whether $E \in \sigma(F)$.*

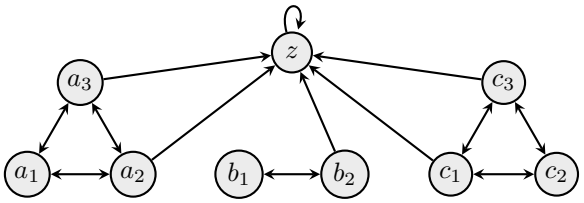
Proof. For all three semantics the problem is known to be in coNP (Caminada, Carnielli, & Dunne 2012; Dimopoulos & Torres 1996; Dvořák & Woltran 2011). For hardness we only give a (prototypical) proof for *pref*. We use a standard reduction from CNF formulas $\varphi(X) = \bigwedge_{c \in C} c$ with each clause $c \in C$ a disjunction of literals from X to an AF F_φ with arguments $A_\varphi = \{\varphi, \bar{\varphi}_1, \bar{\varphi}_2, \bar{\varphi}_3\} \cup C \cup X \cup \bar{X}$ and attacks (i) $\{(c, \varphi) \mid c \in C\}$, (ii) $\{(x, \bar{x}), (\bar{x}, x) \mid x \in X\}$, (iii) $\{(x, c) \mid x \text{ occurs in } c\} \cup \{(\bar{x}, c) \mid \neg x \text{ occurs in } c\}$, (iv) $\{(\varphi, \bar{\varphi}_1), (\bar{\varphi}_1, \bar{\varphi}_2), (\bar{\varphi}_2, \bar{\varphi}_3), (\bar{\varphi}_3, \bar{\varphi}_1)\}$, and (v) $\{(\bar{\varphi}_1, x), (\bar{\varphi}_1, \bar{x}) \mid x \in X\}$. It holds that φ is satisfiable iff there is an $S \neq \emptyset$ in $\sigma_1(F_\varphi)$ (Dimopoulos & Torres 1996). We extend F_φ with four new arguments $\{t_1, t_2, t_3, t_4\}$ and the following attacks: (a) $\{(t_i, t_j), (t_j, t_i) \mid 1 \leq i < j \leq 4\}$, (b) $\{(t_1, c) \mid c \in C\}$, (c) $\{(t_2, c), (t_2, \bar{\varphi}_2) \mid c \in C\}$ and (d) $\{(t_3, \bar{\varphi}_3)\}$. This extended AF is in CAF_{pref} and moreover $\{t_4\}$ is a preferred extension thereof iff $pref(F_\varphi) = \{\emptyset\}$ iff φ is unsatisfiable. □

4 Limits of Compact AFs

Extension-sets obtained from compact AFs satisfy certain structural properties. Knowing these properties can help us decide whether – given an extension-set $\mathbb{S}$ – there is a compact AF F such that $\mathbb{S}$ is exactly the set of extensions of F for a semantics σ. This is also known as *realizability*: A set $\mathbb{S} \subseteq 2^{\mathfrak{A}}$ is called *compactly realizable* under semantics σ iff there is a compact AF F with $\sigma(F) = \mathbb{S}$.

Among the most basic properties that are necessary for compact realizability, we find numerical aspects like possible numbers of σ-extensions.

Example 1. Consider the following AF F_2:



Let us determine the stable extensions of F_2. Clearly, taking one a_i, one b_i and one c_i yields a conflict-free set that is also stable as long as it attacks z. Thus from the $3 \cdot 2 \cdot 3 = 18$ combinations, only one (the set $\{a_1, b_1, c_2\}$) is not stable, whence F_2 has $18 - 1 = 17$ stable extensions. We note that this AF is not compact since z occurs in none of the extensions. Is there an equivalent stable-compact AF? The results of this section will provide us with a negative answer.

In (Baumann & Strass 2014) it was shown that there is a correspondence between the maximal number of stable extensions in argumentation frameworks and the maximal number of maximal independent sets in undirected graphs (Moon & Moser 1965). Recently, the result was generalized to further semantics (Dunne *et al.* 2014) and is stated below.[2] For any natural number n we define:

$$\sigma_{\max}(n) = \max\{|\sigma(F)| \mid F \in \text{AF}_n\}$$

$\sigma_{\max}(n)$ returns the maximal number of σ-extensions among all AFs with n arguments. Surprisingly, there is a closed expression for $\sigma_{\max}$.

Theorem 3. *The function $\sigma_{max}(n) : \mathbb{N} \to \mathbb{N}$ is given by*

$$\sigma_{max}(n) = \begin{cases} 1, & \text{if } n = 0 \text{ or } n = 1, \\ 3^s, & \text{if } n \geq 2 \text{ and } n = 3s, \\ 4 \cdot 3^{s-1}, & \text{if } n \geq 2 \text{ and } n = 3s+1, \\ 2 \cdot 3^s, & \text{if } n \geq 2 \text{ and } n = 3s+2. \end{cases}$$

What about the maximal number of σ-extensions on connected graphs? Does this number coincide with $\sigma_{\max}(n)$?

[2] In this section, unless stated otherwise we use σ as a placeholder for stable, semi-stable, preferred, stage and naive semantics.

The next theorem provides a negative answer to this question and thus, gives space for impossibility results as we will see. For a natural number n define

$$\sigma^{\text{con}}_{\max}(n) = \max\{|\sigma(F)| \mid F \in \text{AF}_n, F \text{ connected}\}$$

$\sigma^{\text{con}}_{\max}(n)$ returns the maximal number of σ-extensions among all *connected* AFs with n arguments. Again, a closed expression exists.

Theorem 4. *The function $\sigma^{con}_{max}(n) : \mathbb{N} \to \mathbb{N}$ is given by*

$$\sigma^{con}_{max}(n) = \begin{cases} n, & \text{if } n \leq 5, \\ 2 \cdot 3^{s-1} + 2^{s-1}, & \text{if } n \geq 6 \text{ and } n = 3s, \\ 3^s + 2^{s-1}, & \text{if } n \geq 6 \text{ and } n = 3s+1, \\ 4 \cdot 3^{s-1} + 3 \cdot 2^{s-2}, & \text{if } n \geq 6 \text{ and } n = 3s+2. \end{cases}$$

Proof. First some notations: for an AF $F = (A, R)$, denote its irreflexive version by $\mathit{irr}(F) = (A, R \setminus \{(a,a) \mid a \in A\})$; denote its symmetric version by $\mathit{sym}(F) = (A, R \cup \{(b,a) \mid (a,b) \in R\}$. Now for the proof. ($\leq$) Assume given a connected AF F. Obviously, $\mathit{naive}(F) \subseteq \mathit{naive}(\mathit{sym}(\mathit{irr}(F)))$. Thus, $|\mathit{naive}(F)| \leq |\mathit{naive}(\mathit{sym}(\mathit{irr}(F)))|$. Note that for any symmetric and irreflexive F, $\mathit{naive}(F) = \mathit{MIS}(\mathit{und}(F))$. Consequently, $|\mathit{naive}(F)| \leq |\mathit{MIS}(\mathit{und}(\mathit{sym}(\mathit{irr}(F))))|$. Fortunately, due to Theorem 2 in (Griggs, Grinstead, & Guichard 1988) the maximal number of maximal independent sets in connected n-graphs are exactly given by the claimed value range of $\sigma^{\text{con}}_{\max}(n)$. ($\geq$) Stable-realizing AFs can be derived by the extremal graphs w.r.t. MIS in connected graphs (consider Fig. 1 in (Griggs, Grinstead, & Guichard 1988)). Replacing undirected edges by symmetric directed attacks accounts for this.
In consideration of $\mathit{stb} \subseteq \mathit{stage} \subseteq \mathit{naive}$ we obtain: $\sigma^{\text{con}}_{\max}(n)$ provides a tight upper bound for $\sigma \in \{\mathit{stb}, \mathit{stage}, \mathit{naive}\}$. Finally, using $\mathit{stb} \subseteq \mathit{sem} \subseteq \mathit{pref}, \mathit{pref}(F) \subseteq \mathit{pref}(\mathit{sym}(\mathit{irr}(F)))$ and $\mathit{pref}(\mathit{sym}(\mathit{irr}(F))) = \mathit{stb}(\mathit{sym}(\mathit{irr}(F)))$ (compare Corollary 1 in (Baroni & Giacomin 2008)) we obtain that $\sigma^{\text{con}}_{\max}(n)$ even serves for $\sigma \in \{\mathit{sem}, \mathit{pref}\}$. □

A further interesting question concerning arbitrary AFs is whether all natural numbers less than $\sigma_{\max}(n)$ are compactly realizable.[3] The following theorem shows that there is a serious gap between the maximal and second largest number. For any positive natural n define

$$\sigma^2_{\max}(n) = \max\left(\{|\sigma(F)| \mid F \in \text{AF}_n\} \setminus \{\sigma_{\max}(n)\}\right)$$

$\sigma^2_{\max}(n)$ returns the second largest number of σ-extensions among all AFs with n arguments. Graph theory provides us with an expression.

Theorem 5. *Function $\sigma^2_{max}(n) : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ is given by*

$$\sigma^2_{max}(n) = \begin{cases} \sigma_{max}(n) - 1, & \text{if } 1 \leq n \leq 7, \\ \sigma_{max}(n) \cdot \frac{11}{12}, & \text{if } n \geq 8 \text{ and } n = 3s+1, \\ \sigma_{max}(n) \cdot \frac{8}{9}, & \text{otherwise.} \end{cases}$$

[3] We sometimes speak about realizing a natural number n and mean realizing an extension-set with n extensions.

Proof. ($\geq$) σ-realizing AFs can be derived by the extremal graphs w.r.t. the second largest number of MIS (consider Theorem 2.4 in (Jin & Li 2008)). Replacing undirected edges by symmetric directed attacks accounts for this. This means, the second largest number of σ-extensions is at least as large as the claimed value range.
($\leq$) If $n \leq 7$, there is nothing to prove. Given $F \in \text{AF}_n$ s.t. $n \geq 8$. Suppose, towards a contradiction, that $\sigma^2_{\max}(n) < |\sigma(F)| < \sigma_{\max}(n)$. It is easy to see that for any symmetric and irreflexive F, $\sigma(F) = \mathit{MIS}(\mathit{und}(F))$. Furthermore, due to Theorem 2.4 in (Jin & Li 2008) the second largest numbers of maximal independent sets in n-graphs are exactly given by the claimed value range of $\sigma^2_{\max}(n)$. Consequently, F cannot be symmetric and self-loop-free simultaneously. Hence, $|\sigma(F)| < |\sigma(\mathit{sym}(\mathit{irr}(F)))| = \sigma_{\max}(n)$. Note that up to isomorphisms the extremal graphs are uniquely determined (cf. Theorem 1 in (Griggs, Grinstead, & Guichard 1988)). Depending on the remainder of n on division by 3 we have K_3's for $n \equiv 0$, either one K_4 or two K_2's and the rest are K_3's in case of $n \equiv 1$ and one K_2 plus K_3's for $n \equiv 2$. Consequently, depending on the remainder we may thus estimate $|\sigma(F)| \leq k \cdot \sigma_{\max}(n)$ where $k \in \{\frac{2}{3}, \frac{3}{4}, \frac{1}{2}\}$. Since ($\geq$) is already shown we finally state $l \cdot \sigma_{\max}(n) \leq \sigma^2_{\max}(n) < |\sigma(F)| \leq \frac{3}{4} \cdot \sigma_{\max}(n)$ where $l \in \{\frac{11}{12}, \frac{8}{9}\}$. This is a clear contradiction concluding the proof. □

To showcase the intended usage of these theorems, we now prove that the AF F_2 seen earlier indeed has no equivalent compact AF.

Example 2. Recall that the (non-compact) AF F_2 we discussed previously had the extension-set $\mathbb{S}$ with $|\mathbb{S}| = 17$ and $|\mathit{Arg}_{\mathbb{S}}| = 8$. Is there a stable-compact AF with the same extensions? Firstly, nothing definitive can be said by Theorem 3 since $17 \leq 18 = \sigma_{\max}(8)$. Furthermore, in accordance with Theorem 4 the set $\mathbb{S}$ cannot be compactly σ-realized by a connected AF since $17 > 15 = \sigma^{\text{con}}_{\max}(8)$. Finally, using Theorem 5 we infer that the set $\mathbb{S}$ is not compactly σ-realizable because $\sigma^2_{\max}(8) = 16 < 17 < 18 = \sigma_{\max}(8)$.

The compactness property is instrumental here, since Theorem 5 has no counterpart in non-compact AFs. More generally, allowing additional arguments as long as they do not occur in extensions enables us to realize any number of stable extensions up to the maximal one.

Proposition 6. *Let n be a natural number. For each $k \leq \sigma_{max}(n)$, there is an AF F with $|\mathit{Arg}_{\mathit{stb}(F)}| = n$ and $|\mathit{stb}(F)| = k$.*

Proof. To realize k stable extensions with n arguments, we start with the construction for the maximal number from Theorem 3. We then subtract extensions as follows: We choose $\sigma_{\max}(n) - k$ arbitrary distinct stable extensions of the AF realizing the maximal number. To exclude them, we use the construction of Def. 9 in (Dunne *et al.* 2014). □

Now we are prepared to provide possible short cuts when deciding realizability of a given extension-set by initially simply counting the extensions. First some formal definitions.

Definition 3. Given an AF $F = (A, R)$, the component-structure $\mathcal{K}(F) = \{K_1, \ldots, K_n\}$ of F is the set of sets of arguments, where each K_i coincides with the arguments of a weakly connected component of the underlying graph; $\mathcal{K}_{\geq 2}(F) = \{K \in \mathcal{K}(F) \mid |K| \geq 2\}$.

Example 3. The AF $F = (\{a, b, c\}, \{(a, b)\})$ has component-structure $\mathcal{K}(F) = \{\{a, b\}, \{c\}\}$.

The component-structure $\mathcal{K}(F)$ gives information about the number n of components of F as well as the size $|K_i|$ of each component. Knowing the components of an AF, computing the σ-extensions can be reduced to computing the σ-extensions of each component and building the cross-product. The AF resulting from restricting F to component K_i is given by $F{\downarrow}_{K_i} = (K_i, R_F \cap K_i \times K_i)$.

Lemma 7. *Given an* AF F *with component-structure* $\mathcal{K}(F) = \{K_1, \ldots, K_n\}$ *it holds that the extensions in* $\sigma(F)$ *and the tuples in* $\sigma(F{\downarrow}_{K_1}) \times \cdots \times \sigma(F{\downarrow}_{K_n})$ *are in one-to-one correspondence.*

Given an extension-set $\mathbb{S}$ we want to decide whether $\mathbb{S}$ is realizable by a compact AF under semantics σ. For an AF $F = (A, R)$ with $\sigma(F) = \mathbb{S}$ we know that there cannot be a conflict between any pair of arguments in $\mathit{Pairs}_{\mathbb{S}}$, hence $R \subseteq \overline{\mathit{Pairs}_{\mathbb{S}}} = (A \times A) \setminus \mathit{Pairs}_{\mathbb{S}}$. In the next section, we will show that it is highly non-trivial to decide which of the attacks in $\overline{\mathit{Pairs}_{\mathbb{S}}}$ can be and should be used to realize $\mathbb{S}$. For now, the next proposition implicitly shows that for argument-pairs $(a, b) \notin \mathit{Pairs}_{\mathbb{S}}$, although there is not necessarily a direct conflict between a and b, they are definitely in the same component.

Proposition 8. *Given an extension-set* $\mathbb{S}$*, the component-structure* $\mathcal{K}(F)$ *of any* AF F *compactly realizing* $\mathbb{S}$ *under semantics* σ *(*$F \in \mathit{CAF}_\sigma$*,* $\sigma(F) = \mathbb{S}$*) is given by the equivalence classes of the transitive closure of* $\overline{\mathit{Pairs}_{\mathbb{S}}}$*,* $\left(\overline{\mathit{Pairs}_{\mathbb{S}}}\right)^*$.

Proof. Consider some extension-set $\mathbb{S}$ together with an AF $F \in \mathit{CAF}_\sigma$ with $\sigma(F) = \mathbb{S}$. We have to show that for any pair of arguments $a, b \in \mathit{Arg}_{\mathbb{S}}$ it holds that $(a, b) \in \left(\overline{\mathit{Pairs}_{\mathbb{S}}}\right)^*$ iff a and b are connected in the graph underlying F.

If a and b are connected in F, this means that there is a sequence $s_1, \ldots, s_n$ such that $a = s_1$, $b = s_n$, and $(s_1, s_2), \ldots, (s_{n-1}, s_n) \notin \mathit{Pairs}_{\mathbb{S}}$, hence $(a, b) \in \left(\overline{\mathit{Pairs}_{\mathbb{S}}}\right)^*$.

If $(a, b) \in \left(\overline{\mathit{Pairs}_{\mathbb{S}}}\right)^*$ then also there is a sequence $s_1, \ldots, s_n$ such that $a = s_1$, $b = s_n$, and $(s_1, s_2), \ldots, (s_{n-1}, s_n) \in \overline{\mathit{Pairs}_{\mathbb{S}}}$. Consider some $(s_i, s_{i+1}) \in \overline{\mathit{Pairs}_{\mathbb{S}}}$ and assume, towards a contradiction, that s_i occurs in another component of F than s_{i+1}. Recall that $F \in \mathit{CAF}_\sigma$, so each of s_i and s_{i+1} occur in some extension and $\sigma(F) \neq \emptyset$. Hence, by Lemma 7, there is some σ-extension $E \supseteq \{s_i, s_{i+1}\}$ of F, meaning that $(s_i, s_{i+1}) \in \mathit{Pairs}_{\mathbb{S}}$, a contradiction. Hence all s_i and s_{i+1} for $1 \leq i < n$ occur in the same component of F, proving that also a and b do so. □

We will denote the component-structure induced by an extension-set $\mathbb{S}$ as $\mathcal{K}(\mathbb{S})$. Note that, by Proposition 8, $\mathcal{K}(\mathbb{S})$ is equivalent to $\mathcal{K}(F)$ for every $F \in \mathit{CAF}_\sigma$ with $\sigma(F) = \mathbb{S}$. Given $\mathbb{S}$, the computation of $\mathcal{K}(\mathbb{S})$ can be done in polynomial time. With this we can use results from graph theory together with number-theoretical considerations in order to get impossibility results for compact realizability.

Recall that for a single connected component with n arguments the maximal number of stable extensions is denoted by $\sigma^{\text{con}}_{\text{max}}(n)$ and its values are given by Theorem 4. In the compact setting it further holds for a connected AF F with at least 2 arguments that $\sigma(F) \geq 2$.

Proposition 9. *Given an extension-set* $\mathbb{S}$ *where* $|\mathbb{S}|$ *is odd, it holds that if* $\exists K \in \mathcal{K}(\mathbb{S}) : |K| = 2$ *then* $\mathbb{S}$ *is not compactly realizable under semantics* σ.

Proof. Assume to the contrary that there is an $F \in \mathit{CAF}_\sigma$ with $\sigma(F) = \mathbb{S}$. We know that $\mathcal{K}(F) = \mathcal{K}(\mathbb{S})$. By assumption there is a $K \in \mathcal{K}(\mathbb{S})$ with $|K| = 2$, whence $|\sigma(K)| = 2$. Thus by Lemma 7 the total number of σ-extensions is even. Contradiction. □

Example 4. Consider the extension-set $\mathbb{S} = \{\{a, b, c\}, \{a, b', c'\}, \{a', b, c'\}, \{a', b', c\}, \{a, b, c'\}, \{a', b, c\}, \{a, b', c\}\} = \mathit{stb}(F_1)$ where F_1 is the non-compact AF from the introduction. There, it took us some effort to argue that $\mathbb{S}$ is not compactly *stb*-realizable. Proposition 9 now gives an easier justification: $\mathit{Pairs}_{\mathbb{S}}$ yields $\mathcal{K}(\mathbb{S}) = \{\{a, a'\}, \{b, b'\}, \{c, c'\}\}$. Thus $\mathbb{S}$ with $|\mathbb{S}| = 7$ cannot be realized.

We denote the set of possible numbers of σ-extensions of a compact AF with n arguments as $\mathcal{P}(n)$; likewise we denote the set of possible numbers of σ-extensions of a compact and *connected* AF with n arguments as $\mathcal{P}^c(n)$. Although we know that $p \in \mathcal{P}(n)$ implies $p \leq \sigma_{\text{max}}(n)$, there may be $q \leq \sigma_{\text{max}}(n)$ which are not realizable by a compact AF under σ; likewise for $q \in \mathcal{P}^c(n)$.

Clearly, any $p \leq n$ is possible by building an undirected graph with p arguments where every argument attacks all other arguments, a K_p, and filling up with k isolated arguments (k distinct copies of K_1) such that $k + p = n$. This construction obviously breaks down if we want to realize more extensions than we have arguments, that is, $p > n$. In this case, we have to use Lemma 7 and further graph-theoretic gadgets for addition and even a limited form of subtraction. Space does not permit us to go into too much detail, but let us show how for $n = 7$ any number of extensions up to the maximal number 12 is realizable. For $12 = 3 \cdot 4$, Theorem 3 yields the realization, a disjoint union of a K_3 and a K_4 (). For the remaining numbers, we have that $8 = 2 \cdot 4 \cdot 1$ and so we can combine a K_2, a K_4 and a K_1 (); likewise, $9 = 3 \cdot 3 \cdot 1$ (); $10 = 3 \cdot 3 + 1$ () and finally $11 = 3 \cdot 4 - 1$ (). These small examples already show that $\mathcal{P}$ and $\mathcal{P}^c$ are closely intertwined and let us deduce some general corollaries: Firstly, $\mathcal{P}^c(n) \subseteq \mathcal{P}(n)$ since connected AFs are a subclass of AFs. Next, $\mathcal{P}(n) \subseteq \mathcal{P}(n+1)$ as in the step from to . We even know that $\mathcal{P}(n) \subsetneq \mathcal{P}(n+1)$ since $\sigma_{\text{max}}(n+1) \in \mathcal{P}(n+1) \setminus \mathcal{P}(n)$. Furthermore, whenever $p \in \mathcal{P}(n)$, then $p + 1 \in \mathcal{P}^c(n+1)$, as in the step from  to  . The construction that goes from 12 to

11 above obviously only works if there are two weakly connected components overall, which underlines the importance of the component structure of the realizing AF. Indeed, multiplication of extension numbers of single components is our only chance to achieve overall numbers that are substantially larger than the number of arguments. This is what we will turn to next. Having to leave the exact contents of $\mathcal{P}(n)$ and $\mathcal{P}^c(n)$ open, we can still state the following result:

Proposition 10. *Let $\mathbb{S}$ be an extension-set that is compactly realizable under semantics σ where $\mathcal{K}_{\geq 2}(\mathbb{S}) = \{K_1, \ldots, K_n\}$. Then for each $1 \leq i \leq n$ there is a $p_i \in \mathcal{P}^c(|K_i|)$ such that $|\mathbb{S}| = \prod_{i=1}^{n} p_i$.*

Proof. First note that components of size 1 can be ignored since they have no impact on the number of σ-extensions. Lemma 7 also implies that the number of σ-extensions of an AF with multiple components is the product of the number of σ-extensions of each component. Since the factor of any component K_i must be in $\mathcal{P}^c(|K_i|)$ the result follows. □

Example 5. Consider the extension-set $\mathbb{S}' = \{\{a, b, c\}, \{a, b', c'\}, \{a', b, c'\}, \{a', b', c\}\}$. (In fact there exists a (non-compact) AF F with $stb(F) = \mathbb{S}'$). We have the same component-structure $\mathcal{K}(\mathbb{S}') = \mathcal{K}(\mathbb{S})$ as in Example 4, but since now $|\mathbb{S}'| = 4$ we cannot use Proposition 9 to show impossibility of realization in terms of a compact AF. But with Proposition 10 at hand we can argue in the following way: $\mathcal{P}^c(2) = \{2\}$ and since $\forall K \in \mathcal{K}(\mathbb{S}') : |K| = 2$ it must hold that $|\mathbb{S}| = 2 \cdot 2 \cdot 2 = 8$, which is obviously not the case.

In particular, we have a straightforward non-realizability criterion whenever $|\mathbb{S}|$ is prime: the AF (if any) must have at most one weakly connected component of size greater than two. Theorem 4 gives us the maximal number of σ-extensions in a single weakly connected component. Thus whenever the number of desired extensions is larger than that number and prime, it cannot be realized.

Corollary 11. *Let extension-set $\mathbb{S}$ with $|Arg_{\mathbb{S}}| = n$ be compactly realizable under σ. If $|\mathbb{S}|$ is a prime number, then $|\mathbb{S}| \leq \sigma^{con}_{max}(n)$.*

Example 6. Let $\mathbb{S}$ be an extension-set with $|Arg_{\mathbb{S}}| = 9$ and $|\mathbb{S}| = 23$. We find that $\sigma^{\text{con}}_{\text{max}}(9) = 2 \cdot 3^2 + 2^2 = 22 < 23 = |\mathbb{S}|$ and thus $\mathbb{S}$ is not compactly realizable under semantics σ.

We can also make use of the derived component structure of an extension-set $\mathbb{S}$. Since the total number of extensions of an AF is the product of these numbers for its weakly connected components (Lemma 7), each non-trivial component contributes a non-trivial amount to the total. Hence if there are more components than the factorization of $|\mathbb{S}|$ has primes in it, then $\mathbb{S}$ cannot be realized.

Corollary 12. *Let extension-set $\mathbb{S}$ be compactly realizable under σ and $f_1^{z_1} \cdot \ldots \cdot f_m^{z_m}$ be the integer factorization of $|\mathbb{S}|$, where $f_1, \ldots, f_m$ are prime numbers. Then $z_1 + \ldots + z_m \geq |\mathcal{K}_{\geq 2}(\mathbb{S})|$.*

Example 7. Consider an extension-set $\mathbb{S}$ containing 21 extensions and $|\mathcal{K}(\mathbb{S})| = 3$. Since $21 = 3^1 * 7^1$ and further $1 + 1 < 3$, $\mathbb{S}$ is not compactly realizable under semantics σ.

5 Capabilities of Compact AFs

The results in the previous section made clear that the restriction to compact AFs entails certain limits in terms of compact realizability. Here we provide some results approaching an exact characterization of the capabilities of compact AFs with a focus on stable semantics.

5.1 C-Signatures

The *signature* of a semantics σ is defined as $\Sigma_\sigma = \{\sigma(F) \mid F \in AF_{\mathfrak{A}}\}$ and contains all possible sets of extensions an AF can possess under σ (see (Dunne *et al.* 2014) for characterizations of such signatures). We first provide alternative, yet equivalent, characterizations of the signatures of some the semantics under consideration. Then we strengthen the concept of signatures to "compact" signatures (c-signatures), which contain all extension-sets realizable with compact AFs.

The most central concept when structurally analyzing extension-sets is captured by the *Pairs*-relation from Definition 1. Whenever two arguments a and b occur jointly in some element S of extension-set $\mathbb{S}$ (i.e. $(a, b) \in Pairs_{\mathbb{S}}$) there cannot be a conflict between those arguments in an AF having $\mathbb{S}$ as solution under any standard semantics. $(a, b) \in Pairs_{\mathbb{S}}$ can be read as "evidence of no conflict" between a and b in $\mathbb{S}$. Hence, the *Pairs*-relation gives rise to sets of arguments that are conflict-free in any AF realizing $\mathbb{S}$.

Definition 4. Given an extension-set $\mathbb{S}$, we define

- $\mathbb{S}^{cf} = \{S \subseteq Arg_{\mathbb{S}} \mid \forall a, b \in S : (a, b) \in Pairs_{\mathbb{S}}\}$;
- $\mathbb{S}^{+} = \max_{\subseteq} \mathbb{S}^{cf}$.

To show that the characterizations of signatures in Proposition 13 below are indeed equivalent to the ones given in (Dunne *et al.* 2014) we first recall some definitions from there.

Definition 5. For an extension-set $\mathbb{S} \subseteq 2^{\mathfrak{A}}$, the *downward-closure* of $\mathbb{S}$ is defined as $dcl(\mathbb{S}) = \{S' \subseteq S \mid S \in \mathbb{S}\}$. Moreover, $\mathbb{S}$ is called

- *incomparable*, if for all $S, S' \in \mathbb{S}$, $S \subseteq S'$ implies $S{=}S'$,
- *tight* if for all $S \in \mathbb{S}$ and $a \in Arg_{\mathbb{S}}$ it holds that if $(S \cup \{a\}) \notin \mathbb{S}$ then there exists an $s \in S$ such that $(a, s) \notin Pairs_{\mathbb{S}}$.

Proposition 13. $\Sigma_{naive} = \{\mathbb{S} \neq \emptyset \mid \mathbb{S} = \mathbb{S}^{+}\}$;
$\Sigma_{stb} = \{\mathbb{S} \mid \mathbb{S} \subseteq \mathbb{S}^{+}\}$; $\Sigma_{stage} = \{\mathbb{S} \neq \emptyset \mid \mathbb{S} \subseteq \mathbb{S}^{+}\}$.

Proof. Being aware of Theorem 1 from (Dunne *et al.* 2014) we have to show that, given an extension-set $\mathbb{S} \subseteq 2^{\mathfrak{A}}$ the following hold:

1. $\mathbb{S}$ is incomparable and tight iff $\mathbb{S} \subseteq \mathbb{S}^{+}$,
2. $\mathbb{S}$ is incomparable and $dcl(\mathbb{S})$ is tight iff $\mathbb{S} = \mathbb{S}^{+}$.

(1) $\Rightarrow$: Consider an incomparable and tight extension-set $\mathbb{S}$ and assume that $\mathbb{S} \not\subseteq \mathbb{S}^{+}$. To this end let $S \in \mathbb{S}$ with $S \notin \mathbb{S}^{+}$. Since $S \in \mathbb{S}^{cf}$ by definition, there must be some $S' \supset S$ with $S' \in \mathbb{S}^{+}$. $S' \notin \mathbb{S}$ holds by incomparability of $\mathbb{S}$. But $S' \in \mathbb{S}^{+}$ means that there is some $a \in (S' \setminus S)$ such that $\forall s \in S : (a, s) \in Pairs_{\mathbb{S}}$, a contradiction to the assumption that $\mathbb{S}$ is tight.

$\Leftarrow$: Let $\mathbb{S}$ be an extension-set such that $\mathbb{S} \subseteq \mathbb{S}^+$. Incomparability is clear. Now assume, towards a contradiction, that are some $S \in \mathbb{S}$ and $a \in \mathit{Arg}_{\mathbb{S}}$ such that $(S \cup \{a\}) \notin \mathbb{S}$ and $\forall s \in S : (a, s) \in \mathit{Pairs}_{\mathbb{S}}$. Then there is some $S' \supseteq (S \cup \{a\})$ with $S' \in \mathbb{S}^+$, a contradiction to $S \in \mathbb{S}^+$.
(2) $\Rightarrow$: Consider an incomparable extension-set $\mathbb{S}$ where $\mathit{dcl}(\mathbb{S})$ is tight and assume that $\mathbb{S} \neq \mathbb{S}^+$. Note that $\mathit{Pairs}_{\mathbb{S}} = \mathit{Pairs}_{\mathit{dcl}(\mathbb{S})}$. Since $\mathit{dcl}(\mathbb{S})$ being tight implies that $\mathbb{S}$ is tight (cf. Lemma 2.1 in (Dunne *et al.* 2014)), $\mathbb{S} \subseteq \mathbb{S}^+$ follows by (1). Now assume there is some $S \in \mathbb{S}^+$ with $S \notin \mathbb{S}$. Note that $|S| \geq 3$. Now let $S' \subset S$ and $a \in (S \setminus S')$ such that $S' \in \mathit{dcl}(\mathbb{S})$ and $(S' \cup \{a\}) \notin \mathit{dcl}(\mathbb{S})$. Such an S' exists since for each pair of arguments $a, b \in S'$, $(a, b) \in \mathit{Pairs}_{\mathbb{S}}$ holds as $S \in \mathbb{S}^+$. Since also $\forall s \in S' : (a, s) \in \mathit{Pairs}_{\mathbb{S}}$, we get a contradiction to the assumption that $\mathit{dcl}(\mathbb{S})$ is tight.
$\Leftarrow$: Consider an extension-set $\mathbb{S}$ with $\mathbb{S} = \mathbb{S}^+$. Incomparability is straight by definition. Now assume, towards a contradiction, that are some $S \in \mathit{dcl}(\mathbb{S})$ and $a \in \mathit{Arg}_{\mathbb{S}}$ such that $(S \cup \{a\}) \notin \mathit{dcl}(\mathbb{S})$ and $\forall s \in S : (a, s) \in \mathit{Pairs}_{\mathbb{S}}$. Then $(S \cup \{a\}) \in \mathbb{S}^{cf}$, and moreover there is some $S' \supseteq (S \cup \{a\})$ with $S' \in \mathbb{S}^+$ and $S' \notin \mathbb{S}$, a contradiction to $\mathbb{S} = \mathbb{S}^+$. □

Let us now turn to signatures for compact AFs.

Definition 6. The *c-signature* Σ^c_σ of a semantics σ is defined as

$$\Sigma^c_\sigma = \{\sigma(F) \mid F \in \mathit{CAF}_\sigma\}.$$

It is clear that $\Sigma^c_\sigma \subseteq \Sigma_\sigma$ holds for any semantics. The following result is mainly by the fact that the canonical AF

$$F^{cf}_{\mathbb{S}} = (A^{cf}_{\mathbb{S}}, R^{cf}_{\mathbb{S}}) = (\mathit{Arg}_{\mathbb{S}}, (\mathit{Arg}_{\mathbb{S}} \times \mathit{Arg}_{\mathbb{S}}) \setminus \mathit{Pairs}_{\mathbb{S}})$$

has $\mathbb{S}^+$ as extensions under all semantics under consideration and by extension-sets obtained from non-compact AFs which definitely cannot be transformed to equivalent compact AFs.

The following technical lemma makes this clearer.

Lemma 14. *Given a non-empty extension-set $\mathbb{S}$, it holds that $\sigma(F^{cf}_{\mathbb{S}}) = \mathbb{S}^+$ where $\sigma \in \{\mathit{naive}, \mathit{stb}, \mathit{stage}, \mathit{pref}, \mathit{sem}\}$.*

Proof. naive: The set $\mathit{naive}(F^{cf}_{\mathbb{S}})$ contains the $\subseteq$-maximal elements of $\mathit{cf}(F^{cf}_{\mathbb{S}})$ just as $\mathbb{S}^+$ does of $\mathbb{S}^{cf}$. Therefore $\mathit{naive}(F^{cf}_{\mathbb{S}}) = \mathbb{S}^+$ follows directly from the obvious fact that $\mathit{cf}(F^{cf}_{\mathbb{S}}) = \mathbb{S}^{cf}$.
stb, *stage*, *pref*, *sem*: Follow from the fact that for the symmetric AF $F^{cf}_{\mathbb{S}}$, $\mathit{naive}(F^{cf}_{\mathbb{S}}) = \mathit{stb}(F^{cf}_{\mathbb{S}}) = \mathit{stage}(F^{cf}_{\mathbb{S}}) = \mathit{pref}(F^{cf}_{\mathbb{S}}) = \mathit{sem}(F^{cf}_{\mathbb{S}})$ (Coste-Marquis, Devred, & Marquis 2005). □

Proposition 15. *It holds that (1) $\Sigma^c_{\mathit{naive}} = \Sigma_{\mathit{naive}}$; and (2) $\Sigma^c_\sigma \subset \Sigma_\sigma$ for $\sigma \in \{\mathit{stb}, \mathit{stage}, \mathit{sem}, \mathit{pref}\}$.*

Proof. $\Sigma^c_{\mathit{naive}} = \Sigma_{\mathit{naive}}$ follows directly from the facts that $\mathit{naive}(F^{cf}_{\mathbb{S}}) = \mathbb{S}^+$ (cf. Lemma 14) and $F^{cf}_{\mathbb{S}} \in \mathit{CAF}_{\mathit{naive}}$.
stb, *stage*: Consider the extension-set $\mathbb{S} = \{\{a, b, c\}, \{a, b, c'\}, \{a, b', c\}, \{a, b', c'\}, \{a', b, c\}, \{a', b, c'\}, \{a', b', c\}\}$ from the example in the introduction. It is easy to verify that $\mathbb{S} \subseteq \mathbb{S}^+$, thus $\mathbb{S} \in \Sigma_{\mathit{stb}}$ and $\mathbb{S} \in \Sigma_{\mathit{stage}}$. The AF realizing $\mathbb{S}$ under *stb* and *stage* is F_1 from the introduction. We now show that there is no AF $F = (\mathit{Arg}_{\mathbb{S}}, R)$ such that $\mathit{stb}(F) = \mathbb{S}$ or $\mathit{stage}(F) = \mathbb{S}$. First, given that the sets in $\mathbb{S}$ must be conflict-free the only possible attacks in R are (a, a'), (a', a), (b, b'), (b', b), (c, c'), (c', c). We next argue that all of them must be in R. First consider the case of *stb*. As $\{a, b, c\} \in \mathit{stb}(F)$ it attacks a' and the only chance to do so is $(a, a') \in R$ and similar as $\{a', b, c\} \in \mathit{stb}(F)$ it attacks a and the only chance to do so is $(a', a) \in R$. By symmetry we obtain $\{(b, b'), (b', b), (c, c'), (c', c)\} \subseteq R$. Now let us consider the case of *stage*. As $\{a, b, c\} \in \mathit{stage}(F) \subseteq \mathit{naive}(F)$ either $(a, a') \in R$ or $(a', a) \in R$. Consider $(a, a') \notin R$ then $\{a', b, c\}^+_F \supset \{a, b, c\}^+_F$, contradicting that $\{a, b, c\}$ is a stage extension. The same holds for pairs (b, b') and (c, c'); thus for both cases we obtain $R = \{(a, a'), (a', a), (b, b'), (b', b), (c, c'), (c', c)\}$. However, for the resulting framework $F = (A, R)$, we have that $\{a', b', c'\} \in \mathit{stb}(F) = \mathit{stage}(F)$, but $\{a', b', c'\} \notin \mathbb{S}$. Hence we know that $\mathbb{S} \notin \Sigma^c_{\mathit{stb}}$.

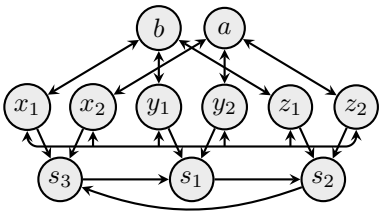
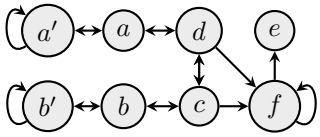


Figure 2: AF compactly realizing an extension-set $\mathbb{S} \not\subseteq \mathbb{S}^+$ under *pref*.

pref, *sem*: Let $\sigma \in \{\mathit{pref}, \mathit{sem}\}$ and consider $\mathbb{S} = \{\{a, b\}, \{a, c, e\}, \{b, d, e\}\}$. The figure below shows an AF (with additional arguments) realizing $\mathbb{S}$ under *pref* and *sem*. Hence $\mathbb{S} \in \Sigma_\sigma$ holds.



Now suppose there exists an AF $F = (\mathit{Arg}_{\mathbb{S}}, R)$ such that $\sigma(F) = \mathbb{S}$. Since $\{a, c, e\}, \{b, d, e\} \in \mathbb{S}$, it is clear that R must not contain an edge involving e. But then, e is contained in each $E \in \sigma(F)$. It follows that $\sigma(F) \neq \mathbb{S}$. □

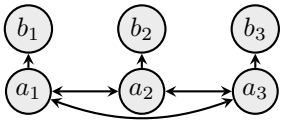
For ordinary signatures it holds that $\Sigma_{\mathit{naive}} \subset \Sigma_{\mathit{stage}} = (\Sigma_{\mathit{stb}} \setminus \{\emptyset\}) \subset \Sigma_{\mathit{sem}} = \Sigma_{\mathit{pref}}$ (Dunne *et al.* 2014). This picture changes when considering the relationship of c-signatures.

Proposition 16. *$\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stb}}$; $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stage}}$; $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{sem}}$; $\Sigma^c_{\mathit{naive}} \subset \Sigma^c_\sigma$ for $\sigma \in \{\mathit{stb}, \mathit{stage}, \mathit{sem}\}$; $\Sigma^c_{\mathit{stb}} \subseteq \Sigma^c_{\mathit{sem}}$; $\Sigma^c_{\mathit{stb}} \subseteq \Sigma^c_{\mathit{stage}}$.*

Proof. $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stb}}$, $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stage}}$: For the extension-set $\mathbb{S} = \{\{a, b\}, \{a, x_1, s_1\}, \{a, y_1, s_2\}, \{a, z_1, s_3\}, \{b, x_2, s_1\}, \{b, y_2, s_2\}, \{b, z_2, s_3\}\}$ it does not hold that $\mathbb{S} \subseteq \mathbb{S}^+$ (as $\{a, b, s_1\}, \{a, b, s_2\}, \{a, b, s_3\} \in \mathbb{S}^{cf}$, hence $\{a, b\} \notin \mathbb{S}^+$), but there is a compact AF F realizing $\mathbb{S}$ under the preferred semantics, namely the one depicted in Figure 2. Hence $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stb}}$ and $\Sigma^c_{\mathit{pref}} \not\subseteq \Sigma^c_{\mathit{stage}}$.

$\Sigma^c_{pref} \not\subseteq \Sigma^c_{sem}$: Let $\mathbb{T} = (\mathbb{S} \cup \{\{x_1, x_2, s_1\}, \{y_1, y_2, s_2\}, \{z_1, z_2, s_3\}\})$ and assume there is some $F = (Arg_{\mathbb{T}}, R)$ compactly realizing $\mathbb{T}$ under the semi-stable semantics. Consider the extensions $S = \{a, x_1, s_1\}$ and $T = \{x_1, x_2, s_1\}$. There must be a conflict between a and x_2, otherwise $(S \cup T) \in sem(F)$. If $(a, x_2) \in R$ then, since T must defend itself and $(s_1, a), (x_1, a) \in Pairs_{\mathbb{T}}$, also $(x_2, a) \in R$. On the other hand if $(x_2, a) \in R$ then, since $\{a, b\}$ must defend itself and $(b, x_2) \in Pairs_{\mathbb{T}}$, also $(a, x_2) \in R$. Hence, by all symmetric cases we get $\{(a, \alpha_1), (\alpha_1, a), (b, \alpha_2), (\alpha_2, b) \mid \alpha \in \{x, y, z\}\} \subseteq R$. Now as $U = \{a, b\} \in \mathbb{T}$ and U must not be in conflict with any of s_1, s_2, and s_3, each s_i must have an attacker which is not attacked by any a, b, or s_i. Hence wlog. $\{(s_1, s_2), (s_2, s_3), (s_3, s_1)\} \subseteq R$. Again consider extension S and observe that s_1 must be defended from s_3, hence $(x_1, s_3) \in R$. We know that $S^+_F \supseteq (Arg_{\mathbb{T}} \setminus \{y_1, z_1\})$. Now we observe that S has to attack both y_1 and z_1 since otherwise either S would not defend itself or y_1 (resp. z_1) would have to be part of S. But this leads us to a contradiction because $S^+_F = Arg_{\mathbb{T}}$, but $U^+_F \subset Arg_{\mathbb{T}}$, meaning that U cannot be a semi-stable extension of F. $\Sigma^c_{pref} \not\subseteq \Sigma^c_{sem}$ now follows from the fact that $pref(F') = \mathbb{T}$ for $F' = (A_F, R_F \setminus \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_1) \mid \alpha \in \{x, y, z\}\})$ where F is the AF depicted in Figure 2.

$\Sigma^c_{naive} \subset \Sigma^c_\sigma$ for $\sigma \in \{stb, stage, sem\}$: First of all note that any extensions-set compactly realizable under *naive* is compactly realizable under σ (by making the AF symmetric). Now consider the extension-set $\mathbb{S} = \{\{a_1, b_2, b_3\}, \{a_2, b_1, b_3\}, \{a_3, b_1, b_2\}\}$. $\mathbb{S} \neq \mathbb{S}^+$ since $\{b_1, b_2, b_3\} \in \mathbb{S}^+$, hence $\mathbb{S} \notin \Sigma^c_{naive}$. $\Sigma^c_{naive} \subset \Sigma^c_\sigma$ follows from the fact that the AF below compactly realizes $\mathbb{S}$ under σ.



$\Sigma^c_{stb} \subseteq \Sigma^c_{sem}$, $\Sigma^c_{stb} \subseteq \Sigma^c_{stage}$: Follow from the fact that $stage(F) = sem(F) = stb(F)$ for any $F \in CAF_{stb}$ (Caminada, Carnielli, & Dunne 2012). □

5.2 The Explicit-Conflict Conjecture

So far we only have exactly characterized c-signatures for the naive semantics (Proposition 15). Deciding membership of an extension-set in the c-signature of the other semantics is more involved. In what follows we focus on stable semantics in order to illustrate difficulties and subtleties in this endeavor.

Although there are, as Proposition 1 showed, more compact AFs for *naive* than for *stb*, one can express a greater diversity of outcomes with the stable semantics, i.e. $\mathbb{S} = \mathbb{S}^+$ does not necessarily hold. Consider some AF F with $\mathbb{S} = stb(F)$. By Proposition 13 we know that $\mathbb{S} \subseteq \mathbb{S}^+$ must hold. Now we want to compactly realize extension-set $\mathbb{S}$ under *stb*. If $\mathbb{S} = \mathbb{S}^+$, then we can obviously find a compact AF realizing $\mathbb{S}$ under *stb*, since $F^{cf}_{\mathbb{S}}$ will do so. On the other hand, if $\mathbb{S} \neq \mathbb{S}^+$ we have to find a way to handle the argument-sets in $\mathbb{S}^- = \mathbb{S}^+ \setminus \mathbb{S}$. In words, each $S \in \mathbb{S}^-$ is a $\subseteq$-maximal set with evidence of no conflict, which is not contained in $\mathbb{S}$.

Now consider some AF $F' \in CAF_{stb}$ having $\mathbb{S} \subsetneq \mathbb{S}^+$ as its stable extensions. Further take some $S \in \mathbb{S}^-$. There cannot be a conflict within S in F', hence we must be able to map S to some argument $t \in (Arg_{\mathbb{S}} \setminus S)$ not attacked by S in F'. Still, the collection of these mappings must fulfill certain conditions in order to preserve a justification for all $S \in \mathbb{S}$ to be a stable extension and not to give rise to other stable extensions. We make these things more formal.

Definition 7. Given an extension-set $\mathbb{S}$, an *exclusion-mapping* is the set

$$\mathfrak{R}_{\mathbb{S}} = \bigcup_{S \in \mathbb{S}^-} \{(s, \mathfrak{f}_{\mathbb{S}}(S)) \mid s \in S \text{ s.t. } (s, \mathfrak{f}_{\mathbb{S}}(S)) \notin Pairs_{\mathbb{S}}\}$$

where $\mathfrak{f}_{\mathbb{S}} : \mathbb{S}^- \to Arg_{\mathbb{S}}$ is a function with $\mathfrak{f}_{\mathbb{S}}(S) \in (Arg_{\mathbb{S}} \setminus S)$.

Definition 8. A set $\mathbb{S} \subseteq 2^{\mathfrak{A}}$ is called *independent* if there exists an antisymmetric exclusion-mapping $\mathfrak{R}_{\mathbb{S}}$ such that it holds that

$$\forall S \in \mathbb{S} \forall a \in (Arg_{\mathbb{S}} \setminus S) : \exists s \in S : (s, a) \notin (\mathfrak{R}_{\mathbb{S}} \cup Pairs_{\mathbb{S}}).$$

The concept of independence suggests that the more separate the elements of some extension-set $\mathbb{S}$ are, the less critical is $\mathbb{S}^-$. An independent $\mathbb{S}$ allows to find the required orientation of attacks to exclude sets from $\mathbb{S}^-$ from the stable extensions without interferences.

Theorem 17. *For every independent extension-set $\mathbb{S}$ with $\mathbb{S} \subseteq \mathbb{S}^+$ it holds that $\mathbb{S} \in \Sigma^c_{stb}$.*

Proof. Consider, given an independent extension-set $\mathbb{S}$ and an antisymmetric exclusion-mapping $\mathfrak{R}_{\mathbb{S}}$ fulfilling the independence-condition (cf. Definition 8), the AF $F^{stb}_{\mathbb{S}} = (Args_{\mathbb{S}}, R^{stb}_{\mathbb{S}})$ with $R^{stb}_{\mathbb{S}} = (R^{cf}_{\mathbb{S}} \setminus \mathfrak{R}_{\mathbb{S}})$. We show that $stb(F^{stb}_{\mathbb{S}}) = \mathbb{S}$. First note that $stb(F^{cf}_{\mathbb{S}}) = \mathbb{S}^+ \supseteq \mathbb{S}$. As $\mathfrak{R}_{\mathbb{S}}$ is antisymmetric, one direction of each symmetric attack of $F^{cf}_{\mathbb{S}}$ is still in $F^{stb}_{\mathbb{S}}$. Hence $stb(F^{stb}_{\mathbb{S}}) \subseteq \mathbb{S}^+$.

$stb(F^{stb}_{\mathbb{S}}) \subseteq \mathbb{S}$: Consider some $S \in stb(F^{stb}_{\mathbb{S}})$ and assume that $S \notin \mathbb{S}$, i.e. $S \in \mathbb{S}^-$. Since $\mathfrak{R}_{\mathbb{S}}$ is an exclusion-mapping fulfilling the independence-condition by assumption, there is an argument $\mathfrak{f}_{\mathbb{S}}(S) \in (Arg_{\mathbb{S}} \setminus S)$ such that $\{(s, \mathfrak{f}_{\mathbb{S}}(S)) \mid s \in S, (s, \mathfrak{f}_{\mathbb{S}}(S)) \notin Pairs_{\mathbb{S}}\} \subseteq \mathfrak{R}_{\mathbb{S}}$. But then, by construction of $F^{stb}_{\mathbb{S}}$, there is no $a \in S$ such that $(a, \mathfrak{f}_{\mathbb{S}}(S)) \in R^{stb}_{\mathbb{S}}$, a contradiction to $S \in stb(F^{stb}_{\mathbb{S}})$.

$stb(F^{stb}_{\mathbb{S}}) \supseteq \mathbb{S}$: Consider some $S \in \mathbb{S}$ and assume that $S \notin stb(F^{stb}_{\mathbb{S}})$. We know that S is conflict-free in $F^{stb}_{\mathbb{S}}$. Therefore there must be some $t \in (Arg_{\mathbb{S}} \setminus S)$ with $S \not\mapsto_{F^{stb}_{\mathbb{S}}} t$. Hence $\forall s \in S : (s, t) \in (Pairs_{\mathbb{S}} \cup \mathfrak{R}_{\mathbb{S}})$, a contradiction to the assumption that $\mathbb{S}$ is independent. □

Corollary 18. *For every $\mathbb{S} \in \Sigma_{stb}$, with $|\mathbb{S}| \leq 3$, $\mathbb{S} \in \Sigma^c_{stb}$.*

Proof. It is easy to see that for an extension-set $\mathbb{S}$ with $|\mathbb{S}| \leq 3$ it holds that $|\mathbb{S}^-| \leq 1$. If $\mathbb{S}^- = \emptyset$ we are done; if $\mathbb{S}^- = \{S\}$ observe that by $\mathbb{S} \subseteq \mathbb{S}^+$ for each $T \in \mathbb{S}$ there is some $t \in T$ with $t \notin S$. Hence choosing arbitrary $T \in \mathbb{S}$ and $t \in T$ with $t \notin S$ yields the antisymmetric exclusion-mapping $\mathfrak{R}_{\mathbb{S}} = \{(s, t) \mid s \in S \text{ s.t. } (s, t) \notin Pairs_{\mathbb{S}}\}$ which fulfills the independence-condition from Definition 8. □

Theorem 17 gives a sufficient condition for an extension-set to be contained in Σ^c_{stb}. Section 4 provided necessary conditions with respect to number of extensions. As these conditions do not match, we have not arrived at an exact characterization of the c-signature for stable semantics yet. In what follows, we identify the missing step which we have to leave open but, as we will see, results in an interesting problem of its own. Let us first define a further class of frameworks.

Definition 9. We call an AF $F = (A, R)$ conflict-explicit under semantics σ iff for each $a, b \in A$ such that $(a, b) \notin \mathit{Pairs}_{\sigma(F)}$, we find $(a, b) \in R$ or $(b, a) \in R$ (or both).

In words, a framework is conflict-explicit under σ if any two arguments of the framework which do not occur together in any σ-extension are explicitly conflicting, i.e. they are linked via the attack relation.

As a simple example consider the AF $F = (\{a, b, c, d\}, \{(a, b), (b, a), (a, c), (b, d)\})$ which has $\mathbb{S} = \mathit{stb}(F) = \{\{a, d\}, \{b, c\}\}$. Note that $(c, d) \notin \mathit{Pairs}_{\mathbb{S}}$ but $(c, d) \notin R$ as well as $(d, c) \notin R$. Thus F is not conflict-explicit under stable semantics. However, if we add attacks (c, d) or (d, c) we obtain an equivalent (under stable semantics) conflict-explicit (under stable semantics) AF.

Theorem 19. *For each compact* AF F *which is conflict-explicit under stb, it holds that* $\mathit{stb}(F)$ *is independent.*

Proof. Consider some $F \in \mathit{CAF}_{stb}$ which is conflict-explicit under *stb* and let $\mathbb{E} = \mathit{stb}(F)$. Observe that $\mathbb{E} \subseteq \mathbb{E}^+$. We have to show that there exists an antisymmetric exclusion-mapping $\mathfrak{R}_{\mathbb{S}}$ fulfilling the independence-condition from Definition 8. Let $\mathfrak{R}_{\mathbb{E}} = \{(b, a) \notin R \mid (a, b) \in R\}$ and consider the AF $F^s = (A_F, R_F \cup \mathfrak{R}_{\mathbb{E}})$ being the symmetric version of F. Now let $E \in \mathbb{E}^-$. Note that $E \in \mathit{cf}(F) = \mathit{cf}(F^s)$. But as $E \notin \mathbb{E}$ there must be some $t \in (A \setminus E)$ such that for all $e \in E$, $(e, t) \notin R_F$. For all such $e \in E$ with $(e, t) \notin \mathit{Pairs}_{\mathbb{E}}$ it holds, as F is conflict-explicit under *stb*, that $(t, e) \in R_F$, hence $(e, t) \in \mathfrak{R}_{\mathbb{E}}$, showing that $\mathfrak{R}_{\mathbb{E}}$ is an exclusion-mapping.

It remains to show that $\mathfrak{R}_{\mathbb{E}}$ is antisymmetric and $\forall E \in \mathbb{E} \forall a \in \mathit{Arg}_{\mathbb{S}} \setminus E : \exists e \in E : (e, a) \notin (\mathfrak{R}_{\mathbb{E}} \cup \mathit{Pairs}_{\mathbb{E}})$ holds. As some pair (b, a) is in $\mathfrak{R}_{\mathbb{E}}$ iff $(a, b) \in R$ and $(b, a) \notin R$, $\mathfrak{R}_{\mathbb{E}}$ is antisymmetric. Finally consider some $E \in \mathbb{E}$ and $a \in \mathit{Arg}_{\mathbb{S}} \setminus E$ and assume that $\forall e \in E : (e, a) \in \mathfrak{R}_{\mathbb{E}} \vee (e, a) \in \mathit{Pairs}_{\mathbb{E}}$. This means that $e \not\mapsto_F a$, a contradiction to E being a stable extension of F. □

Since our characterizations of signatures completely abstract away from the actual structure of AFs but only focus on the set of extensions, our problem would be solved if the following was true.

EC-Conjecture. For each AF $F = (A, R)$ there exists an AF $F' = (A, R')$ which is conflict-explicit under the stable semantics such that $\mathit{stb}(F) = \mathit{stb}(F')$.

Note that the EC-conjecture implies that for each compact AF, there exists a stable-equivalent conflict-explicit (under stable) AF.

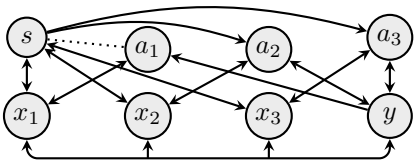


Figure 3: Orientation of non-explicit conflicts matters.

Theorem 20. *Under the assumption that the EC-conjecture holds,*

$$\Sigma^c_{stb} = \{\mathbb{S} \mid \mathbb{S} \subseteq \mathbb{S}^+ \wedge \mathbb{S} \textit{ is independent}\}.$$

Unfortunately, the question whether an equivalent conflict-explicit AF exists is not as simple as the example above suggests. We provide a few examples showing that proving the conjecture includes some subtle issues. Our first example shows that for adding missing attacks, the orientation of the attack needs to be carefully chosen.

Example 8. Consider the AF F in Figure 3 and observe $\mathit{stb}(F) = \{\{a_1, a_2, x_3\}, \{a_1, a_3, x_2\}, \{a_2, a_3, x_1\}, \{s, y\}\}$.
$\mathit{Pairs}_{\mathit{stb}(F)}$ yields one pair of arguments a_1 and s whose conflict is not explicit by F, i.e. $(a_1, s) \notin \mathit{Pairs}_{\mathit{stb}(F)}$, but $(a_1, s), (s, a_1) \notin R_F$. Now adding the attack $a_1 \mapsto_F s$ to F would reveal the additional stable extension $\{a_1, a_2, a_3\} \in (\mathit{stb}(F))^+$. On the other hand by adding the attack $s \mapsto_F a_1$ we get the conflict-explicit AF F' with $\mathit{stb}(F) = \mathit{stb}(F')$.

Finally recall the role of the arguments x_1, x_2, and x_3. Each of these arguments enforces exactly one extension (being itself part of it) by attacking (and being attacked by) all arguments not in this extension. We will make use of this construction-concept in Example 9.

Even worse, it is sometimes necessary to not only add the missing conflicts but also change the orientation of existing attacks such that the missing attack "fits well".

Example 9. Let $X = \{x_{s,t,i}, x_{s,u,i}, x_{t,u,i} \mid 1 \leq i \leq 3\} \cup \{x_{a,1,2}, x_{a,1,3}, x_{a,2,3}\}$ and $\mathbb{S} = \{\{s_i, t_i, x_{s,t,i}\}, \{s_i, u_i, x_{s,u,i}\}, \{t_i, u_i, x_{t,u,i}\} \mid i \in \{1, 2, 3\}\} \cup \{\{a_1, a_2, x_{a,1,2}\}, \{a_1, a_3, x_{a,1,3}\}, \{a_2, a_3, x_{a,2,3}\}\}$. Consider the AF $F = (A' \cup X, R' \cup \bigcup_{x \in X} \{(x, b), (b, x) \mid b \in (A' \setminus \mathbb{S}_x)\} \cup \{(x, x') \mid x, x' \in X, x \neq x'\})$, where the essential part (A', R') is depicted in Figure 4 and $\mathbb{S}_x$ is the unique set $X \in \mathbb{S}$ with $x \in X$. We have $\mathit{stb}(F) = \mathbb{S}$. Observe that F contains three non-explicit conflicts under the stable semantics, namely the argument-pairs (a_1, s_1), (a_2, s_2), and (a_3, s_3). Adding any of (s_i, a_i) to R_F would turn $\{s_i, t_i, u_i\}$ into a stable extension; adding all (a_i, s_i) to R_F would yield $\{a_1, a_2, a_3\}$ as additional stable extension. Hence there is no way of making the conflicts explicit without changing other parts of F and still getting a stable-equivalent AF. Still, we can realize $\mathit{stb}(F)$ by a compact and conflict-explicit AF, for example by $G = (A_F, (R_F \cup \{(a_1, s_1), (a_2, s_2), (a_3, s_3)\}) \setminus \{(a_1, x_{a,2,3}), (a_2, x_{a,1,3}), (a_3, x_{a,1,2})\})$.

This is another indicator, yet far from a proof, that the EC-conjecture holds and by that Theorem 20 describes the exact characterization of the c-signature under stable semantics.

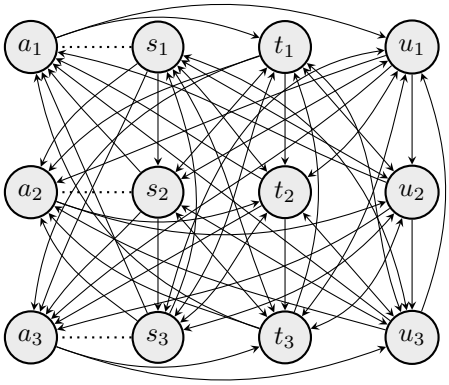


Figure 4: Guessing the orientation of non-explicit conflicts is not enough.

6 Discussion

We introduced and studied the novel class of σ-compact argumentation frameworks for σ among naive, stable, stage, semi-stable and preferred semantics. We provided the full relationships between these classes, and showed that the extension verification problem is still coNP-hard for stage, semi-stable and preferred semantics. We next addressed the question of compact realizability: Given a set of extensions, is there a compact AF with this set of extensions under semantics σ? Towards this end, we first used and extended recent results on maximal numbers of extensions to provide shortcuts for showing non-realizability. Lastly we studied signatures, sets of compactly realizable extension-sets, and provided sufficient conditions for compact realizability. This culminated in the explicit-conflict conjecture, a deep and interesting question in its own right: Given an AF, can all implicit conflicts be made explicit?

Our work bears considerable potential for further research. First and foremost, the explicit-conflict conjecture is an interesting research question. But the EC-conjecture (and compact AFs in general) should not be mistaken for a mere theoretical exercise. There is a fundamental computational significance to compactness: When searching for extensions, arguments span the search space, since extensions are to be found among the subsets of the set of all arguments. Hence the more arguments, the larger the search space. Compact AFs are argument-minimal since none of the arguments can be removed without changing the outcome, thus leading to a minimal search space. The explicit-conflict conjecture plays a further important role in this game: implicit conflicts are something that AF solvers have to deduce on their own, paying mostly with computation time. If there are no implicit conflicts in the sense that all of them have been made explicit, solvers have maximal information to guide search.

References

[Amgoud, Dimopoulos, & Moraitis 2008] Amgoud, L.; Dimopoulos, Y.; and Moraitis, P. 2008. Making decisions through preference-based argumentation. In *KR*, 113–123.

[Baroni & Giacomin 2008] Baroni, P., and Giacomin, M. 2008. A systematic classification of argumentation frameworks where semantics agree. In *COMMA*, volume 172 of *FAIA*, 37–48.

[Baroni, Caminada, & Giacomin 2011] Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *KER* 26(4):365–410.

[Baumann & Strass 2014] Baumann, R., and Strass, H. 2014. On the Maximal and Average Numbers of Stable Extensions. In *TAFA 2013*, volume 8306 of *LNAI*, 111–126.

[Bench-Capon & Dunne 2007] Bench-Capon, T. J. M., and Dunne, P. E. 2007. Argumentation in artificial intelligence. *AIJ* 171(10-15):619–641.

[Caminada, Carnielli, & Dunne 2012] Caminada, M.; Carnielli, W. A.; and Dunne, P. E. 2012. Semi-stable semantics. *JLC* 22(5):1207–1254.

[Coste-Marquis, Devred, & Marquis 2005] Coste-Marquis, S.; Devred, C.; and Marquis, P. 2005. Symmetric argumentation frameworks. In *ECSQARU*, volume 3571 of *Lecture Notes in Computer Science*, 317–328.

[Dimopoulos & Torres 1996] Dimopoulos, Y., and Torres, A. 1996. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science* 170(1-2):209–244.

[Dung 1995] Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AIJ* 77(2):321–357.

[Dunne & Bench-Capon 2002] Dunne, P. E., and Bench-Capon, T. J. M. 2002. Coherence in finite argument systems. *AIJ* 141(1/2):187–203.

[Dunne *et al.* 2013] Dunne, P. E.; Dvořák, W.; Linsbichler, T.; and Woltran, S. 2013. Characteristics of multiple viewpoints in abstract argumentation. In *Proc. DKB*, 16–30. Available under `http://www.dbai.tuwien.ac.at/staff/linsbich/pubs/dkb_2013.pdf`.

[Dunne *et al.* 2014] Dunne, P. E.; Dvořák, W.; Linsbichler, T.; and Woltran, S. 2014. Characteristics of multiple viewpoints in abstract argumentation. In *KR*.

[Dunne 2007] Dunne, P. E. 2007. Computational properties of argument systems satisfying graph-theoretic constraints. *AIJ* 171(10–15):701–729.

[Dvořák & Woltran 2011] Dvořák, W., and Woltran, S. 2011. On the intertranslatability of argumentation semantics. *JAIR* 41:445–475.

[Dvořák *et al.* 2014] Dvořák, W.; Järvisalo, M.; Wallner, J. P.; and Woltran, S. 2014. Complexity-sensitive decision procedures for abstract argumentation. *AIJ* 206:53–78.

[Griggs, Grinstead, & Guichard 1988] Griggs, J. R.; Grinstead, C. M.; and Guichard, D. R. 1988. The number of maximal independent sets in a connected graph. *Discrete Mathematics* 68(23):211–220.

[Jin & Li 2008] Jin, Z., and Li, X. 2008. Graphs with the second largest number of maximal independent sets. *Discrete Mathematics* 308(23):5864–5870.

[Moon & Moser 1965] Moon, J. W., and Moser, L. 1965. On cliques in graphs. *Israel Journal of Mathematics* 23–28.

[Rahwan & Simari 2009] Rahwan, I., and Simari, G. R., eds. 2009. *Argumentation in Artificial Intelligence*.

[Verheij 1996] Verheij, B. 1996. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In *Proc. NAIC*, 357–368.

Extension–based Semantics of Abstract Dialectical Frameworks

Sylwia Polberg
Vienna University of Technology
Institute of Information Systems
Favoritenstraße 9-11, 1040 Vienna, Austria *

Abstract

One of the most prominent tools for abstract argumentation is the Dung's framework, AF for short. It is accompanied by a variety of semantics including grounded, complete, preferred and stable. Although powerful, AFs have their shortcomings, which led to development of numerous enrichments. Among the most general ones are the abstract dialectical frameworks, also known as the ADFs. They make use of the so–called acceptance conditions to represent arbitrary relations. This level of abstraction brings not only new challenges, but also requires addressing existing problems in the field. One of the most controversial issues, recognized not only in argumentation, concerns the support cycles. In this paper we introduce a new method to ensure acyclicity of the chosen arguments and present a family of extension–based semantics built on it. We also continue our research on the semantics that permit cycles and fill in the gaps from the previous works. Moreover, we provide ADF versions of the properties known from the Dung setting. Finally, we also introduce a classification of the developed sub–semantics and relate them to the existing labeling–based approaches.

1 Introduction

Over the last years, argumentation has become an influential subfield of artificial intelligence (Rahwan and Simari 2009). One of its subareas is the *abstract argumentation*, which became especially popular thanks to the research of Phan Minh Dung (Dung 1995). Although the framework he has developed was relatively limited, as it took into account only the conflict relation between the arguments, it inspired a search for more general models (see (Brewka, Polberg, and Woltran 2013) for an overview). Among the most abstract enrichments are the abstract dialectical frameworks, ADFs for short (Brewka and Woltran 2010). They make use of the so–called acceptance conditions to express arbitrary interactions between the arguments. However, a framework cannot be considered a suitable argumentation tool without properly developed semantics.

The semantics of a framework are meant to represent what is considered rational. Given many of the advanced semantics, such as grounded or complete, we can observe that they return same results when faced with simple, tree–like frameworks. The differences between them become more visible when we work with more complicated cases. On various occasions examples were found for which none of the available semantics returned satisfactory answers. This gave rise to new concepts: for example, for handling indirect attacks and defenses we have prudent and careful semantics (Coste-Marquis, Devred, and Marquis 2005a; 2005b). For the problem of even and odd attack cycles we can resort to some of the SCC–recursive semantics (Baroni, Giacomin, and Guida 2005), while for treatment of self attackers, sustainable and tolerant semantics were developed (Bodanza and Tohmé 2009). Introducing a new type of relation, such as support, creates additional problems.

The most controversial issue in the bipolar setting concerns the support cycles and is handled differently from formalism to formalism. Among the best known structures are the Bipolar Argumentation Frameworks (BAFs for short) (Cayrol and Lagasquie-Schiex 2009; 2013), Argumentation Frameworks with Necessities (AFNs) (Nouioua 2013) and Evidential Argumentation Systems (EASs) (Oren and Norman 2008). While AFNs and EASs discard support cycles, BAFs do not make such restrictions. In ADFs cycles are permitted unless the intuition of a given semantics is clearly against it, for example in stable and grounded cases. This variety is not an error in any of the structures; it is caused by the fact that, in a setting that allows more types of relations, a standard Dung semantics can be extended in several ways. Moreover, since one can find arguments both for and against any of the cycle treatments, lack of consensus as to what approach is the best should not be surprising.

Many properties of the available semantics can be seen as "inside" ones, i.e. "what can I consider rational?". On the other hand, some can be understood as on the "outside", e.g. "what can be considered a valid attacker, what should I defend from?". Various examples of such behavior exist even in the Dung setting. An admissible extension is conflict–free and defends against attacks carried out by any other argument in the framework. We can then add new restrictions by saying that self–attackers are not rational. Consequently, we limit the set of arguments we have to protect our choice from. In a bipolar setting, we can again define admissibility in the basic manner. However, one often demands that the extension is free from support cy-

*The author is funded by the Vienna PhD School of Informatics. This research is a part of the project I1102 supported by the Austrian Science Fund FWF.

cles and that we only defend from acyclic arguments, thus again trimming the set of attackers. From this perspective semantics can be seen as a two–person discussion, describing what "I can claim" and "what my opponent can claim". This is also the point of view that we follow in this paper. Please note that this sort of dialogue perspective can already be found in argumentation (Dung and Thang 2009; Jakobovits and Vermeir 1999), although it is used in a slightly different context.

Although various extension–based semantics for ADFs have already been proposed in the original paper (Brewka and Woltran 2010), many of them were defined only for a particular ADF subclass called the bipolar and were not suitable for all types of situations. As a result, only three of them – conflict–free, model and grounded – remain. Moreover, the original formulations did not solve the problem of positive dependency cycles. Unfortunately, neither did the more recent work into labeling–based semantics (Brewka et al. 2013), even though they solve most of the problems of their predecessors. The aim of this paper is to address the issue of cycles and the lack of properly developed extension–based semantics. We introduce a family of such semantics and specialize them to handle the problem of support cycles, as their treatment seems to be the biggest difference among the available frameworks. Furthermore, a classification of our sub–semantics in the inside–outside fashion that we have described before is introduced. We also recall our previous research on admissibility in (Polberg, Wallner, and Woltran 2013) and show how it fits into the new system. Our results also include which known properties, such as Fundamental Lemma, carry over from the Dung framework. Finally, we provide an analysis of similarities and differences between the extension and labeling–based semantics in the context of produced extensions.

The paper is structured as follows. In Sections 2 to 4 we provide a background on argumentation frameworks. Then we introduce the new extension–based semantics and analyze their behavior in Section 5. We close the paper with a comparison between the new concepts and the existing labeling–based approach.

2 Dung's Argumentation Frameworks

Let us recall the abstract argumentation framework by Dung (Dung 1995) and its semantics. For more details we refer the reader to (Baroni, Caminada, and Giacomin 2011).

Definition 2.1. *A **Dung's abstract argumentation framework** (AF for short) is a pair (A, R), where A is a set of arguments and $R \subseteq A \times A$ represents an attack relation.*

Definition 2.2. *Let $AF = (A, R)$ be a Dung's framework. We say that an argument $a \in A$ is **defended**[1] by a set E in AF, if for each $b \in A$ s.t. $(b, a) \in R$, there exists $c \in E$ s.t. $(c, b) \in R$. A set $E \subseteq A$ is:*

- ***conflict–free** in AF iff for each $a, b \in E$, $(a, b) \notin R$.*
- ***admissible** iff conflict–free and defends all of its members.*
- ***preferred** iff it is maximal w.r.t set inclusion admissible.*
- ***complete** iff it is admissible and all arguments defended by E are in E.*
- ***stable** iff it is conflict–free and for each $a \in A \setminus E$ there exists an argument $b \in E$ s.t. $(b, a) \in R$.*

*The **characteristic function** $F_{AF} : 2^A \to 2^A$ is defined as: $F_{AF}(E) = \{a \mid a \text{ is defended by } E \text{ in } AF\}$. The **grounded extension** is the least fixed point of F_{AF}.*

In the context of this paper, we would also like to recall the notion of range:

Definition 2.3. *Let E^+ be the set of arguments attacked by E and E^- the set of arguments that attack E . $E^+ \cup E$ is the **range** of E.*

Please note the concepts E^+ and the E^- sets can be used to redefine defense. This idea will be partially used in creating the semantics of ADFs. Moreover, there is also an alternative way of computing the grounded extension:

Proposition 2.4. *The unique **grounded extension** of AF is defined as the outcome E of the following algorithm. Let us start with $E = \emptyset$:*

1. *put each argument $a \in A$ which is not attacked in AF into E; if no such argument exists, return E.*
2. *remove from AF all (new) arguments in E and all arguments attacked by them (together with all adjacent attacks) and continue with Step 1.*

What we have described above forms a family of the extension–based semantics. However, there exist also labeling–based ones (Caminada and Gabbay 2009; Baroni, Caminada, and Giacomin 2011). Instead of computing sets of accepted arguments, they generate labelings, i.e. total functions $Lab : A \to \{in, out, undec\}$. Although we will not recall them here, we would like to draw the attention to the fact that for every extension we can obtain an appropriate labeling and vice versa. This property is particularly important as it does not fully carry over to the ADF setting.

Finally, we would like to recall several important lemmas and theorems from the original paper on AFs (Dung 1995).

Lemma 2.5. ***Dung's Fundamental Lemma** Let E be an admissible extension, a and b two arguments defended by E. Then $E' = E \cup \{a\}$ is admissible and b is defended by E'.*

Theorem 2.6. *Every stable extension is a preferred extension, but not vice versa. Every preferred extension is a complete extension, but not vice versa. The grounded extension is the least w.r.t. set inclusion complete extension. The complete extensions form a complete semilattice w.r.t. set inclusion.* [2]

3 Argumentation Frameworks with Support

Currently the most recognized frameworks with support are the Bipolar Argumentation Framework BAF (Cayrol and Lagasquie-Schiex 2013), Argumentation Framework with Necessities AFN (Nouioua 2013) and Evidential Argumentation System EAS (Oren and Norman 2008). We will now

[1] Please note defense is often also termed acceptability, i.e. if a set defends an argument, the argument is acceptable w.r.t. this set.

[2] A partial order $(A, \leq)$ is a complete semilattice iff each nonempty subset of A has a glb and each increasing sequence of S has a lub.

briefly recall them in order to further motivate the directions of the semantics we have taken in ADFs.

The original bipolar argumentation framework BAF (Cayrol and Lagasquie-Schiex 2009) studied a relation we will refer to as abstract support:

Definition 3.1. *A **bipolar argumentation framework** is a tuple (A, R, S), where A is a set of **arguments**, $R \subseteq A \times A$ represents the **attack** relation and $S \subseteq A \times A$ the **support**.*

The biggest difference between this abstract relation and any other interpretation of support is the fact that it did not affect the acceptability of an argument, i.e. even a supported argument could be accepted "alone". The positive interaction was used to derive additional indirect forms of attack and based on them, stronger versions of conflict–freeness were developed.

Definition 3.2. *We say that an argument a **support attacks** argument b, if there exists some argument c s.t. there is a sequence of supports from a to c (i.e. $aS...Sc$) and cRb. We say that a **secondary attacks** b if there is some argument c s.t. $cS...Sb$ and aRc. We say that $B \subseteq A$ is:*

- ***+conflict–free** iff $\nexists a, b \in B$ s.t. a (directly or indirectly) attacks b.*
- ***safe** iff $\nexists b \in A$ s.t. b is at the same time (directly or indirectly) attacked by B and either there is a sequence of supports from an element of B to b, or $b \in B$.*
- ***closed under** S iff $\forall b \in B, a \in A$, if bSa then $a \in B$.*

The definition of defense remains the same and any Dung semantics is specialized by choosing an given notion of conflict–freeness or safety. Apart from the stable semantics, no assumptions as to cycles occurring in the support relation are made. The later developed deductive support (Boella et al. 2010) remains in the BAF setting and is also modeled by new indirect attacks (Cayrol and Lagasquie-Schiex 2013). Consequently, acyclicity is not required.

The most recent formulation of the framework with necessary support is as follows (Nouioua 2013):

Definition 3.3. *An **argumentation framework with necessities** is a tuple (A, R, N), where A is the set of **arguments**, $R \subseteq A \times A$ represents (binary) **attacks**, and $N \subseteq (2^A \setminus \emptyset) \times A$ is the **necessity relation**.*

Given a set $B \subseteq A$ and an argument a, BNa should be read as "at least one element of B needs to be present in order to accept a". The AFN semantics are built around the notions of coherence:

Definition 3.4. *We say that a set of arguments B is **coherent** iff every $b \in B$ is powerful, i.e. there exists a sequence $a_0, .., a_n$ of some elements of B s.t 1) $a_n = b$, 2) $\nexists C \subseteq A$ s.t. CNa_0, and 3) for $1 \leq i \leq n$ it holds that for every set $C \subseteq A$ if CNa_i, then $C \cap \{a_0, ..., a_{i-1}\} \neq \emptyset$. A coherent set B is **strongly coherent** iff it is conflict–free.*

Although it may look a bit complicated at first, the definition of coherence grasps the intuition that we need to provide sufficient acyclic support for the arguments we want to accept. Defense in AFNs is understood as the ability to provide support and to counter the attacks from any coherent set.

Definition 3.5. *We say that a set $B \subseteq A$ **defends** a, if $B \cup \{a\}$ is coherent and for every $c \in A$, if cRa then for every coherent set $C \subseteq A$ containing c, BRC.*

Using the notion of strong coherence and defense, the AFN semantics are built in a way corresponding to Dung semantics. It is easy to see that, through the notion of coherency, AFNs discard cyclic arguments both on the "inside" and the "outside". This means we cannot accept them in an extension and they are not considered as valid attackers.

The last type of support we will consider here is the the *evidential support* (Oren and Norman 2008). It distinguishes between standard and *prima facie* arguments. The latter are the only ones that are valid without any support. Every other argument that we want to accept needs to be supported by at least one prima facie argument, be it directly or not.

Definition 3.6. *An **evidential argumentation system** (EAS) is a tuple (A, R, E) where A is a set of **arguments**, $R \subseteq (2^A \setminus \emptyset) \times A$ is the **attack** relation, and $E \subseteq (2^A \setminus \emptyset) \times A$ is the **support** relation. We distinguish a special argument $\eta \in A$ s.t. $\nexists (x, y) \in R$ where $\eta \in x$; and $\nexists x$ where $(x, \eta) \in R$ or $(x, \eta) \in E$.*

η represents the prima facie arguments and is referred to as evidence or environment. The idea that the valid arguments (and attackers) need to trace back to it is captured with the notions of e–support and e–supported attack[3].

Definition 3.7. *An argument $a \in A$ has **evidential support** (e–support) from a set $S \subseteq A$ iff $a = \eta$ or there is a non-empty $S' \subseteq S$ s.t. $S'Ea$ and $\forall x \in S'$, x has e–support from $S \setminus \{a\}$.*

Definition 3.8. *A set $S \subseteq A$ carries out an **evidence supported attack** (e–supported attack) on a iff $(S', a) \in R$ where $S' \subseteq S$, and for all $s \in S'$, s has e–support from S. An e–supported attack by S on a is **minimal** iff there is no $S' \subset S$ that carries out an e–supported attack on a.*

The EASs semantics are built around the notion of acceptability in a manner similar to those of Dung's. However, in AFs only the attack relation was considered. In EASs, also sufficient support is required:

Definition 3.9. *An argument a is **acceptable** w.r.t. a set $S \subseteq A$ iff a is e–supported by S and given a minimal e–supported attack by a set $T \subseteq A$ against a, it is the case that S carries out an e–supported attack against a member of T.*

The notion of conflict–freeness is easily adapted to take set, not just binary conflict into account. With this and the notion of acceptability, the EASs semantics are built just like AF semantics. From the fact that every valid argument needs to be grounded in the environment it clearly results that EAS semantics are acyclic both on the inside and outside.

4 Abstract Dialectical Frameworks

Abstract dialectical frameworks have been defined in (Brewka and Woltran 2010) and further studied in (Brewka

[3]The presented definition is slightly different from the one available in (Oren and Norman 2008). The new version was obtained through personal communication with the author.

et al. 2013; Polberg, Wallner, and Woltran 2013; Strass 2013a; 2013b; Strass and Wallner 2014). The main goal of ADFs is to be able to express arbitrary relations and avoid the need of extending AFs by new relation sets each time they are needed. This is achieved by the means of the acceptance conditions, which define what arguments should be present in order to accept or reject a given argument.

Definition 4.1. *An **abstract dialectical framework** (ADF) as a tuple (S, L, C), where S is a set of abstract **arguments** (nodes, statements), $L \subseteq S \times S$ is a set of **links** (edges) and $C = \{C_s\}_{s \in S}$ is a set of **acceptance conditions**, one condition per each argument. An acceptance condition is a total function $C_s : 2^{par(s)} \to \{in, out\}$, where $par(s) = \{p \in S \mid (p, s) \in L\}$ is the set of parents of an argument s.*

One can also represent the acceptance conditions by propositional formulas (Ellmauthaler 2012) rather than functions. By this we mean that given an argument $s \in S$, $C_s = \varphi_s$, where φ_s is a propositional formula over arguments S. As we will be making use of both extension and labeling–based semantics, we need to provide necessary information on interpretations first (more details can be found in (Brewka et al. 2013; Polberg, Wallner, and Woltran 2013)). Please note that the links in ADFs only represent connections between arguments, while the burden of deciding the nature of these connections falls to the acceptance conditions. Moreover, parents of an argument can be easily extracted from the conditions. Thus, we will use of shortened notation $D = (S, C)$ through the rest of this paper.

Interpretations and decisiveness

A two (or three–valued) interpretation is simply a mapping that assigns the truth values $\{\mathbf{t}, \mathbf{f}\}$ (respectively $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$) to arguments. We will be making use both of partial (i.e. defined only for a subset of S) and the full ones. In the three–valued setting we will adopt the precision (information) ordering of the values: $\mathbf{u} \leq_i \mathbf{t}$ and $\mathbf{u} \leq_i \mathbf{f}$ The pair $(\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}, \leq_i)$ forms a complete meet–semilattice with the meet operation $\sqcap$ assigning values in the following way: $\mathbf{t} \sqcap \mathbf{t} = \mathbf{t}, \mathbf{f} \sqcap \mathbf{f} = \mathbf{f}$ and $\mathbf{u}$ in all other cases. It can naturally be extended to interpretations: given two interpretations v and v' on S, we say that v' contains more information, denoted $v \leq_i v'$, iff $\forall_{s \in S}\, v(s) \leq_i v'(s)$. Similar follows for the meet operation. In case v is three and v' two–valued, we say that v' extends v. This means that elements mapped originally to $\mathbf{u}$ are now assigned either $\mathbf{t}$ or $\mathbf{f}$. The set of all two–valued interpretations extending v is denoted $[v]_2$.

Example 4.2. *Let $v = \{a : \mathbf{t}, b : \mathbf{t}, c : \mathbf{f}, d : \mathbf{u}\}$ be a three–valued interpretation. We have two extending interpretations, $v' = \{a : \mathbf{t}, b : \mathbf{t}, c : \mathbf{f}, d : \mathbf{t}\}$ and $v'' = \{a : \mathbf{t}, b : \mathbf{t}, c : \mathbf{f}, d : \mathbf{f}\}$. Clearly, it holds that $v \leq_i v'$ and $v \leq_i v''$. However, v' and v'' are incomparable w.r.t. $\leq_i$.*

Let now $w = \{a : \mathbf{f}, b : \mathbf{f}, c : \mathbf{f}, d : \mathbf{t}\}$ be another three–valued interpretation. $v \sqcap w$ gives us a new interpretation $w' = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{f}, d : \mathbf{u}\}$: as the assignments of a, b and d differ between v and w, the resulting value is $\mathbf{u}$. On the other hand, c is in both cases $\mathbf{f}$ and thus retains its value.

We will use v^x to denote a set of arguments mapped to x by v, where x is some truth–value. Given an acceptance condition C_s for some argument $s \in S$ and an interpretation v, we define a shorthand $v(C_s)$ as $C_s(v^{\mathbf{t}} \cap par(s))$. For a given propositional formula φ and an interpretation v defined over all of the atoms of the formula, $v(\varphi)$ will just stand for the value of the formula under v. However, apart from knowing the "current" value of a given acceptance condition for some interpretation, we would also like to know if this interpretation is "final". By this we understand that no new information will cause the value to change. This is expressed by the notion of decisive interpretations, which are at the core of the extension–based ADF semantics.

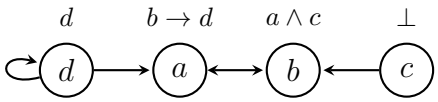


Figure 1: Sample ADF

Definition 4.3. *Given an interpretation v defined over a set A, **completion** of v to a set Z where $A \subseteq Z$ is an interpretation v' defined on Z in a way that $\forall a \in A\ v(a) = v'(a)$. By a $\mathbf{t}/\mathbf{f}$ completion we will understand v' that maps all arguments in $Z \setminus A$ respectively to $\mathbf{t}/\mathbf{f}$.*

The similarity between the concepts of completion and extending interpretation should not be overlooked. Basically, given a three–valued interpretation v defined over S, the set $[v]_2$ precisely corresponds to the set of completions to S of the two–valued part of v. However, the extension notion from the three–valued setting can be very misleading when used in the extension–based semantics. Therefore, we would like to keep the notion of completion.

Definition 4.4. *We say that a two–valued interpretation v is **decisive** for an argument $s \in S$ iff for any two completions $v_{par(s)}$ and $v'_{par(s)}$ of v to $A \cup par(s)$, it holds that $v_{par(s)}(C_s) = v'_{par(s)}(C_s)$. We say that s is **decisively out/in** wrt v if v is decisive and all of its completions evaluate C_s to respectively out, in.*

Example 4.5. *Let $(\{a, b, c, d\}, \{\varphi_a : b \to d, \varphi_b : a \wedge c, \varphi_c : \bot, \varphi_d : d\})$ be an ADF depicted in Figure 1. Example of a decisively in interpretation for a is $v = \{b : \mathbf{f}\}$. It simply means that knowing that b is false, not matter the value of d, the implication is always true and thus the acceptance condition satisfied. From the more technical side, it is the same as checking that both completions to $\{b, d\}$, namely $\{b : \mathbf{f}, d : \mathbf{t}\}$ and $\{b : \mathbf{f}, d : \mathbf{f}\}$ satisfy the condition. Example of a decisively out interpretation for b is $v' = \{c : \mathbf{f}\}$. Again, it suffices to falsify one element of a conjunction to know that the whole formula will evaluate to false.*

Acyclicity

Let us now focus on the issue of positive dependency cycles. Please note we refrain from calling them support cycles in the ADF setting in order not to confuse them with specific definitions of support available in the literature (Cayrol and Lagasquie-Schiex 2013).

Informally speaking, an argument takes part in a cycle if its acceptance depends on itself. An intuitive way of verifying the acyclicity of an argument would be to "track" its

evaluation, e.g. in order to accept a we need to accept b, to accept b we need to accept c and so on. This basic case becomes more complicated when disjunction is introduced. We then receive a number of such "paths", with only some of them proving to be acyclic. Moreover, they might be conflicting one with each other, and we can have a situation in which all acyclic evaluations are blocked and a cycle is forced. Our approach to acyclicity is based on the idea of such "paths" that are accompanied by sets of arguments used to detect possible conflicts.

Let us now introduce the formal definitions. Given an argument $s \in S$ and $x \in \{in, out\}$, by $min\_dec(x, s)$ we will denote the set of minimal two–valued interpretations that are decisively x for s. By minimal we understand that both $v^{\mathbf{t}}$ and $v^{\mathbf{f}}$ are minimal w.r.t. set inclusion.

Definition 4.6. *Let $A \subseteq S$ be a nonempty set of arguments. A **positive dependency function** on A is a function pd assigning every argument $a \in A$ an interpretation $v \in min\_dec(in, a)$ s.t. $v^t \subseteq A$ or $\mathcal{N}$ (null) iff no such interpretation can be found.*

Definition 4.7. *An **acyclic positive dependency evaluation** ace^a for $a \in A$ based on a given pd–function pd is a pair $((a_0, ..., a_n), B)$, [4] where $B = \bigcup_{i=0}^{n} pd(a_i)^{\mathbf{f}}$ and $(a_0, ..., a_n)$ is a sequence of distinct elements of A s.t.: 1) $\forall_{i=0}^{n} pd(a_i) \neq \mathcal{N}$, 2) $a_n = a$, 3) $pd(a_0)^{\mathbf{t}} = \emptyset$, and 4) $\forall_{i=1}^{n}$, $pd(a_i)^{\mathbf{t}} \subseteq \{a_0, ..., a_{i-1}\}$. We will refer to the sequence part of the evaluation as **pd–sequence** and to the B as the **blocking set**. We will say that an argument a is **pd–acyclic** in A iff there exist a pd–function on A and a corresponding acyclic pd–evaluation for a.*

We will write that an argument has an acyclic pd–evaluation on A if there is some pd–function on A from which we can produce the evaluation. There are two ways we can "attack" an acyclic evaluation. We can either discard an argument required by the evaluation or accept one that is capable of preventing it. This corresponds to rejecting a member of a pd–sequence or accepting an argument from the blocking set. We can now formulate this "conflict" by the means of an interpretation:

Definition 4.8. *Let $A \subseteq S$ be a set of arguments and $a \in A$ s.t. a has an acyclic pd–evaluation $ace^a = ((a_0, ..., a_n), B)$ in A. We say that a two–valued interpretation v **blocks** ace^a iff $\exists b \in B$ s.t. $v(b) = \mathbf{t}$ or $\exists a_i \in \{a_0, ..., a_n\}$ s.t. $v(a_i) = \mathbf{f}$.*

Let us now show on an example why we require minimality on the chosen interpretations and why do we store the blocking set:

Example 4.9. *Let us assume an ADF $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$ depicted in Figure 2. For argument a there exist the following decisively in interpretations: $v_1 = \{c : \mathbf{f}\}, v_2 = \{b : \mathbf{t}\}, v_3 = \{b : \mathbf{t}, c : \mathbf{f}\}, v_4 = \{b : \mathbf{t}, c : \mathbf{t}\}, v_5 = \{b : \mathbf{f}, c : \mathbf{f}\}$. Only the first two are minimal. Considering v_4 would give us a wrong view that a requires c for acceptance, which is not a desirable reading. The interpretations for b and c are respectively $w_1 = \{a : \mathbf{t}\}$ and $z_1 = \{c : \mathbf{t}\}$. Consequently, we have two pd–functions on $\{a, b, c\}$, namely $pd_1 = \{a : v_1, b : w_1, c : z_1\}$ and $pd_2 = \{a : v_2, b : w_1, c : z_1\}$. From them we obtain one acyclic pd–evaluation for a: $((a), \{c\})$, one for b: $((a, b), \{c\})$ and none for c.*

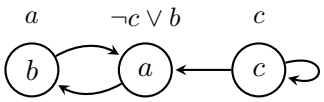


Figure 2: Sample ADF

Let us look closer at a set $E = \{a, b, c\}$. We can see that c is not pd–acyclic in E. However, the presence of c also "forces" a cycle between a and b. The acceptance conditions of all arguments are satisfied, thus this simple check is not good enough to verify if a cycle occurs. Only looking at the whole evaluations shows us that a and b are both blocked by c. Although a and b are pd–acyclic in E, we see that their evaluations are in fact blocked and this second level of conflict needs to be taken into account by the semantics.

As a final remark, please note that it can be the case that an evaluation is self–blocking. We can now proceed to recall existing and introduce new semantics of the abstract dialectical frameworks.

5 Extension–Based Semantics of ADFs

Although various semantics for ADFs have already been defined in the original paper (Brewka and Woltran 2010), only three of them – conflict–free, model and grounded (initially referred to as well–founded) – are still used (issues with the other formulations can be found in (Brewka et al. 2013; Polberg, Wallner, and Woltran 2013; Strass 2013a)). Moreover, the treatment of cycles and their handling by the semantics was not sufficiently developed. In this section we will address all of those issues. Before we continue, let us first motivate our choice on how to treat cycles. The opinions on support cycles differ between the available frameworks, as we have shown in Section 3. Therefore, we would like to explore the possible approaches in the context of ADFs by developing appropriate semantics.

The classification of the sub–semantics that we will adopt in this paper is based on the inside–outside intuition we presented in the introduction. Appropriate semantics will receive a two–element prefix $xy-$, where x will denote whether cycles are permitted or not on the "inside" and y on the "outside". We will use $x, y \in \{a, c\}$, where a will stand for *acyclic* and c for *cyclic* constraints. As the conflict–free (and naive) semantics focus only on what we can accept, we will drop the prefixing in this case. Although the model, stable and grounded fit into our classification (more details can be found in this section and in (Polberg 2014)), they have a sufficiently unique naming and further annotations are not necessary. We are thus left with admissible, preferred and complete. The BAF approach follows the idea that we can accept arguments that are not acyclic in our opinion and we allow our opponent to do the same. The ADF semantics we have developed in (Polberg, Wallner, and Woltran 2013) also shares this view. Therefore, they will receive the $cc-$ prefix. On the other hand, AFN and EAS semantics do

[4] Please note that it is not required that $B \subseteq A$

not permit cycles both in extensions and as attackers. Consequently, the semantics following this line of reasoning will be prefixed with $aa-$. Please note we believe that a non–uniform approach can also be suitable in certain situations. By non–uniform we mean not accepting cyclic arguments, but still treating them as valid attackers and so on (i.e. $ca-$ and $ac-$). However, in this paper we would like to focus only on the two perspectives mentioned before.

Conflict–free and naive semantics

In the Dung setting, conflict–freeness meant that the elements of an extension could not attack one another. Providing an argument with the required support is then a separate condition in frameworks such as AFNs and EASs. In ADFs, where we lose the set representation of relations in favor of abstraction, not including "attackers" and accepting "supporters" is combined into one notion. This represents the intuition of arguments that can stand together presented in (Baroni, Caminada, and Giacomin 2011). Let us now assume an ADF $D = (S, C)$.

Definition 5.1. *A set of arguments $E \subseteq S$ is **conflict–free** in D iff for all $s \in E$ we have $C_s(E \cap par(s)) = in$.*

In the acyclic version of conflict–freeness we also need to deal with the conflicts arising on the level of evaluations. To meet the formal requirements, we first have to show how the notions of range and the E^+ set are moved to ADFs.

Definition 5.2. *Let $E \subseteq S$ a conflict–free extension of D and v_E a partial two–valued interpretation built as follows:*

1. *Let $M = E$ and for every $a \in M$ set $v_E(a) = \mathbf{t}$;*
2. *For every argument $b \in S \setminus M$ that is decisively out in v_E, set $v_E(b) = \mathbf{f}$ and add b to M;*
3. *Repeat the previous step until there are no new elements added to M.*

*By E^+ we understand the set of arguments $v_E^{\mathbf{f}}$ and we will refer to it as the **discarded set**. v_E now forms the **range interpretation** of E.*

However, the notions of the discarded set and the range are quite strict in the sense that they require an explicit "attack" on arguments that take part in dependency cycles. This is not always a desirable property. Depending on the approach we might not treat cyclic arguments as valid and hence want them "out of the way".

Definition 5.3. *Let $E \subseteq S$ a conflict–free extension of D and v_E^a a partial two–valued interpretation built as follows:*

1. *Let $M = E$. For every $a \in M$ set $v_E^a(a) = \mathbf{t}$.*
2. *For every argument $b \in S \setminus M$ s.t. every acyclic pd–evaluation of b in S is blocked by v_E^a, set $v_E^a(b) = \mathbf{f}$ and add b to M.*
3. *Repeat the previous step until there are no new elements added to M.*

*By E^{a+} we understand the set of arguments mapped to $\mathbf{f}$ by v_E^a and refer to it as **acyclic discarded set**. We refer to v_E^a as **acyclic range interpretation** of E.*

We can now define an acyclic version of conflict–freeness:

Definition 5.4. *A conflict–free extension E is a **pd–acyclic conflict–free extension** of D iff every argument $a \in E$ has an unblocked acyclic pd–evaluation on E w.r.t. v^E.*

As we are dealing with a conflict– free extension, all the arguments of a given pd–sequence are naturally $\mathbf{t}$ both in v_E and v_E^a. Therefore, in order to ensure that an evaluation $((a_0, ..., a_n), B)$ is unblocked it suffices to check whether $E \cap B = \emptyset$. Consequently, in this case it does not matter w.r.t. to which version of range we are verifying the evaluations.

Definition 5.5. *The **naive** and **pd–acyclic naive** extensions are respectively maximal w.r.t. set inclusion conflict–free and pd–acyclic conflict–free extensions.*

Example 4.9 (Continued)**.** *Recall the ADF $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$. The conflict–free extensions are $\emptyset, \{a\}, \{c\}, \{a, b\}$ and $\{a, b, c\}$. Their standard discarded set in all cases is just $\emptyset$ – none of the sets has the power to decisively out the non–members. The acyclic discarded set of $\emptyset$, $\{a\}$ and $\{a, b\}$ is now $\{c\}$, since it has no acyclic evaluation to start with. In the case of $\{c\}$, it is $\{a, b\}$, which is to be expected since c had the power to block their evaluations. Finally, $\{a, b, c\}^{a+}$ is $\emptyset$. In the end, only $\emptyset, \{a\}$ and $\{a, b\}$ qualify for acyclic type. The naive and pd–acyclic naive extensions are respectively $\{a, b, c\}$ and $\{a, b\}$.*

Model and stable semantics

The concept of a model basically follows the intuition that if something can be accepted, it should be accepted:

Definition 5.6. *A conflict–free extension E is a **model** of D if $\forall\ s \in S,\ \ C_s(E \cap par(s)) = in$ implies $s \in E$.*

Although the semantics is simple, several of its properties should be explained. First of all, given a model candidate E, checking whether a condition of some argument s is satisfied does not verify if an argument depends on itself or if it "outs" a previously included member of E. This means that an argument we should include may break conflict–freeness of the set. On the other hand, an argument can be out due to positive dependency cycles, i.e. its supporter is not present. And since model makes no acyclicity assumptions on the inside, arguments outed this way can later appear in a model $E \subset E'$. Consequently, it is clear to see that model semantics is not universally defined and the produced extensions might not be maximal w.r.t. subset inclusion.

The model semantics was used as a mean to obtain the stable models. The main idea was to make sure that the model is acyclic. Unfortunately, the used reduction method was not adequate, as shown in (Brewka et al. 2013). However, the initial idea still holds and we use it to define stability. Although the produced extensions are now incomparable w.r.t. set inclusion, the semantics is still not universally defined.

Definition 5.7. *A model E is a **stable extension** iff it is pd–acyclic conflict–free.*

Example 4.9 (Continued)**.** *Let us again come back to the ADF $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$. The conflict–free extensions were $\emptyset, \{a\}, \{c\}, \{a, b\}$ and $\{a, b, c\}$. The first two are not models, as in the first case a and in the latter b can be accepted. Recall that $\emptyset, \{a\}$ and $\{a, b\}$ were*

the pd–acyclic conflict–free extensions. The only one that is also a model is $\{a, b\}$ and thus we obtain our single stable extension.

Grounded semantics

Next comes the grounded semantics (Brewka and Woltran 2010). Just like in the Dung setting, it preserves the unique–status property, i.e. produces only a single extension. Moreover, it is defined in the terms of a special operator:

Definition 5.8. *Let $\Gamma'_D(A, R) = (acc(A, R), reb(A, R))$, where $acc(A, R) = \{r \in S \mid A \subseteq S' \subseteq (S \backslash R) \Rightarrow C_r(S' \cap par(r)) = in\}$ and $reb(A, R) = \{r \in S \mid A \subseteq S' \subseteq (S \backslash R) \Rightarrow C_r(S' \cap par(r)) = out\}$. Then E is the **grounded model** of D iff for some $E' \subseteq S$, (E, E') is the least fix–point of Γ'_D.*

Although it might look complicated at first, this is nothing more than analyzing decisiveness using a set, not interpretation form (please see (Polberg 2014) for more details). Thus, one can also obtain the grounded extension by an ADF version of Proposition 2.4:

Proposition 5.9. *Let v be an empty interpretation. For every argument $a \in S$ that is decisively in w.r.t. v, set $v(a) = \mathbf{t}$ and for every argument $b \in S$ that is decisively w.r.t. v, set $v(b) = \mathbf{f}$. Repeat the procedure until no further assignments can be done. The **grounded extension** of D is then $v^{\mathbf{t}}$.*

Example 4.9 (Continued)**.** *Recall our ADF $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$. Let v be an empty interpretation. It is easy to see that no argument is decisively in/out w.r.t. v. If we analyze a, it is easy to see that if we accept c, the condition is out, but if we accept both b and c it is in again. Although both b and c are out in v, the condition of b can be met if we accept a, and condition of c if we accept c. Hence, we obtain no decisiveness again. Thus, $\emptyset$ is the grounded extension.*

Admissible and preferred semantics

In (Polberg, Wallner, and Woltran 2013) we have presented our first definition of admissibility, before the sub–semantics classification was developed. The new, simplified version of our previous formulation, is now as follows:

Definition 5.10. *A conflict–extension $E \subseteq S$ is **cc–admissible** in D iff every element of E is decisively in w.r.t to its range interpretation v_E.*

It is important to understand how decisiveness encapsulates the defense known from the Dung setting. If an argument is decisively in, then any set of arguments that would have the power to out the acceptance condition is "prevented" by the interpretation. Hence, the statements required for the acceptance of a are mapped to $\mathbf{t}$ and those that would make us reject a are mapped to $\mathbf{f}$. The former encapsulates the required support, while the latter contains the "attackers" known from the Dung setting.

When working with the semantics that have to be acyclic on the "inside", we not only have to defend the members, but also their acyclic evaluations:

Definition 5.11. *A pd–acyclic conflict–free extension E is **aa–admissible** iff every argument in E 1) is decisively in w.r.t. acyclic range interpretation v^a_E, and 2) has an unblocked acyclic pd–evaluation on E s.t. all members of its blocking set B are mapped to $\mathbf{f}$ by v^a_E.*

Definition 5.12. *A set of arguments is **xy–preferred** iff it is maximal w.r.t. set inclusion xy–admissible.*

The following example shows that decisiveness encapsulates defense of an argument, but not necessarily of its evaluation:

Example 5.13. *Let us modify the ADF depicted in Figure 2 by changing the condition of c: $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : \top\})$. The new pd–evaluations are $((a), \{c\})$ for a, $((a, b), \{c\})$ for b and $((c), \emptyset)$ for c. The conflict–free extensions are now $\emptyset, \{a\}, \{c\}, \{a, b\}$ and $\{a, b, c\}$. Apart from the last, all are pd–acyclic conflict–free. $\emptyset$ and $\{c\}$ are trivially both aa and cc–admissible and $\{a, b, c\}$ cc–admissible. The standard and acyclic discarded sets of $\{a\}$ are both empty, thus a is not decisively in (we can always utter c) and the set is neither aa nor cc–admissible. The discarded sets of $\{a, b\}$ are also empty; however, it is easy to see that both a and b are decisively in. Although uttering c would not change the values of acceptance conditions, it blocks the pd–evaluations of a and b. Thus, $\{a, b\}$ is cc, but not aa–admissible. The cc and aa–preferred extensions are respectively $\{a, b, c\}$ and $\{c\}$.*

Example 4.9 (Continued)**.** *Let us come back to the original ADF $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$. $\emptyset, \{a\}, \{c\}, \{a, b\}$ and $\{a, b, c\}$ were the standard and $\emptyset, \{a\}, \{a, b\}$ pd–acyclic conflict–free extensions. $\emptyset$ is trivially both aa and cc, while $\{c\}$ and $\{a, b, c\}$ cc–admissible. The standard discarded sets of $\{a\}$ and $\{a, b\}$ are both empty, while the acyclic ones are $\{c\}$. Consequently, $\{a\}$ is aa, but not cc–admissible. $\{a, b\}$ is both, but for different reasons; in the cc–case, all arguments are decisively in (due to cyclic defense). In aa–approach, they are again decisively in, but the evaluations are "safe" only because c is not considered a valid attacker.*

Complete semantics

Completeness represents an approach in which we have to accept everything we can safely conclude from our opinions. In the Dung setting, "safely" means defense, while in the bipolar setting it is strengthened by providing sufficient support. In a sense, it follows the model intuition that what we can accept, we should accept. However, now we not only use an admissible base in place of a conflict–free one, but also defend the arguments in question. Therefore, instead of checking if an argument is in, we want it to be decisively in.

Definition 5.14. *A cc–admissible extension E is **cc–complete** in D iff every argument in S that is decisively in w.r.t. to range interpretation v_E is in E.*

Definition 5.15. *An aa–admissible extension E is **aa–complete** in D iff every argument in S that is decisively in w.r.t. to acyclic range interpretation v^a_E is in E.*

Please note that in the case of aa–complete semantics, no further "defense" of the evaluation is needed, as visible in

AA Fundamental Lemma (i.e. Lemma 5.17). This comes from the fact that if we already have a properly "protected" evaluation, then appending a decisively in argument to it is sufficient for creating an evaluation for this argument.

Example 4.9 (Continued)**.** *Let us now finish with the ADF* $(\{a, b, c\}, \{C_a : \neg c \vee b, C_b : a, C_c : c\})$*. It is easy to see that all cc–admissible extensions are also cc–complete. However, only* $\{a, b\}$ *is aa–complete. Due to the fact that* c *is trivially included in any discarded set,* a *can always be accepted (thus,* $\emptyset$ *is disqualified). Then, from acceptance of* a*, acceptance of* b *follows easily and* $\{a\}$ *is disqualified.*

Properties and examples

Although the study provided here will by not be exhaustive, we would like to show how the lemmas and theorems from the original paper on AFs (Dung 1995) are shifted into this new setting. The proofs can be found in (Polberg 2014).

Even though every pd–acyclic conflict–free extension is also conflict–free, it does not mean that every aa–admissible is cc–admissible. These approaches differ significantly. The first one makes additional restrictions on the "inside", but due to acyclicity requirements on the "outside" there are less arguments a given extension has to defend from. The latter allows more freedom as to what we can accept, but also gives this freedom to the opponent, thus there are more possible attackers. Moreover, it should not come as a surprise that these differences pass over to the preferred and complete semantics, as visible in Example 5.19. Our results show that admissible sub–semantics satisfy the Fundamental Lemma.

Lemma 5.16. *CC Fundamental Lemma: Let* E *be a cc–admissible extension,* v_E *its range interpretation and* $a, b \in S$ *two arguments decisively in w.r.t.* v_E*. Then* $E' = E \cup \{a\}$ *is cc–admissible and* b *is decisively in w.r.t.* v'_E*.*

Lemma 5.17. *AA Fundamental Lemma: Let* E *be an aa-admissible extension,* v^a_E *its acyclic range interpretation and* $a, b \in S$ *two arguments decisively in w.r.t.* v^a_E*. Then* $E' = E \cup \{a\}$ *is aa–admissible and* b *is decisively in w.r.t.* v'_E*.*

The relations between the semantics presented in (Dung 1995) are preserved by some of the specializations:

Theorem 5.18. *Every stable extension is an aa–preferred extension, but not vice versa. Every xy–preferred extension is an xy–complete extension for* $x, y \in \{a, c\}$*, but not vice versa. The grounded extension might not be an aa–complete extension. The grounded extension is the least w.r.t. set inclusion cc–complete extension.*

Example 5.19. *Let* $(\{a, b, c, d\}, \{C_a : \neg b, C_b : \neg a, C_c : b \wedge \neg d, C_d : d\})$ *be the ADF depicted in Figure 3. The obtained extensions are visible in Table 1. The conflict–free, model, stable, grounded, admissible, complete and preferred semantics will be abbreviated to CF, MOD, STB, GRD, ADM, COMP and PREF. The prefixing is visible in second column. In case of conflict–freeness,* C *will denote the standard, and* A *the pd–acyclic one.*

6 Labeling–Based Semantics of ADFs

The two approaches towards labeling–based semantics of ADFs were developed in (Strass 2013a; Brewka et al. 2013).

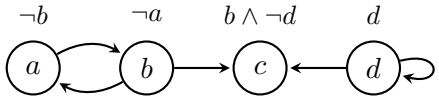


Figure 3: Sample ADF

Table 1: Extensions of the ADF from Figure 3.

CF	C	$\emptyset$, $\{a\}$, $\{b\}$, $\{d\}$, $\{b, c\}$, $\{a, d\}$, $\{b, d\}$
	A	$\emptyset$, $\{a\}$, $\{b\}$, ~~$\{d\}$~~, $\{b, c\}$, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~
MOD		~~$\emptyset$~~, $\{a\}$, ~~$\{b\}$~~, ~~$\{d\}$~~, $\{b, c\}$, $\{a, d\}$, $\{b, d\}$
STB		~~$\emptyset$~~, $\{a\}$, ~~$\{b\}$~~, ~~$\{d\}$~~, $\{b, c\}$, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~
GRD		$\emptyset$, ~~$\{a\}$~~, ~~$\{b\}$~~, ~~$\{d\}$~~, ~~$\{b, c\}$~~, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~
ADM	CC	$\emptyset$, $\{a\}$, $\{b\}$, $\{d\}$, ~~$\{b, c\}$~~, $\{a, d\}$, $\{b, d\}$
	AA	$\emptyset$, $\{a\}$, $\{b\}$, ~~$\{d\}$~~, $\{b, c\}$, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~
COMP	CC	$\emptyset$, $\{a\}$, $\{b\}$, $\{d\}$, ~~$\{b, c\}$~~, $\{a, d\}$, $\{b, d\}$
	AA	$\emptyset$, $\{a\}$, ~~$\{b\}$~~, ~~$\{d\}$~~, $\{b, c\}$, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~
PREF	CC	~~$\emptyset$~~, ~~$\{a\}$~~, ~~$\{b\}$~~, ~~$\{d\}$~~, ~~$\{b, c\}$~~, $\{a, d\}$, $\{b, d\}$
	AA	~~$\emptyset$~~, $\{a\}$, ~~$\{b\}$~~, ~~$\{d\}$~~, $\{b, c\}$, ~~$\{a, d\}$~~, ~~$\{b, d\}$~~

We will focus on the latter one, based on the notion of a three–valued characteristic operator:

Definition 6.1. *Let* V_S *be the set of all three–valued interpretations defined on* S*,* s *and argument in* S *and* v *an interpretation in* V_S*. The* ***three–valued characteristic operator*** *of* D *is a function* $\Gamma_D : V_S \to V_S$ *s.t.* $\Gamma_D(v) = v'$ *with* $v'(s) = \bigsqcap_{w \in [v]_2} C_s(par(s) \cap w^{\mathbf{t}})$*.*

Verifying the value of an acceptance condition under a set of extensions$[v]_2$ of a three–valued interpretation v is exactly checking its value in the completions of the two–valued part of v. Thus, an argument that is $\mathbf{t}$/$\mathbf{f}$ in $\Gamma_D(v)$ is decisively in/out w.r.t. to the two–valued part of v.

It is easy to see that in a certain sense this operator allows self–justification and self–falsification, i.e. that status of an argument depends on itself. Take, for example, a self–supporter; if we generate an interpretation in which it is false then, obviously, it will remain false. Same follows if we assume it to be true. This results from the fact that the operator functions on interpretations defined on all arguments, thus allowing a self–dependent argument to affect its status.

The labeling–based semantics are now as follows:

Definition 6.2. *Let* v *be a three–valued interpretation for* D *and* Γ_D *its characteristic operator. We say that* v *is:*

- ***three–valued model*** *iff for all* $s \in S$ *we have that* $v(s) \neq \mathbf{u}$ *implies that* $v(s) = v(\varphi_s)$*;*
- ***admissible*** *iff* $v \leq_i \Gamma_D(v)$*;*
- ***complete*** *iff* $v = \Gamma_D(v)$*;*
- ***preferred*** *iff it is* $\leq_i$*–maximal admissible;*
- ***grounded*** *iff it is the least fixpoint of* Γ_D*.*

Although in the case of stable semantics we formally receive a set, not an interpretation, this difference is not significant. As nothing is left undecided, we can safely map all remaining arguments to $\mathbf{f}$. The current state of the art definition (Strass 2013a; Brewka et al. 2013) is as follows:

Definition 6.3. *Let M be a model of D. A **reduct** of D w.r.t. M is $D^M = (M, L^M, C^M)$, where $L^M = L \cap (M \times M)$ and for $m \in M$ we set $C_m^M = \varphi_m[b/\mathbf{f} : b \notin M]$. Let gv be the grounded model of D^M. Model M is **stable** iff $M = gv^{\mathbf{t}}$.*

Example 5.19 (Continued)**.** *Let us now compute the possible labelings of our ADF. As there are over twenty possible three–valued models, we will not list them. We have in total 15 admissible interpretations: $v_1 = \{a : \mathbf{f}, b : \mathbf{t}, c : \mathbf{u}, d : \mathbf{t}\}, v_2 = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{u}, d : \mathbf{u}\}, v_3 = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{u}, d : \mathbf{t}\}, v_4 = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{u}, d : \mathbf{t}\}, v_5 = \{a : \mathbf{f}, b : \mathbf{t}, c : \mathbf{u}, d : \mathbf{f}\}, v_6 = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{u}, d : \mathbf{f}\}, v_7 = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{u}, d : \mathbf{u}\}, v_8 = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{f}, d : \mathbf{t}\}, v_9 = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{f}, d : \mathbf{t}\}, v_{10} = \{a : \mathbf{f}, b : \mathbf{t}, c : \mathbf{t}, d : \mathbf{f}\}, v_{11} = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{u}, d : \mathbf{f}\}, v_{12} = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{f}, d : \mathbf{u}\}, v_{13} = \{a : \mathbf{f}, b : \mathbf{t}, c : \mathbf{u}, d : \mathbf{u}\}, v_{14} = \{a : \mathbf{f}, b : \mathbf{t}, c : \mathbf{f}, d : \mathbf{t}\}$ and $v_{15} = \{a : \mathbf{t}, b : \mathbf{f}, c : \mathbf{f}, d : \mathbf{f}\}$. Out of them v_7 to v_{15} are complete. The ones that maximize the information content in this case are the ones without any $\mathbf{u}$ mappings: v_9, v_{10}, v_{14} and v_9. v_{10} and v_{15} are stable and finally, v_7 is grounded.*

Comparison with the extension–based approach

We will start the comparison of extensions and labelings by relating conflict–freeness and three–valued models. Please note that the intuitions of two–valued and three–valued models are completely different and should not be confused. We will say that an extension E and a labeling v correspond iff $v^{\mathbf{t}} = E$.

Theorem 6.4. *Let E be a conflict–free and A a pd–acyclic conflict–free extension. The $\mathbf{u}$–completions of v_E, v_A and v_A^a are three–valued models.*

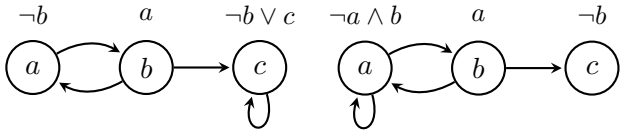
Let us continue with the admissible semantics. First, we will tie the notion of decisiveness to admissibility, following the comparison of completions and extending interpretations that we have presented in Section 4.

Theorem 6.5. *Let v be a three–valued interpretation and v' its (maximal) two–valued sub–interpretation. v is admissible iff all arguments mapped to $\mathbf{t}$ are decisively in w.r.t. v' and all arguments mapped to $\mathbf{f}$ are decisively out w.r.t. v'.*

Please note that this result does not imply that admissible extensions and labelings "perfectly" coincide. In labelings, we guess an interpretation, and thus assign initial values to arguments that we want to verify later. If they are self–dependent, it of course affects the outcome. In the extension based approaches, we distinguish whether this dependency is permitted. Therefore, the aa– and cc– approaches will have a corresponding labeling, but not vice versa.

Theorem 6.6. *Let E be a cc–admissible and A an aa–admissible extension. The $\mathbf{u}$–completions of v_E and v_A^a are admissible labelings.*

Let us now consider the preferred semantics. Information maximality is not the same as maximizing the set of accepted arguments and due to the behavior of Γ_D we can obtain a preferred interpretation that can map to $\mathbf{t}$ a subset of arguments of another interpretation. Consequently, we fail to receive an exact correspondence between the semantics. By this we mean that given a framework there can exist an (arbitrary) preferred extension without a labeling counterpart and a labeling without an appropriate extension of a given type.



(a) ADF_1 (b) ADF_2

Figure 4: Sample ADFs

Theorem 6.7. *For any xy–preferred extension there might not exist a corresponding preferred labeling and vice versa.*

Example 6.8. *Let us look at $ADF_1 = (\{a, b, c\}, \{C_a : \neg a, C_b : a, C_c : \neg b \vee c\})$, as depicted in Figure 4a. a and b cannot form a conflict–free extension to start with, so we are only left with c. However, the attack from b on c can be only overpowered by self–support, thus it cannot be part of an aa–admissible extension. Therefore, we obtain only one aa–preferred extension, namely the empty set. The single preferred labeling solution would be $v = \{a : \mathbf{u}, b : \mathbf{u}, c : \mathbf{t}\}$ and we can see there is no correspondence between the results. On the other hand, there is one with the cc–preferred extension $\{c\}$.*

Finally, we have $ADF_2 = (\{a, b, c\}, \{C_a : \neg a \wedge b, C_b : a, C_c : \neg b\})$ depicted in Figure 4b. The preferred labeling is $\{a : \mathbf{f}, b : \mathbf{f}, c : \mathbf{t}\}$. The single cc–preferred extension is $\emptyset$ and again, we receive no correspondence. However, it is compliance with the aa–preferred extension $\{c\}$.

The labeling–based complete semantics can also be defined in terms of decisiveness:

Theorem 6.9. *Let v be a three–valued interpretation and v' its (maximal) two–valued sub–interpretation. v is complete iff all arguments decisively out w.r.t. v' are mapped to $\mathbf{f}$ by v and all arguments decisively in w.r.t. v' are mapped to $\mathbf{t}$ by v.*

Fortunately, just like in the case of admissible semantics, complete extensions and labelings partially correspond:

Theorem 6.10. *Let E be a cc–complete and A an aa–complete extension. The $\mathbf{u}$–completions of v_E and v_A^a are complete labelings.*

Please recall that in the Dung setting, extensions and labelings agreed on the sets of accepted arguments. In ADFs, this relation is often only one way – like in the case of admissible and complete cc– and aa– sub–semantics – or simply nonexistent, like in preferred approach. In this context, the labeling–based admissibility (and completeness) can be seen as the most general one. This does not mean that specializations, especially handling cycles, are not needed. Even more so, as to the best of our knowledge no methods for ensuring acyclicity in a three–valued setting are yet available.

Due to the fact that the grounded semantics has a very clear meaning, it is no wonder that both available approaches coincide, as already noted in (Brewka et al. 2013). We conclude this section by relating both available notions of stability. The relevant proofs can be found in (Polberg 2014).

Theorem 6.11. *The two–valued grounded extension and the grounded labeling correspond.*

Theorem 6.12. *A set $M \subseteq S$ of arguments is labeling stable iff it is extension–based stable.*

7 Concluding Remarks

In this paper we have introduced a family of extension–based semantics as well as their classification w.r.t. positive dependency cycles. Our results also show that they satisfy ADF versions of Dung's Fundamental Lemma and that appropriate sub–semantics preserve the relations between stable, preferred and complete semantics. We have also explained how our formulations relate to the labeling–based approach. Our results show that the precise correspondence between the extension–based and labeling–based semantics, that holds in the Dung setting, does not fully carry over.

It is easy to see that in a certain sense, labelings provide more information than extensions due to distinguishing false and undecided states. Therefore, one of the aims of our future work is to present the sub–semantics described here also in a labeling form. However, since our focus is primarily on accepting arguments, a comparison w.r.t. information content would not be fully adequate for our purposes and the current characteristic operator could not be fully reused. We hope that further research will produce satisfactory formulations.

References

Baroni, P.; Caminada, M.; and Giacomin, M. 2011. An introduction to argumentation semantics. *Knowledge Eng. Review* 26(4):365–410.

Baroni, P.; Giacomin, M.; and Guida, G. 2005. SCC-Recursiveness: A general schema for argumentation semantics. *Artif. Intell.* 168(1-2):162–210.

Bodanza, G. A., and Tohmé, F. A. 2009. Two approaches to the problems of self-attacking arguments and general odd-length cycles of attack. *Journal of Applied Logic* 7(4):403 – 420. Special Issue: Formal Models of Belief Change in Rational Agents.

Boella, G.; Gabbay, D.; van der Torre, L.; and Villata, S. 2010. Support in abstract argumentation. In *Proc. of COMMA 2010*, 111–122. Amsterdam, The Netherlands, The Netherlands: IOS Press.

Brewka, G., and Woltran, S. 2010. Abstract dialectical frameworks. In *Proc. KR '10*, 102–111. AAAI Press.

Brewka, G.; Ellmauthaler, S.; Strass, H.; Wallner, J. P.; and Woltran, S. 2013. Abstract dialectical frameworks revisited. In *Proc. IJCAI'13*, 803–809. AAAI Press.

Brewka, G.; Polberg, S.; and Woltran, S. 2013. Generalizations of Dung frameworks and their role in formal argumentation. *Intelligent Systems, IEEE* PP(99). Forthcoming.

Caminada, M., and Gabbay, D. M. 2009. A logical account of formal argumentation. *Studia Logica* 93(2):109–145.

Cayrol, C., and Lagasquie-Schiex, M.-C. 2009. Bipolar abstract argumentation systems. In Simari, G., and Rahwan, I., eds., *Argumentation in Artificial Intelligence*. 65–84.

Cayrol, C., and Lagasquie-Schiex, M.-C. 2013. Bipolarity in argumentation graphs: Towards a better understanding. *Int. J. Approx. Reasoning* 54(7):876–899.

Coste-Marquis, S.; Devred, C.; and Marquis, P. 2005a. Inference from controversial arguments. In Sutcliffe, G., and Voronkov, A., eds., *Proc. LPAR '05*, volume 3835 of *LNCS*, 606–620. Springer Berlin Heidelberg.

Coste-Marquis, S.; Devred, C.; and Marquis, P. 2005b. Prudent semantics for argumentation frameworks. In *Proc. of ICTAI'05*, 568–572. Washington, DC, USA: IEEE Computer Society.

Dung, P. M., and Thang, P. M. 2009. A unified framework for representation and development of dialectical proof procedures in argumentation. In *Proc. of IJCAI'09*, 746–751. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77:321–357.

Ellmauthaler, S. 2012. Abstract dialectical frameworks: properties, complexity, and implementation. Master's thesis, Faculty of Informatics, Institute of Information Systems, Vienna University of Technology.

Jakobovits, H., and Vermeir, D. 1999. Dialectic semantics for argumentation frameworks. In *Proc. of ICAIL '99*, 53–62. New York, NY, USA: ACM.

Nouioua, F. 2013. AFs with necessities: Further semantics and labelling characterization. In Liu, W.; Subrahmanian, V.; and Wijsen, J., eds., *Proc. SUM '13*, volume 8078 of *LNCS*. Springer Berlin Heidelberg. 120–133.

Oren, N., and Norman, T. J. 2008. Semantics for evidence-based argumentation. In *Proc. COMMA '08*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, 276–284. IOS Press.

Polberg, S.; Wallner, J. P.; and Woltran, S. 2013. Admissibility in the abstract dialectical framework. In *Proc. CLIMA'13*, volume 8143 of *LNCS*, 102–118. Springer.

Polberg, S. 2014. Extension–based semantics of abstract dialectical frameworks. Technical Report DBAI-TR-2014-85, Institute for Information Systems, Technical University of Vienna.

Rahwan, I., and Simari, G. R. 2009. *Argumentation in Artificial Intelligence*. Springer, 1st edition.

Strass, H., and Wallner, J. P. 2014. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. In *Proc. KR '14*. Forthcoming.

Strass, H. 2013a. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence* 205:39 – 70.

Strass, H. 2013b. Instantiating knowledge bases in abstract dialectical frameworks. In *Proc. CLIMA'13*, volume 8143 of *LNCS*, 86–101. Springer.

Credulous and Skeptical Argument Games for Complete Semantics in Conflict Resolution based Argumentation *

Jozef Frtús
Department of Applied Informatics
Faculty of Mathematics, Physics, and Informatics
Comenius University in Bratislava, Slovakia

Abstract

Argumentation is one of the most popular approaches of defining a non-monotonic formalism and several argumentation based semantics were proposed for defeasible logic programs. Recently, a new approach based on notions of conflict resolutions was proposed, however with declarative semantics only. This paper gives a more procedural counterpart by developing skeptical and credulous argument games for complete semantics and soundness and completeness theorems for both games are provided. After that, distribution of defeasible logic program into several contexts is investigated and both argument games are adapted for multi-context system.

Introduction

Argumentation is successfully applied as an approach of defining non-monotonic formalisms. The main advantage of semantics based on formal models of argumentation is its closeness to real humans discussions. Therefore, the semantics can be explained also for people not trained in formal logic or mathematics.

To capture the knowledge, a logical language is needed. Usually the language of Defeasible Logic Programming (DeLP) is considered, where two kinds for rules are distinguished. Strict rules represent deductive reasoning: whenever their preconditions hold, we accept the conclusion. On the other hand, defeasible rules formalize tentative knowledge that can be defeated. Several semantics based on argumentation were proposed for defeasible logic programs (Prakken and Sartor 1997), (García and Simari 2004), (Caminada and Amgoud 2007), (Prakken 2010), (Modgil and Prakken 2011), (Baláž, Frtús, and Homola 2013). However, as Caminada and Amgoud (Caminada and Amgoud 2007) pointed out, careless design of semantics may lead to very unintuitive results, such as inconsistency of the system (justification for both an atom A and its negation $\neg A$ is provided) or unsatisfying of strict rules (system justifies all preconditions, but not the conclusion of a strict rule).

*This work is supported from the VEGA project no. 1/1333/12.

In this paper we take the approach by Baláž et al. (Baláž, Frtús, and Homola 2013) as the starting point, since it both respects intuitions of logic programming and satisfies desired semantical properties. In (Baláž, Frtús, and Homola 2013) notion of conflict resolutions and new methodology of justification of arguments is introduced, however only in a declarative way. Our main goal, in this paper, is to give a more procedural counterpart. This is especially useful when dealing with algorithms and implementations. We adapt skeptical and credulous argument games for complete semantics and prove soundness and completeness for both of them, what is the main contribution of this paper. Then we are investigating with distribution of defeasible logic program into several contexts (programs) and both argument games are adapted for distributed computing. This can be useful in ambient intelligence environments, where distributed and contextual defeasible reasoning is heavily applied.

The paper is structured as follows: first preliminaries of Dung's abstract argumentation frameworks and defeasible logic programming are introduced. Then the declarative conflict resolution based semantics introduced in (Baláž, Frtús, and Homola 2013) is recapitulated. Argument games are developed and their properties are proved in the next section. The last section is devoted to contextualization of defeasible logic programs.

Preliminaries

Argumentation Framework

Definition 1 (Abstract Argumentation Framework (Dung 1995))**.** An *abstract argumentation framework* is a pair $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ where

1. $\mathcal{A}$ is a set of *arguments*, and
2. $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is an *attack* relation on $\mathcal{A}$.

An argument A *attacks* an argument B if $(A, B) \in \mathcal{R}$. A set of arguments S *attacks* an argument A if an argument in S attacks A. A set of arguments S is *attack-free*[1] if S does not attack an argument in S.

[1]Note that we will use the original term "conflict-free" in slightly different context.

A set of arguments S *defends* an argument A if each argument attacking A is attacked by S. An attack-free set of arguments S is *admissible* iff S defends each argument in S. The *characteristic function* F_{AF} of an argumentation framework $AF = (\mathcal{A}, Def)$ is a mapping $F_{AF}\colon 2^{\mathcal{A}} \mapsto 2^{\mathcal{A}}$ where for all $S \subseteq \mathcal{A}$, $F_{AF}(S)$ is defined as $\{a \in \mathcal{A} \mid S \text{ defends } a\}$.

Definition 2 (Extension (Dung 1995))**.** An admissible set of arguments S is

1. a *complete extension* iff S contains each argument defended by S.
2. the *grounded extension* iff S is the least complete extension.
3. a *preferred extension* iff S is a maximal complete extension.
4. a *stable extension* iff S attacks each argument which does not belong to S.

We will prove following lemma[2], which will be used in procedural formalization of the grounded semantics. Its intuitive meaning is that an argument x to be in the grounded extension, it can not be defended only by itself.

Lemma 1. *Given an argumentation framework $(\mathcal{A}, Def)$ and a finite ordinal i, argument $A \in F^{i+1}$ iff for each argument Y defeating A, there is an argument $Z \in F^i$ such that $(Z, Y) \in Def$ and $Z \neq A$.*

Defeasible Logic Program

An *atom* is a propositional variable. A *classical literal* is either an atom or an atom preceded by classical negation $\neg$. A *default literal* is a classical literal preceded by default negation $\sim$. A *literal* is either a classical or a default literal. By definition $\neg\neg A$ equals to A and $\sim\sim L$ equals to L, for an atom A and a classical literal L. By $\mathcal{D}$ we will denote the set of all default literals. By convention $\sim S$ equals to $\{\sim L \mid L \in S\}$ for any set of literals S.

A *strict rule* is an expression of the form $L_1, \ldots, L_n \rightarrow L_0$ where $0 \leq n$, each L_i, $1 \leq i \leq n$, is a literal, and L_0 is a classical literal. A *defeasible rule* is an expression of the form $L_1, \ldots, L_n \Rightarrow L_0$ where $0 \leq n$, each L_i, $1 \leq i \leq n$, is a literal, and L_0 is a classical literal. A *defeasible logic program* $\mathcal{P}$ is a finite set of of strict rules Π and defeasible rules Δ. In the following text we use the symbol $\rightsquigarrow$ to denote either strict or defeasible rule.

Conflict Resolution based Semantics

Existing argumentation formalisms (Prakken 2010; García and Simari 2004; Prakken and Sartor 1997) are usually defined through five steps. At the beginning, some underlying logical *language* is chosen for describing knowledge. The notion of an *argument* is then defined within this language. Then *conflicts* between arguments are identified. The resolution of conflicts is captured by an *attack* relation among conflicting arguments. The *status* of an argument is then determined by the attack relation.

The conflict resolution based approach (Baláž, Frtús, and Homola 2013) diverge from this methodology. Instead of attacking a conflicting argument, one of the weaker building blocks (called vulnerabilities) used to construct the argument is attacked. Specifically, the resolution of a conflict is either a default assumption or a defeasible rule. The status of an argument does not depend on attack relation between arguments but on attack relation between conflict resolutions.

Conflict resolution based semantics for the DeLP consists of five steps:

1. Construction of arguments on top of the language of defeasible logic programs.
2. Identification of conflicts between arguments.
3. Proposing a conflict resolution strategy.
4. Instantiation of Dung's AFs with conflict resolutions.
5. Determination of the status of default assumptions, defeasible rules, and arguments with respect to successful conflict resolutions.

A vulnerability is a part of an argument that may be defeated to resolve a conflict. It is either a defeasible rule or a default literal.

Definition 3 (Vulnerability)**.** Let $\mathcal{P}$ be a defeasible logic program. A *vulnerability* is a defeasible rule in $\mathcal{P}$ or a default literal in $\mathcal{D}$. By $\mathcal{V}_{\mathcal{P}}$ we will denote the set of all vulnerabilities of $\mathcal{P}$.

Two kinds of arguments are usually be constructed in the language of defeasible logic programs. Default arguments correspond to default literals. Deductive arguments are constructed by chaining of rules. The following is a slightly more general definition, where a knowledge base $\mathcal{K}$ denotes literals for which no further backing is needed.

Definition 4 (Argument)**.** Let $\mathcal{P} = (\Pi, \Delta)$ be a defeasible logic program. An argument A for a literal L over a knowledge base $\mathcal{K}$ is

1. $[L]$, where $L \in \mathcal{K}$

$$\textsc{Conc}(A) = L$$
$$\textsc{Vuls}(A) = \{L\} \cap \mathcal{D}$$

2. $[A_1, \ldots, A_n \rightsquigarrow L]$ where each A_i, $0 \leq i \leq n$, is an argument for a literal L_i, $r\colon L_1, \ldots, L_n \rightsquigarrow L$ is a rule in $\mathcal{P}$.

$$\textsc{Conc}(A) = L$$
$$\textsc{Vuls}(A) = \textsc{Vuls}(A_1) \cup \cdots \cup \textsc{Vuls}(A_n) \cup (\{r\} \cap \Delta)$$

By $\mathcal{A}_{\mathcal{P}}$ we will denote the set of all arguments of $\mathcal{P}$.

[2]Note that all proofs are presented in the extended version of the paper available at `http://dai.fmph.uniba.sk/~frtus/nmr2014.pdf`

The typical example of knowledge base within the language of defeasible logic programming is the set of default literals $\mathcal{D}$ and we will not specify $\mathcal{K}$ until the section about contextual DeLP. Therefore, whenever the $\mathcal{K}$ is left unspecified, it is implicitly set to $\mathcal{D}$. Arguments created by chaining of rules will be called *deductive*.

Example 1. Consider the following defeasible logic program $\mathcal{P}$:

$$\begin{array}{rcl@{\qquad}rcl} \Rightarrow & a & & \Rightarrow & c & \\ \Rightarrow & b & & \Rightarrow & d & \\ a, b & \rightarrow & h & c, d & \rightarrow & \neg\, h \end{array}$$

Six deductive arguments can be constructed from $\mathcal{P}$

$$\begin{array}{rcl@{\qquad}rcl} A_1 & = & [\Rightarrow a] & A_4 & = & [\Rightarrow c] \\ A_2 & = & [\Rightarrow b] & A_5 & = & [\Rightarrow d] \\ A_3 & = & [A_1, A_2 \rightarrow h] & A_6 & = & [A_3, A_4 \rightarrow \neg h] \end{array}$$

Vulnerabilities of arguments A_3 are A_6 are $\textsc{Vuls}(A_3) = \{\Rightarrow a, \Rightarrow b\}$ and $\textsc{Vuls}(A_6) = \{\Rightarrow c, \Rightarrow d\}$.

Two kinds of conflicts among arguments may arise, each corresponds to one type of negation.

Definition 5 (Conflict)**.** Let $\mathcal{P}$ be a defeasible logic program. Arguments $A, B \in \mathcal{A}_\mathcal{P}$ are *conflicting* iff A rebuts or undercuts B where

1. A *rebuts* B iff A and B are deductive arguments and $\textsc{Conc}(A) = \neg\, \textsc{Conc}(B)$,
2. A *undercuts* B iff A is a deductive argument, B is a default argument, and $\textsc{Conc}(A) = \sim \textsc{Conc}(B)$.

The set $C = \{A, B\}$ is called a *conflict*. The first kind is called a *rebutting* conflict and the second kind is called an *undercutting* conflict. By $\mathcal{C}_\mathcal{P}$ we will denote the set of all conflicts of $\mathcal{P}$.

Conflicts are resolved by defeating one of the building blocks of conflicting arguments. Each default assumption or defeasible rule used to construct a conflicting argument is a possible resolution. Strict rules can not be used as a resolution of any conflict because they have to be always satisfied.

Definition 6 (Conflict Resolution)**.** Let $\mathcal{P}$ be a defeasible logic program. A vulnerability $V \in \mathcal{V}_\mathcal{P}$ is a *resolution of* a conflict $C \in \mathcal{C}_\mathcal{P}$ if $V \in \textsc{Vuls}(C)$. The pair $R = (C, V)$ is called a *conflict resolution*. By $\mathcal{R}_\mathcal{P}$ we will denote the set of all conflict resolutions of $\mathcal{P}$.

In general, each conflict may have more resolutions. Some of them may be more preferred than others. The choice of preferred conflict resolutions is always domain dependent. Some vulnerabilities can be defeated in one domain, but they may as well stay undefeated in another. Therefore we allow the user to choose any conflict resolution strategy she might prefer.

Definition 7 (Conflict Resolution Strategy)**.** Let $\mathcal{P}$ be a defeasible logic program. A *conflict resolution strategy* is a finite subset σ of $\mathcal{R}_\mathcal{P}$. We say that a vulnerability $V \in \mathcal{V}_\mathcal{P}$ is a σ-resolution of a conflict $C \in \mathcal{C}_\mathcal{P}$ if $(C, V) \in \sigma$. A conflict resolution strategy σ is *total* iff for each conflict $C \in \mathcal{C}_\mathcal{P}$ there exists a σ-resolution of C.

In existing approaches various conflict resolution strategies are applied. Examples of default, last-link and weakest-link conflict resolution strategies are presented in (Baláž, Frtús, and Homola 2013).

Example 2 (Continuation of Example 1). The only conflict in the defeasible logic program $\mathcal{P}$ is the $C = \{A_3, A_6\}$. Consider following six conflict resolutions.

$$\begin{array}{rcl@{\qquad}rcl} R_1 & = & (C, \Rightarrow a) & R_3 & = & (C, \Rightarrow c) \\ R_2 & = & (C, \Rightarrow b) & R_4 & = & (C, \Rightarrow d) \end{array}$$

Then $\sigma = \{R_1\}$, $\sigma' = \{R_i \mid 1 \leq i \leq 4\}$, $\sigma'' = \emptyset$ are examples of conflict resolution strategies for $\mathcal{P}$. We can see that strategies σ, σ' are total.

To determine in which way conflicts will be resolved, Dung's AF is instantiated with conflict resolutions. The intuitive meaning of a conflict resolution (C, V) is "the conflict C will be resolved by defeating the vulnerability V". The conflict resolution based semantics is built on three levels of attacks: attacks on the vulnerabilities, attacks on the arguments, and attacks on the conflict resolutions. Such an approach is necessary: if a vulnerability is defeated, so should be all arguments built on it, and consequently all conflict resolutions respective to the argument.

Definition 8 (Attack)**.** A conflict resolution $R = (C, V)$ *attacks*

- a vulnerability V' iff $V' = V$.
- an argument A iff R attacks a vulnerability in $\textsc{Vuls}(A)$.
- a conflict resolution $R' = (C', V')$ iff either
  1. $V \neq V'$ and R attacks an argument in C' or
  2. $V = V'$ and R attacks all arguments in C'.

A set of conflict resolutions $S \subseteq \mathcal{R}_\mathcal{P}$ attacks a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \mathcal{R}_\mathcal{P}$) iff a conflict resolution in S attacks V (resp. A or R).

Intuitively, it should not happen that both a conflict resolution $R = (C, V)$ and a vulnerability V are accepted. Therefore, if R is accepted, V and all arguments constructed on top of it should be defeated. The notion of attack between conflict resolutions formalizes the ideas that there may be more alternatives how to resolve a conflict and a conflict resolution may resolve other conflicts as well, thus causing other conflict resolutions to be irrelevant. The distinction between two kinds of attacks between conflict resolutions is necessary to achieve the intended semantics when dealing with self-conflicting arguments. The interested reader is kindly referred to (Baláž, Frtús, and Homola 2013) for demonstrative examples.

Definition 9 (Instantiation)**.** The instantiation for a conflict resolution strategy σ is an abstract argumentation framework $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ where

- $\mathcal{A} = \sigma$
- $\mathcal{R}$ is the attack relation on σ from the Definition 8.

Now thanks to the instantiation we can use the Dung's semantics in order to compute which vulnerabilities (resp. arguments, conflict resolutions) are undefeated (status IN), defeated (status OUT), or undecided (status UNDEC).

Definition 10 (Defense)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$. A set of conflict resolutions $S \subseteq \sigma$ *defends* a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \sigma$) iff each conflict resolution in σ attacking V (resp. A or R) is attacked by S.

Definition 11 (Status)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. The *status of* a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \sigma$) with respect to $\mathcal{E}$ is

- IN if $\mathcal{E}$ defends V (resp. A or R),
- OUT if V (resp. A or R) is attacked by $\mathcal{E}$,
- UNDEC otherwise.

Let $s \in \{\text{IN}, \text{UNDEC}, \text{OUT}\}$. By $\mathcal{A}^s_\mathcal{P}(\mathcal{E})$ we denote the set of all arguments with the status s with respect to a complete extension $\mathcal{E}$.

The following definitions define actual semantics of the DeLP program $\mathcal{P}$ and entailment relation between a program $\mathcal{P}$ and a literal L.

Definition 12 (Output)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. The *output* of $\mathcal{E}$ is a set of literals $\text{OUTPUT}_\mathcal{P}(\mathcal{E}) = \{L \in \mathcal{L} \mid \mathcal{A}^{\text{IN}}_\mathcal{P}(\mathcal{E})$ contains an argument for $L\}$.

Note that we will omit default literals in output to improve the legibility.

Definition 13 (Entailment)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{F}$ be the instantiation for σ. Defeasible logic program $\mathcal{P}$ *skeptically* (resp. *credulously*) *entails* a literal L, $\mathcal{P} \models_{sk} L$ (resp. $\mathcal{P} \models_{cr} L$) iff for each (resp. at least one) complete extension $\mathcal{E}$ of $\mathcal{F}$, $L \in \text{OUTPUT}_\mathcal{P}(\mathcal{E})$.

Example 3 (Continuation of Example 2). Consider the conflict resolution strategy σ' from Example 2. The instantiation for σ' is on the Figure 1.

$$\begin{array}{ccc} R_1 & \leftrightarrow & R_2 \\ \updownarrow & \nearrow\!\!\searrow & \updownarrow \\ R_4 & \leftrightarrow & R_3 \end{array}$$

Figure 1: The instantiation for the conflict resolution strategy σ'.

All conflict resolutions are now exclusive, since to resolve the conflict, it is sufficient to reject only one of the defeasible rules. Therefore σ' induces the complete graph.

There are five complete extensions $\{R_1\}$, $\{R_2\}$, $\{R_3\}$, $\{R_4\}$, $\{\}$ of the instantiation and each of them determine one program output $\{b, c, d, \neg h\}$, $\{a, c, d, \neg h\}$, $\{a, b, d, h\}$, $\{a, b, c, h\}$, $\{\}$.

Procedural Semantics

In the previous section we recapitulated (Baláž, Frtús, and Homola 2013) conflict resolution based semantics in the original declarative way. Although this declarative approach is very elegant and provides nice algebraic investigations, the more procedural style of semantics is appropriate when dealing with algorithms and implementations. One can see a parallel in a mathematical logic where we are similarly interested in a logical calculi (proof theory) which is sound and complete with respect to defined model-theoretic semantics. In this section our goal is to define skeptical and credulous argument games for complete semantics.

For a conflict resolution $R = (\{A, B\}, V)$ we define auxiliary functions which will be frequently used.

$$\begin{aligned} \mathit{con}(R) &= \{A, B\} \\ \mathit{res}(R) &= V \\ \mathit{vuls}(R) &= (\text{VULS}(A) \setminus \{V\}) \cup (\text{VULS}(B) \setminus \{V\}) \cup \\ &\quad (\text{VULS}(A) \cap \text{VULS}(B) \cap \{V\}) \end{aligned}$$

$\mathit{con}(R)$ denotes the conflict and $\mathit{res}(R)$ the resolution of a conflict resolution R. The meaning of the set of vulnerabilities $\mathit{vuls}(R)$ can be explained as following: suppose R is in a conflict resolution strategy σ and $\mathcal{E}$ is a complete extension of instantiation for σ. If $R \in \mathcal{E}$ and all the vulnerabilities in $\mathit{vuls}(R)$ have the status IN, then in order to resolve the conflict $\mathit{con}(R)$, the status of the vulnerability $\mathit{res}(R)$ is OUT.

Now we characterize the attack between conflict resolutions in terms of aforementioned functions. This will be useful in proofs for soundness and completeness of argument games.

Proposition 1. *Let $\mathcal{P}$ be a defeasible logic program, σ a conflict resolution strategy and $R = (C, V)$, $R' = (C', V') \in \sigma$ are conflict resolutions. Then R attacks R' iff $\mathit{res}(R) \in \mathit{vuls}(R')$.*

Argumentation can be seen and thus also formalized as a discussion of two players. The aim of the first player (called *proponent* PRO) is to prove an initial argument. The second player is an *opponent* (OPP), what means that her goal is to prevent proponent to prove the initial argument. Hence a dispute essentially is a sequence of moves where each player gives a counterargument to the last stated.

Proof theory of argumentation is well studied area and argument games for various semantics were proposed (Modgil and Caminada 2009), (Prakken and Sartor 1997). The process of proving a literal L via an argument game, in conflict resolution based setting, considered in this paper, takes two steps:

1. Find an argument A with conclusion L.
2. Justify all vulnerabilities in $\text{VULS}(A)$.

Intuitively, a move $(pl, R, \mathcal{V})$ is a triple denoting: player pl claims that the set of vulnerabilities $\mathcal{V}$ is true and resolution R is a reason for the other player why her set of vulnerabilities is not justified.

Definition 14 (Move)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$. A *move* is a triple $\mu = (pl, R, \mathcal{V})$, where $pl \in \{\text{OPP}, \text{PRO}\}$ denotes the player, $R \in \sigma$ is a resolution and $\mathcal{V} \subseteq \mathcal{V}_{\mathcal{P}}$ is a set of vulnerabilities.

Now since the very first move in a dialogue does not counter argue any of the previous move, the resolution R will be left unspecified and in such case we will write $(pl, -, \mathcal{V})$. Convention $\overline{\text{PRO}} = \text{OPP}$ and $\overline{\text{OPP}} = \text{PRO}$ will be used for denoting the opposite players. We say that a move $(pl, R, \mathcal{V})$ attacks a move $(\overline{pl}, R', \mathcal{V}')$ iff $res(R) \in \mathcal{V}'$.

Definition 15 (Argument Dialogue)**.** A *dialogue* is a finite nonempty sequence of moves $\mu_1, \ldots, \mu_n$, $1 \leq i < n$ where:

- $pl_i = \text{PRO}$ (OPP) iff i is odd (even)
- μ_{i+1} attacks μ_i

Intuitively, for a given argument, there can be more than one counterargument. This leads to a tree representation of discussion. Now, since the burden of proof is on the player PRO, proponent proves an initial argument if she wins all disputes. On the other hand, the burden of attack is on the player OPP, meaning that opponent must "play" all possible counterarguments, against PRO's last argument, forming new branches in a discussion tree.

Definition 16 (Argument Game)**.** Let σ be a conflict resolution strategy for a defeasible logic program P. An *argument game for an argument* A is a finite tree such that:

- $(\text{PRO}, -, \text{VULS}(A))$ is the root,
- all branches are dialogues,
- if move μ played by PRO is a node in the tree, then every move $(\text{OPP}, R, vuls(R))$ defeating μ is a child of μ.
- if μ, μ' are any moves played by PRO in T then μ does not defeat μ'.

A player wins a dispute if the counterpart can not make any move (give a counterargument). This can roughly be paraphrased as "the one who has the last word laughs best". Since the burden of proof is on the proponent, PRO, in order to win, has to win all branches in the game. On the other hand, for opponent to win an argument game, it is sufficient to win at least one branch of the game.

Definition 17 (Winner)**.** A player *pl wins a dialogue* iff she plays the last move in it. Player PRO (resp. OPP) *wins an argument game* T iff she wins all (resp. at least on of the) branches in the argument game T. An argument game is *successful* iff it is won by PRO.

Definition 18 (Proved Literal)**.** Let σ be a conflict resolution strategy for a defeasible logic program P. A literal L is :

- *proved in an argument game* T iff T is a successful argument game for an argument A with $\text{CONC}(A) = L$.
- *proved* iff there is an argument game T proving L.

Now we propose two particular argument games and prove their soundness and completeness with respect to declarative semantics defined in the previous section.

Argument Game for Skeptical Complete Semantics

First we will investigate with skeptical complete semantics which corresponds to the grounded semantics. Since the grounded semantics gives the highest burden of proof on membership of the extension it defines, the opponent is allowed to repeat her moves and proponent is not.

Definition 19 (Skeptical Game)**.** An argument game T is called *skeptical* iff in each branch of T holds: if $(\text{PRO}, R, \mathcal{V})$, $(\text{PRO}, R', \mathcal{V}')$ are two moves played by PRO, then $R \neq R'$.

Argument game for skeptical complete semantics is sound and complete with respect to declarative conflict resolution based grounded semantics.

Proposition 2. *Let P be a defeasible logic program and L be a literal. $P \models_{sk} L$ iff L is skeptically proved*[3]*.*

Let demonstrate the skeptical argument game in example.

Example 4. Consider the following defeasible logic program $\mathcal{P} = \{\Rightarrow a, \Rightarrow \neg a\}$ with conflict resolution strategy $\sigma = \{R_1, R_2\}$. There are two deductive arguments A_1, A_2, one conflict C and two conflict resolutions R_1, R_2.

$$\begin{array}{rclcrcl} A_1 & = & [\Rightarrow a] & & A_2 & = & [\Rightarrow \neg a] \\ C & = & \{A_1, A_2\} & & & & \\ R_1 & = & (C, \Rightarrow a) & & R_2 & = & (C, \Rightarrow \neg a) \end{array}$$

We would like to skeptically prove literal a. The skeptical argument game for argument A_1 is on the Figure 2.

Proponent cannot repeat her move μ_3 and therefore she loses the game.

Argument Game for Credulous Complete Semantics

Credulous complete semantics corresponds to the preferred semantics, where an argument can be defended by itself. Therefore, in credulous game, proponent is allowed to repeat her moves and opponent is not.

[3] A *literal L is skeptically proved* iff there is an skeptical argument game T such that L is proved in T.

$$\mu_1 = (\text{PRO}, -, \{\Rightarrow a\})$$
$$|$$
$$\mu_2 = (\text{OPP}, R_1, \{\Rightarrow \neg a\})$$
$$|$$
$$\mu_3 = (\text{PRO}, R_2, \{\Rightarrow a\})$$
$$|$$
$$\mu_4 = (\text{OPP}, R_1, \{\Rightarrow \neg a\})$$

Figure 2: The skeptical argument game for argument A_1.

Definition 20 (Credulous Game)**.** An argument game T is called *credulous* iff in each branch of T holds: if $(\text{OPP}, R, \mathcal{V})$, $(\text{OPP}, R', \mathcal{V}')$ are two moves played by OPP, then $R \neq R'$.

Argument game for credulous complete semantics is sound and complete with respect to declarative conflict resolution based preferred semantics.

Proposition 3. *Let $\mathcal{P}$ be a defeasible logic program and L be a literal. $\mathcal{P} \models_{cr} L$ iff L is credulously proved*[4]*.*

Now we will consider the defeasible logic program $\mathcal{P}$ and conflict resolution strategy σ from Example 4 and try to prove literal a credulously.

Example 5 (Continuation of Example 4). We would like to credulously prove literal a. The credulous argument game for argument A_1 is on the Figure 3.

$$\mu_1 = (\text{PRO}, -, \{\Rightarrow a\})$$
$$|$$
$$\mu_2 = (\text{OPP}, R_1, \{\Rightarrow \neg a\})$$
$$|$$
$$\mu_3 = (\text{PRO}, R_2, \{\Rightarrow a\})$$

Figure 3: The credulous argument game for argument A_1.

Opponent cannot repeat her move μ_2 and therefore the game is successful.

In (Governatori et al. 2004; Billington et al. 2010) several variants of defeasible logics with procedural semantics are proposed. Repeating an argument for PRO in our approach corresponds to the Δ proof tag of (Billington et al. 2010) and repeating an argument by OPP in our approach corresponds to the σ proof tag of (Billington et al. 2010).

Contextual DeLP

In the previous section we developed a procedural semantics based on argument games, now we will generalize these ideas to a distributive setting, where not only one, but the whole set of defeasible logic programs is assumed. Each of these programs may be viewed as a context (i.e. agent), which describes the world within its own language (i.e. propositional symbols). Contexts are interconnected into multi-context system through non-monotonic bridge rules, which import knowledge (foreign literals) from other contexts.

Our goal is to adapt the argument games to multi-context systems and satisfy following requirements:

- To minimize the necessary communication complexity between contexts. The conflict between arguments can be decided in other context, but the structure of arguments should not be communicated.
- Contexts provide just distributive computing, they should not change the semantics. Hence if we look at multi-context system as a monolithic program, the output should be the same as in distributive case.

Note that the distributed reasoning is a very complex task involving also issues of communication protocols and information security. In this chapter we abstract from this and focus only on the reasoning part.

Distributed computing of semantics is a hot topic in the area of multi-agent systems, for García and Simari's (García and Simari 2004) DeLP a distributed argumentation framework was proposed in (Thimm and Kern-Isberner 2008). Contextual defeasible reasoning is also applied in environment of Ambient Intelligence (Bikakis and Antoniou 2010), where devices, software agents and services are supposed to integrate and cooperate in support of human objectives.

A vocabulary V is a set of propositional variables. We say that a literal is *local* if its propositional variable is in V, otherwise it is *foreign*. A *local rule* contains only local literals. A *mapping rule* contains local literal in the head and at least one foreign literal in the body. A *contextual defeasible logic program* is a set of local strict rules, and local or mapping defeasible rules.

Sometimes we will denote the context pertaining to a foreign literal. For example 2: $a, c \Rightarrow b$ means that foreign literal a is imported from the second context.

Definition 21 (Context)**.** A *context* is a triple $C = (V, P, \sigma)$ where V is a set of propositional variables, P is a contextual defeasible logic program and σ is a conflict resolution strategy.

Since, within the one context we do not know the structure of an argument supporting some foreign literal, foreign literals cannot be used as resolutions of conflicts (their set of vulnerabilities is empty).

Contextual argument is an argument, where some of the literals (foreign) do not need a further backing and are considered as an import of the knowledge from the other context.

Definition 22 (Contextual Argument)**.** Given a context $C = (V, P, \sigma)$ and the set of foreign literals F, a *contextual argument* is an argument over a knowledge base $\sim V \cup F$. The set of all foreign literals contained

[4] A *literal L is credulously proved* iff there is an credulous argument game T such that L is proved in T.

by an argument in a set of arguments $\mathcal{A}$ will be denoted $F(\mathcal{A})$.

Contextual argument is *foreign* if it is of the form $[L]$, where L is a foreign literal.

Following proposition means that foreign literals cannot incorporate a conflict.

Proposition 4. *Given a context $C = (V, P, \sigma)$ and the set of foreign literals F, a foreign argument A cannot be in conflict with by any contextual argument from context C.*

Definition 23 (Multi-Context System)**.** A *multi-context system* [5] is a finite nonempty set of contexts $\mathcal{C} = \{C_1, \ldots, C_n\}$ where $0 < n$, each $C_i = (V_i, P_i, \sigma_i)$, $1 \leq i \leq n$, is a context and $\{V_1, \ldots, V_n\}$ is a partition of the set of all propositional variables in $\bigcup_{i=1}^{n} P_i$.

A multi context system $\mathcal{C}$ is *cyclic* iff there are contexts $C_1, C_2, \ldots, C_n$, $n \geq 2$ such that context C_i, $1 \leq i < n$, contains a mapping rule with a foreign literal from the context C_{i+1} and C_n, contains a mapping rule with a foreign literal from the context C_1. A multi context system is *acyclic* iff it is not cyclic.

Sometimes it is useful to look at a multi-context system as a monolithic defeasible logic program and vice versa. We say that a multi-context system $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a *contextualization* of a defeasible logic program $\mathcal{P}$ and conflict resolution strategy σ iff $\mathcal{P} = \bigcup_{i=1}^{n} P_i$ and $\sigma = \bigcup_{i=1}^{n} \sigma_i$. The idea of contextualization of a program or an argument is illustrated in the following example.

Example 6. Consider the following multi-context system consisting of two contexts

$C_1 = (\{a, d, h\}, P_1, \sigma_1)$	$C_2 = (\{b, c\}, P_2, \sigma_2)$
$\Rightarrow a$	$\Rightarrow b$
$\Rightarrow d$	$\Rightarrow c$
$2\colon b, a \rightarrow h$	
$2\colon c, d \rightarrow \neg\, h$	
$\sigma_1 = \{(\{A_3^1, A_6^1\}, \Rightarrow a)\}$	$\sigma_2 = \emptyset$

Six contextual arguments can be constructed in P_1

$A_1^1 = [\Rightarrow a]$ $\quad A_4^1 = [c]$
$A_2^1 = [b]$ $\quad A_5^1 = [\Rightarrow d]$
$A_3^1 = [A_1^1, A_2^1 \rightarrow h]$ $\quad A_6^1 = [A_4^1, A_5^1 \rightarrow \neg h]$

Two contextual arguments can be constructed in P_2

$$A_1^2 = [\Rightarrow b] \qquad A_2^2 = [\Rightarrow c]$$

We can see that $\mathcal{C}$ is a contextualization of defeasible logic program $\mathcal{P}$ and conflict resolution strategy σ from Example 2. Similarly, we will define a notion of *contextual version of argument* by examples: arguments A_1^1, A_3^1, A_5^1, A_6^1 are (in order) contextual versions of arguments A_1, A_3, A_5, A_6, but A_2^1, A_4^1 are not contextual versions of arguments A_2, A_4 in Example 1.

[5] Note that symbol C was originally used to denote a conflict and symbol $\mathcal{C}$ for denoting the set of all conflicts. However, the denotation of symbols will always be clear from the actual text.

The process of proving a literal L via an argument game in contextual setting is still consisting of two steps:

1. Find a contextual argument A with conclusion L.
2. Justify all vulnerabilities in $\textsc{Vuls}(A)$ and send acceptance queries to contexts pertaining to foreign literals $F(\{A\})$.

The second step means that whenever a player *pl* plays in a dialogue a move μ, not only all vulnerabilities of μ but also all foreign literals occurring in μ must be justified in order to *pl* will be the winner.

It is not hard to see that support dependency through foreign literals may be cyclic in a multi-context system. For example context C_1 may use a foreign literal from context C_2 and vice versa. Therefore we have to take care of termination of the queries to other contexts.

Example 7. Consider the following multi-context system consisting of two contexts, each using foreign literal from the other context.

Context 1	Context 2
$\Rightarrow a$	$1\colon a \Rightarrow \neg\, b$
$2\colon b \Rightarrow \neg\, a$	$\Rightarrow b$
$\sigma_1 = \{R_1 = (C_1, \Rightarrow a)\}$	$\sigma_2 = \{R_2 = (C_2, \Rightarrow b)\}$

Where conflict $C_1 = \{[\Rightarrow a], [2\colon b \Rightarrow \neg\, a]\}$ and conflict $C_2 = \{[1\colon a \Rightarrow \neg\, b], [\Rightarrow b]\}$.

Consider now query about credulous acceptance of literal a. There is only one rule deriving a and the only conflict resolution R_1 defeating it. Recall the intuitive meaning of conflict resolution in distributive setting: If the vulnerability $\{b \Rightarrow \neg\, a\}$ and foreign literal b are accepted, rule $\Rightarrow a$ is defeated. Defeasible rule $b \Rightarrow \neg\, a$ is not a resolution of any conflict so its trustworthiness is not a subject of dispute. Now the query about acceptance of the foreign literal b is given to the Context 2. The process of proving b in Context 2 is similar, therefore we skip details and only remark that query about acceptance of the foreign literal a is given back to the Context 1. We can see that naive adaptation of argument games may lead to infinite sending of queries between contexts which have cyclic support dependency.

To overcome problem illustrated in the previous example, from now on in this paper we investigate with acyclic multi-context systems only and more general cases are left for the future work.

Now we will define notions for contextual proving and argument games. Contextual argument game is an argument game T accompanied with a query function $\mathcal{Q}$ defining queries for every move in T. Intuitively, a query is a foreign literal that needs to proved in other context.

Definition 24 (Contextual Argument Game)**.** Let C be a context and μ be a move (pl, R, V). A *contextual argument game for a contextual argument A* is a pair $(T, \mathcal{Q})$, where T is an argument game for A and $\mathcal{Q}$ is

a query function

$$\mathcal{Q}(\mu) = \begin{cases} F(\{A\}) & \text{if } \mu \text{ is the root of the tree} \\ F(con(R)) & \text{otherwise} \end{cases}$$

assigning queries for each move.

We say that a contextual argument game for a literal L is a contextual argument game for a contextual argument A with $\textsc{Conc}(A) = L$. Given a query function $\mathcal{Q}$, the set of all foreign literals, played by a player pl in a contextual argument game $(T, \mathcal{Q})$, will be denoted by $\mathcal{Q}(pl)$.

Contextual skeptical and credulous games respect conditions of move repetitions. That is, in contextual skeptical (credulous) game, opponent (proponent) is allowed to repeat her moves and proponent (opponent) is not. However, since parts of the argument game can be queried to another contexts, we have to take care that requirements of (non)repetitions of moves are satisfied also there. Realize that each time a query about foreign literal F to other context C' is sent from a move $(pl, R, \mathcal{V})$ in an argument game T, no matter whether pl is proponent or opponent, the argument game for F in context C' will be started by proponent. Therefore, if pl is Pro, the semantics of argument game in context C' does not change. On the other hand, if pl is is Opp, the semantics of argument game in context C' will switch in order to keep the requirements of (non)repetitions of moves.

This leads into two mutually recursive definitions of skeptical and credulous contextual argument games. Note however that the recursion is well-founded (always terminates) since we are considering multi-context systems with acyclic support dependency only.

Definition 25 (Contextual Skeptical Game)**.** Let μ be a move $(pl, R, \mathcal{V})$. A contextual argument game $(T, \mathcal{Q})$ is called *skeptical* iff

- T is skeptical game and
- for each move in T with $\mathcal{Q}(\mu) \neq \emptyset$ there is a $sem(\mu)$ contextual argument game, where

$$sem(\mu) = \begin{cases} \text{skeptical} & \text{if } pl = \textsc{Pro} \\ \text{credulous} & \text{otherwise} \end{cases}$$

defines the acceptance semantics for queries.

Definition 26 (Contextual Credulous Game)**.** Let μ be a move $(pl, R, \mathcal{V})$. A contextual argument game $(T, \mathcal{Q})$ is called *credulous* iff

- T is credulous game and
- for each move in T with $\mathcal{Q}(\mu) \neq \emptyset$ there is a $sem(\mu)$ contextual argument game, where

$$sem(\mu) = \begin{cases} \text{credulous} & \text{if } pl = \textsc{Pro} \\ \text{skeptical} & \text{otherwise} \end{cases}$$

defines the acceptance semantics for queries.

Recall that player pl, in order to be the winner, has to justify not only all the vulnerabilities played by her, but also all pl's queries have to successful. Hence, although player does not play the last move in a dialogue, she can still be a winner if a query of the second player is not justified.

Again, the definition is recursive but the assumption of acyclicity guarantees its termination.

Definition 27 (Contextual Winner)**.** Let $(T, \mathcal{Q})$ be a contextual argument game. A player pl *wins a dialogue in contextual argument game* $(T, \mathcal{Q})$ iff

- all contextual argument games for literals in $\mathcal{Q}(pl)$ are successful and
- at least one of the following holds:
  - pl plays the last move in the dialogue, or
  - at least one of the contextual argument game for literals in $\mathcal{Q}(\overline{pl})$ is not successful.

A player Pro (resp. Opp) *wins a contextual argument game* iff she wins all (resp. at least one of the) branches in the contextual argument game. A contextual argument game is *successful* iff it is won by Pro.

Definition 28 (Contextually Proved Literal)**.** Let $\mathcal{C}$ be a multi-context system and $C \in \mathcal{C}$ be a context. A literal L is (skeptically, resp. credulously) *proved* in:

- a contextual argument game $(T, \mathcal{Q})$ iff there is a contextual argument A with $\textsc{Conc}(A) = L$, T is an (skeptical, resp. credulous) argument game for A and $(T, \mathcal{Q})$ is successful.
- a context C iff $C = (V, P, \sigma)$, $L \in V$ and there is a contextual argument game (skeptically, resp. credulously) proving L.
- a multi-context system $\mathcal{C}$ iff there is a context C such that L is (skeptically, resp. credulously) proved in C.

One of our goals was that contextualization of a program provides just a distributive computing and should not change its output. The following proposition claims that we are successful by achieving it.

Proposition 5. *Let $\mathcal{C}$ be an acyclic contextualization of a defeasible logic program P and L be a literal.*

1. *$P \models_{sk} L$ iff L is skeptically proved in $\mathcal{C}$.*
2. *$P \models_{cr} L$ iff L is credulously proved in $\mathcal{C}$.*

Distribution of argument games is demonstrated in example.

Example 8. Consider the following multi-context system consisting of two contexts

$C_1 = (\{a\}, P_1, \sigma_1)$	$C_2 = (\{b\}, P_2, \sigma_2)$
$\Rightarrow a$	$\Rightarrow b$
$2\colon b \Rightarrow \neg\, a$	$\Rightarrow \neg\, b$
$\sigma_1 = \{(\{A_1^1, A_3^1\}, \Rightarrow a)\}$	$\sigma_2 = \{(\{A_1^2, A_2^2\}, \Rightarrow b)\}$

Three contextual arguments can be constructed in P_1

$$A_1^1 = [\Rightarrow a] \qquad A_2^1 = [b]$$
$$A_3^1 = [A_2^1 \Rightarrow \neg\, a]$$

Two contextual arguments can be constructed in P_2

$$A_1^2 = [\Rightarrow b] \qquad A_2^2 = [\Rightarrow \neg\, b]$$

The contextual argument game T (both skeptical and credulous) is on the Figure 4, the contextual game T' for query b is on the Figure 5.

$$\mu_1^1 = (\textsc{Pro}, -, \{\Rightarrow a\})$$
$$|$$
$$\mu_2^1 = (\textsc{Opp}, R_1, \{2\colon b \Rightarrow \neg a\}),\ \mathcal{Q}(\mu_2^1) = \{b\}$$

Figure 4: The contextual argument game for literal a in context C_1.

$$\mu_1^2 = (\textsc{Pro}, -, \{\Rightarrow b\})$$
$$|$$
$$\mu_2^2 = (\textsc{Opp}, R_2, \{\Rightarrow \neg b\})$$

Figure 5: The contextual argument game for a query b in context C_2.

Although the proponent did not play the last move in T, she is still winner, since the query about foreign literal b was not successful.

Conclusion

We have developed a procedural conflict resolution based semantics by adaptation of skeptical and credulous argument games for complete semantics. The soundness and completeness properties for both type of games are proved, what is the main contribution of this paper. At the end we have showed how the semantics of defeasible logic program can be computed in a distributive fashion and both skeptical and credulous argument games were modified for multi-context systems. However, only multi-context systems with acyclic support dependency have been considered and the more general cases were left for the future work.

References

[Baláž, Frtús, and Homola 2013] Baláž, M.; Frtús, J.; and Homola, M. 2013. Conflict resolution in structured argumentation. In *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning.*

[Bikakis and Antoniou 2010] Bikakis, A., and Antoniou, G. 2010. Defeasible Contextual Reasoning with Arguments in Ambient Intelligence. *IEEE Transactions on Knowledge and Data Engineering* 22(11):1492–1506.

[Billington et al. 2010] Billington, D.; Antoniou, G.; Governatori, G.; and Maher, M. 2010. An inclusion theorem for defeasible logics. *ACM Trans. Comput. Logic* 12(1):6:1–6:27.

[Caminada and Amgoud 2007] Caminada, M., and Amgoud, L. 2007. On the evaluation of argumentation formalisms. *Artificial Intelligence* 171(5-6):286–310.

[Dung 1995] 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.

[García and Simari 2004] García, A. J., and Simari, G. R. 2004. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming* 4(2):95–138.

[Governatori et al. 2004] Governatori, G.; Maher, M. J.; Antoniou, G.; and Billington, D. 2004. Argumentation semantics for defeasible logic. *J. Log. and Comput.* 14:675–702.

[Modgil and Caminada 2009] Modgil, S., and Caminada, M. 2009. Proof theories and algorithms for abstract argumentation frameworks. In Rahwan, I., and Simari, G., eds., *Argumentation in Artificial Intelligence*. Springer Publishing Company Incorporated. 105–129.

[Modgil and Prakken 2011] Modgil, S., and Prakken, H. 2011. Revisiting Preferences and Argumentation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 1021–1026. AAAI Press.

[Prakken and Sartor 1997] Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Nonclassical Logics* 7(1):25–75.

[Prakken 2010] Prakken, H. 2010. An abstract framework for argumentation with structured arguments. *Argument & Computation* 1(2):93–124.

[Thimm and Kern-Isberner 2008] Thimm, M., and Kern-Isberner, G. 2008. A distributed argumentation framework using defeasible logic programming. In *Proceedings of the 2008 Conference on Computational Models of Argument: Proceedings of COMMA 2008*, 381–392. Amsterdam, The Netherlands, The Netherlands: IOS Press.

On the Relative Expressiveness of Argumentation Frameworks, Normal Logic Programs and Abstract Dialectical Frameworks

Hannes Strass
Computer Science Institute
Leipzig University, Germany

Abstract

We analyse the expressiveness of the two-valued semantics of abstract argumentation frameworks, normal logic programs and abstract dialectical frameworks. By expressiveness we mean the ability to encode a desired set of two-valued interpretations over a given propositional signature using only atoms from that signature. While the computational complexity of the two-valued model existence problem for all these languages is (almost) the same, we show that the languages form a neat hierarchy with respect to their expressiveness.

Introduction

More often than not, different knowledge representation languages have conceptually similar and partially overlapping intended application areas. What are we to do if faced with an application and a choice of several possible knowledge representation languages which could be used for the application? One of the first axes along which to compare different formalisms that comes to mind is computational complexity: if a language is computationally too expensive when considering the problem sizes typically encountered in practice, then this is a clear criterion for exclusion.

But what if the available language candidates have the same computational complexity? If their expressiveness in the computational-complexity sense of "What kinds of *problems* can the formalism solve?" is the same, we need a more fine-grained notion of expressiveness. In this paper, we use such an alternative notion and perform an exemplary study of the relative expressiveness of several different knowledge representation languages: argumentation frameworks (AFs) (Dung, 1995), normal logic programs (LPs), abstract dialectical frameworks (ADFs) (Brewka and Woltran, 2010) and propositional logic.

This choice of languages is largely motivated by the similar intended application domains of argumentation frameworks and abstract dialectical frameworks and the close relation of the latter to normal logic programs. We add propositional logic to have a well-known reference point. Furthermore, the computational complexity of their respective model existence problems is the same (with one exception):

- for AFs, deciding stable extension existence is NP-complete (Dimopoulos, Nebel, and Toni, 2002);
- for LPs, deciding the existence of supported/stable models is NP-complete (Bidoit and Froidevaux, 1991; Marek and Truszczyński, 1991);
- for ADFs, deciding the existence of models is NP-complete (Brewka et al., 2013), deciding the existence of stable models is Σ_2^P-complete for general ADFs (Brewka et al., 2013) and NP-complete for the subclass of bipolar ADFs (Strass and Wallner, 2014);
- the satisfiability problem of propositional logic is NP-complete.

In view of these almost identical complexities, we use an alternative measure of the expressiveness of a knowledge representation language L: "Given a set of two-valued interpretations, is there a knowledge base in L that has this exact model set?" This notion lends itself straightforwardly to compare different formalisms (Gogic et al., 1995):

> Formalism L_2 is at least as expressive as formalism L_1 if and only if every knowledge base in L_1 has an equivalent knowledge base in L_2.

So here expressiveness is understood in terms of *realisability*, "What kinds of model sets can the formalism express?"

It is easy to see that propositional logic can express any set of two-valued interpretations. The same is easy (but less easy) to see for logic programs under supported model semantics. For logic programs under stable model semantics, it is clear that not all model sets can be expressed, since two different stable models are always incomparable with respect to the subset relation. In this paper, we study such expressiveness properties for all the mentioned formalisms under different semantics. It will turn out that the languages form a more or less strict expressiveness hierarchy, with AFs at the bottom, ADFs and LPs under stable semantics higher up and ADFs and LPs under supported model semantics at the top together with propositional logic.

To show that a language L_2 is at least as expressive as a language L_1 we will mainly use two different techniques. In the best case, we can use a syntactic compact and faithful translation from knowledge bases of L_1 to those of L_2. *Compact* means that the translation does not change the vocabulary, that is, does not introduce new atoms. *Faithful* means that the translation exactly preserves the models of the knowledge base for respective semantics of the two languages. In the second best case, we assume given the

knowledge base of L_1 in the form of a set X of desired models and construct a semantic *realisation* of X in L_2, that is, a knowledge base in L_2 whose model set corresponds exactly to X. To show that language L_2 is *strictly more expressive* than L_1, we additionally have to present a knowledge base K from L_2 of which we prove that L_1 cannot express the model set of K.

For all methods, we can make use of several recent works on the formalisms we study here. First of all, we [2013] studied the syntactic intertranslatability of ADFs and LPs, but did not look at expressiveness or realisability. The latter was recently studied for argumentation frameworks by Dunne et al. (2014). They allow to extend the vocabulary in order to realise a given model set, as long as the new vocabulary elements are evaluated to false in all models. For several semantics of AFs, Dunne et al. found necessary (and sufficient) conditions for realisability. While their sufficient conditions are not applicable to our setting, they discovered a necessary condition for realisability with stable extension semantics that we will make use of in this paper. There has also been work on translating ADFs into AFs for the ADF model and AF stable extension semantics (Brewka, Dunne, and Woltran, 2011), however this translation introduces additional arguments and is therefore not compact.

The gain that is achieved by our results is not only that of increased clarity about fundamental properties of these knowledge representation languages – *What can these formalisms express, actually?* – but has several further applications. As Dunne et al. (2014) remarked, a major application is in constructing knowledge bases with the aim of encoding a certain model set. As a necessary prerequisite to this, it must be known that the intended model set is realisable in the first place. For example, in a recent approach to revising argumentation frameworks (Coste-Marquis et al., 2013), the authors avoid this problem by assuming to produce a *collection* of AFs whose model sets in union produce the desired model set. While the work of Dunne et al. (2014) showed that this is indeed necessary in the case of AFs and stable extension semantics (that is, there are model sets that a single AF just cannot express), our work shows that for ADFs under the model semantics, a single knowledge base (ADF) is always enough to realise any given model set.

Of course, the fact that the languages we study have the same computational complexity means that there in principle exist polynomial intertranslations for the respective decision problems. But such intertranslations may involve the introduction of new atoms. In theory, a polynomial blowup from n atoms to n^k atoms for some k is of no consequence. In practice, it has a profound impact: the number n of atoms directly influences the search space that any implementation potentially has to cover. There, an increase from 2^n to 2^{n^k} is no longer polynomial, but exponential, and accordingly makes itself felt. Being able to realise a model set compactly, without new atoms, therefore attests that a language L has a certain basic kind of efficiency property, in the sense that the L-realisation of a model set does not unnecessarily enlarge the search space of algorithms operating on it.

The paper proceeds as follows. We first define the notion of expressiveness formally and then introduce the languages we will study. After reviewing several intertranslatability results for these languages, we stepwise obtain the results that lead to the expressiveness hierarchy. We conclude with a discussion of avenues for future work.

Background

We assume given a finite set A of atoms (statements, arguments), the *vocabulary*. A knowledge representation language interpreted over A is then some set L; a (two-valued) semantics for L is a mapping $\sigma : L \to 2^{2^A}$ that assigns sets of two-valued models to the language elements. (So A is implicit in L.) Strictly speaking, a two-valued interpretation is a mapping from the set of atoms into the two truth values true and false, but for technical ease we represent two-valued interpretations by the sets containing the atoms that are true.

For a language L, we denote the range of the semantics σ by $\sigma(L)$. Intuitively, $\sigma(L)$ is the set of models that language L can express, with any knowledge base over vocabulary A whatsoever. For example, for $L = \text{PL}$ propositional logic and $\sigma = mod$ the usual model semantics, we have $\sigma(\text{PL}) = 2^{2^A}$ since obviously any set of models is realisable in propositional logic.[1] This leads us to compare different pairs of languages and semantics with respect to the semantics' range of models. Our concept of "language" concentrates on semantics and decidedly remains abstract.

Definition 1. Let A be a finite vocabulary, L_1, L_2 be languages that are interpreted over A and $\sigma_1 : L_1 \to 2^{2^A}$ and $\sigma_2 : L_2 \to 2^{2^A}$ be two-valued semantics. We define

$$L_1^{\sigma_1} \leq_e L_2^{\sigma_2} \quad \text{iff} \quad \sigma_1(L_1) \subseteq \sigma_2(L_2)$$

Intuitively, language L_2 under semantics σ_2 is at least as expressive as language L_1 under semantics σ_1, because all models that L_1 can express under σ_1 are also contained in those that L_2 can produce under σ_2. (If the semantics are clear from the context we will omit them; this holds in particular for argumentation frameworks and propositional logic, where we only look at a single semantics.) As usual,

- $L_1 <_e L_2$ iff $L_1 \leq_e L_2$ and $L_2 \not\leq_e L_1$;
- $L_1 \cong_e L_2$ iff $L_1 \leq_e L_2$ and $L_2 \leq_e L_1$.

The relation $\leq_e$ is reflexive and transitive by definition, but not necessarily antisymmetric. That is, there might different languages $L_1 \neq L_2$ that are equally expressive: $L_1 \cong_e L_2$.

We next introduce the particular knowledge representation languages we study in this paper. All will make use of a vocabulary A; the results of the paper are all considered parametric in such a given vocabulary.

Logic Programs

For a vocabulary A define $not\ A = \{not\ a \mid a \in A\}$ and the set of literals over A as $A^{\pm} = A \cup not\ A$. A *normal logic program rule* over A is then of the form $a \leftarrow B$ where $a \in A$ and $B \subseteq A^{\pm}$. The rule can be read as logical consequence, "a is true if all literals in B are true." The set B

[1] For a set $X \subseteq 2^A$ we can simply define $\varphi_X = \bigvee_{M \in X} \varphi_M$ with $\varphi_M = \bigwedge_{a \in M} a \wedge \bigwedge_{a \in A \setminus M} \neg a$ and clearly $mod(\varphi_X) = X$.

is called the *body* of the rule, we denote by $B^+ = B \cap A$ and $B^- = \{a \in A \mid not\ a \in B\}$ the *positive* and *negative body* atoms, respectively. A rule is *definite* if $B^- = \emptyset$. For singleton $B = \{b\}$ we denote the rule just by $a \leftarrow b$. A *logic program (LP)* P over A is a set of logic program rules over A, and it is definite if all rules in it are definite.

At first, logic programs were restricted to definite programs, whose semantics was defined through the proof-theoretic procedure of SLD resolution. The meaning of negation not was only defined operationally through negation as failure. Clark (1978) gave the first declarative semantics for normal logic programs via a translation to classical logic that will be recalled shortly. This leads to the supported model semantics for logic programs: A rule $a \leftarrow B \in P$ is *active* in a set $M \subseteq A$ iff $B^+ \subseteq M$ and $B^- \cap M = \emptyset$ imply $a \in M$. M is a *supported model* for P iff $M = \{a \in A \mid a \leftarrow B \in P \text{ is active in } M\}$. For a logic program P we denote the set of its supported models by $su(P)$. The intuition behind this semantics is that everything that is true in a model has some kind of support.

However, this support might be cyclic self-support. For instance, the logic program $\{a \leftarrow a\}$ has two supported models, $\emptyset$ and $\{a\}$, where the latter is undesired in many application domains. As an alternative, Gelfond and Lifschitz (1988) proposed the stable model semantics, a declarative semantics for negation as failure that does not allow self-support: $M \subseteq A$ is a *stable model* for P iff M is the $\subseteq$-least supported model of P^M, where the definite program P^M is obtained from P by (1) eliminating each rule whose body contains a literal $not\ a$ with $a \in M$, and (2) deleting all literals of the form $not\ a$ from the bodies of the remaining rules. We write $st(P)$ for the set of stable models of P. It follows from the definition of stable models that $st(P)$ is a $\subseteq$-antichain: for all $M_1 \neq M_2 \in st(P)$ we have $M_1 \not\subseteq M_2$.

Argumentation Frameworks

Dung (1995) introduced argumentation frameworks as pairs $F = (A, R)$ where A is a set and $R \subseteq A \times A$ a relation. The intended reading of an AF F is that the elements of A are arguments whose internal structure is abstracted away. The only information about the arguments is given by the relation R encoding a notion of attack: a pair $(a, b) \in R$ expresses that argument a attacks argument b in some sense.

The purpose of semantics for argumentation frameworks is to determine sets of arguments (called *extensions*) which are acceptable according to various standards. For a given extension $S \subseteq A$, the arguments in S are considered to be accepted, those that are attacked by some argument in S are considered to be rejected, and all others are neither, their status is undecided. We will only be interested in so-called *stable* extensions, sets S of arguments that do not attack each other and attack all arguments not in the set. For stable extensions, each argument is either accepted or rejected by definition, thus the semantics is two-valued. More formally, a set $S \subseteq A$ of arguments is *conflict-free* iff there are no $a, b \in S$ with $(a, b) \in R$. A set S is a *stable extension* for (A, R) iff it is conflict-free and for all $a \in A \setminus S$ there is a $b \in S$ with $(b, a) \in R$. For an AF F, we denote the set of its stable extensions by $st(F)$. Again, it follows from the definition of a stable extension that the set $st(F)$ is always a $\subseteq$-antichain.

Abstract Dialectical Frameworks

An abstract dialectical framework (ADF) is a directed graph whose nodes represent statements or positions which can be accepted or not. The links represent dependencies: the status of a node a only depends on the status of its parents (denoted $par(a)$), that is, the nodes with a direct link to a. In addition, each node a has an associated acceptance condition C_a specifying the exact conditions under which a is accepted. C_a is a function assigning to each subset of $par(a)$ one of the truth values $\mathbf{t}$ or $\mathbf{f}$. Intuitively, if for some $R \subseteq par(a)$ we have $C_a(R) = \mathbf{t}$, then a will be accepted provided the nodes in R are accepted and those in $par(a) \setminus R$ are not accepted.

More formally, an *abstract dialectical framework* is a tuple $D = (A, L, C)$ where

- A is a set of statements,
- $L \subseteq A \times A$ is a set of links,
- $C = \{C_a\}_{a \in A}$ is a collection of total functions $C_a : 2^{par(a)} \to \{\mathbf{t}, \mathbf{f}\}$, one for each statement a. The function C_a is called *acceptance condition of* a.

It is often convenient to represent acceptance conditions by propositional formulas. In particular, we will do so for several results of this paper. There, each C_a is represented by a propositional formula φ_a over $par(a)$. Then, clearly, $C_a(R \cap par(a)) = \mathbf{t}$ iff R is a model for φ_a, $R \models \varphi_a$.

Brewka and Woltran (2010) introduced a useful subclass of ADFs: an ADF $D = (A, L, C)$ is *bipolar* iff all links in L are supporting or attacking (or both). A link $(b, a) \in L$ is *supporting in* D iff for all $R \subseteq par(a)$, we have that $C_a(R) = \mathbf{t}$ implies $C_a(R \cup \{b\}) = \mathbf{t}$. Symmetrically, a link $(b, a) \in L$ is *attacking in* D iff for all $R \subseteq par(a)$, we have that $C_a(R \cup \{b\}) = \mathbf{t}$ implies $C_a(R) = \mathbf{t}$. If a link (b, a) is both supporting and attacking then b has no influence on a, the link is redundant (but does not violate bipolarity). We will sometimes use this circumstance when searching for ADFs; there we simply assume that $L = A \times A$, then links that are actually not needed can be expressed by acceptance conditions that make them redundant.

There are numerous semantics for ADFs; we will only be interested in two of them, (supported) models and stable models. A set $M \subseteq A$ is a *model of* D iff for all $a \in A$ we find that $a \in M$ iff $C_a(M) = \mathbf{t}$. The definition of stable models is inspired by logic programming and slightly more complicated (Brewka et al., 2013). Define an operator by $\Gamma_D(Q, R) = (acc(Q, R), rej(Q, R))$ for $Q, R \subseteq A$, where

$$acc(Q, R) = \{a \in A \mid \text{for all } Q \subseteq Z \subseteq (A \setminus R), \text{ we have } C_a(Z) = \mathbf{t}\}$$
$$rej(Q, R) = \{a \in A \mid \text{for all } Q \subseteq Z \subseteq (A \setminus R), \text{ we have } C_a(Z) = \mathbf{f}\}$$

The intuition behind the operator is as follows: A pair (Q, R) represents a partial interpretation of the set of statements where those in Q are accepted (true), those in R are rejected (false), and those in $S \setminus (Q \cup R)$ are neither.

The operator checks for each statement a whether all total interpretations that can possibly arise from (Q, R) agree on their truth value for the acceptance condition for a. That is, if a has to be accepted no matter how the statements in $S \setminus (Q \cup R)$ are interpreted, then $a \in acc(Q, R)$. The set $rej(Q, R)$ is computed symmetrically, so the pair $(acc(Q, R), rej(Q, R))$ constitutes a refinement of (Q, R).

For $M \subseteq A$, the reduced ADF $D^M = (M, L^M, C^M)$ is defined by $L^M = L \cap M \times M$ and for each $a \in M$ setting $\varphi_a^M = \varphi_a[b/\mathbf{f} : b \notin M]$, that is, replacing all $b \notin M$ by false in the acceptance formula of a. A model M for D is a *stable model* of D iff the least fixpoint of the operator Γ_{D^M} is given by $(M, \emptyset)$. As usual, $su(D)$ and $st(D)$ denote the model sets of the two semantics. While ADF models can be subsets of one another, ADF stable models cannot.

Translations between the formalisms

From AFs to BADFs Brewka and Woltran (2010) showed how to translate AFs into ADFs: For an AF $F = (A, R)$, define the ADF associated to F as $D(F) = (A, R, C)$ with $C = \{\varphi_a\}_{a \in A}$ and $\varphi_a = \bigwedge_{(b,a) \in R} \neg b$ for $a \in A$. Clearly, the resulting ADF is bipolar; parents are always attacking. Brewka and Woltran (2010) proved that this translation is faithful for the AF stable extension and ADF model semantics (Proposition 1). Brewka et al. (2013) later proved the same for the AF stable extension and ADF stable model semantics (Theorem 4). It is easy to see that the translation can be computed in polynomial time.

From ADFs to PL Brewka and Woltran (2010) also showed that ADFs under supported model semantics can be faithfully translated into propositional logic: When acceptance conditions of statements $a \in A$ are represented by propositional formulas φ_a, then the supported models of an ADF D over A are given by the classical models of the formula set $\{a \leftrightarrow \varphi_a \mid a \in A\}$.

From AFs to PL In combination, the previous two translations yield a polynomial and faithful translation chain from AFs into propositional logic.

From ADFs to LPs In recent work we showed that ADFs can be faithfully translated into normal logic programs (Strass, 2013). For an ADF $D = (A, L, C)$, its standard logic program $P(D)$ is given by

$$\{a \leftarrow (M \cup not\,(par(a) \setminus M)) \mid a \in A, C_a(M) = \mathbf{t}\}$$

It is an easy consequence of Lemma 3.14 in (Strass, 2013) that this translation preserves the supported model semantics. For complexity reasons, we cannot expect that this translation is also faithful for the stable semantics. And indeed, the ADF $D = (\{a\}, \{(a, a)\}, \{\varphi_a = a \vee \neg a\})$ has a stable model $\{a\}$ while its standard logic program $P(D) = \{a \leftarrow a, a \leftarrow not\ a\}$ has no stable model.

From AFs to LPs The translation chain from AFs to ADFs to LPs is compact, and faithful for AF stable semantics and LP stable semantics (Osorio et al., 2005), and AF stable semantics and LP supported semantics (Strass, 2013).

From LPs to PL It is well-known that normal logic programs under supported model semantics can be translated to propositional logic (Clark, 1978). There, a logic program P is translated to a propositional theory $\Phi_P = \{a \leftrightarrow \varphi_a \mid a \in A\}$ where

$$\varphi_a = \bigvee_{a \leftarrow B \in P} \left(\bigwedge_{b \in B^+} b \wedge \bigwedge_{b \in B^-} \neg b \right)$$

for $a \in A$. For the stable model semantics, additional formulas have to be added, but the extended translation works all the same (Lin and Zhao, 2004).

From LPs to ADFs The Clark completion of a normal logic program directly yields an equivalent ADF over the same signature (Brewka and Woltran, 2010). Clearly the translation is computable in polynomial time and the blowup (with respect to the original logic program) is at most linear. The resulting translation is faithful for the supported model semantics, which is a straightforward consequence of Lemma 3.16 in (Strass, 2013).

Relative Expressiveness

We now analyse and compare the relative expressiveness of argumentation frameworks – AFs –, (bipolar) abstract dialectical frameworks – (B)ADFs –, normal logic programs – LPs – and propositional logic – PL. We first look at the different families of semantics – supported and stable models – in isolation and afterwards combine the two. For the languages $L \in \{\text{ADF}, \text{LP}\}$ that have both supported and stable semantics, we will indicate the semantics σ via a superscript as in Definition 1. For AFs we only consider the stable extension semantics, as this is (to date) the only two-valued semantics for AFs. For propositional logic PL we consider the usual model semantics.

With the syntactic translations we reviewed in the previous section, we currently have the following relationships. For the supported semantics,

$$\text{AF} \leq_e \text{BADF}^{su} \leq_e \text{ADF}^{su} \cong_e \text{LP}^{su} \leq_e \text{PL}$$

and for the stable semantics,

$$\text{AF} \leq_e \text{BADF}^{st} \leq_e \text{ADF}^{st} <_e \text{PL}$$
$$\text{AF} \leq_e \text{LP}^{st} <_e \text{PL}$$

Note that $\text{ADF}^{st} <_e \text{PL}$ and $\text{LP}^{st} <_e \text{PL}$ hold since sets of stable models have an antichain property, in contrast to model sets of propositional logic.

Supported semantics

As depicted above, we know that expressiveness from AFs to propositional logic does not decrease. However, it is not yet clear if any of the relationships is strict.

We first show that ADFs can realise any set of models. To show this, we first make a case distinction whether the desired-model set is empty. If there should be no model, we construct an ADF without models. If the set of desired models is nonempty, we construct acceptance conditions directly from the set of desired interpretations. The construction is

similar in design to the one we reviewed for propositional logic, but takes into account the additional interaction between statements and their acceptance conditions.

Theorem 1. *PL* $\leq_e$ *ADF*^{su}

Proof. Consider a vocabulary A and a set $X \subseteq 2^A$. We construct an ADF D^{su}_X with $su(D^{su}_X) = X$ as follows.

1. $X = \emptyset$. We choose some $a \in A$ and set $D^{su}_X = (\{a\}, \{(a,a)\}, \{C_a\})$ with $C_a(\emptyset) = \mathbf{t}$ and $C_a(\{a\}) = \mathbf{f}$. It is easy to see that D^{su}_X has no model.
2. $X \neq \emptyset$. Define $D^{su}_X = (A, L, C)$ where $L = A \times A$ and for each $a \in A$ and $M \subseteq A$, we set $C_a(M) = \mathbf{t}$ iff

$$(M \in X \text{ and } a \in M) \text{ or } (M \notin X \text{ and } a \notin M)$$

We have to show that $M \in X$ iff M is a model for D^{su}_X.
"if": Let M be a model of D^{su}_X.
(a) $M = \emptyset$. Pick any $a \in A$. Since M is a model of D^{su}_X, we have $C_a(M) = \mathbf{f}$. So either (A) $M \in X$ and $a \notin M$ or (B) $M \notin X$ and $a \in M$, by definition of C_a. By assumption $M = \emptyset$, thus $a \notin M$ and $M \in X$.
(b) $M \neq \emptyset$. Let $a \in M$. Then $C_a(M) = \mathbf{t}$ since M is a model of D^{su}_X. By definition of C_a, $M \in X$.
"only if": Let $M \in X$.
(a) $M = \emptyset$. Choose any $a \in A$. By assumption, $a \notin M$ and $M \in X$, whence $C_a(M) = \mathbf{f}$ by definition. Since $a \in A$ was chosen arbitrarily, we have $C_a(M) = \mathbf{f}$ iff $a \notin M$. Thus M is a model of D^{su}_X.
(b) $M \neq \emptyset$. Let $a \in A$. If $a \in M$, then by assumption and definition of C_a we have $C_a(M) = \mathbf{t}$. Conversely, if $a \notin M$, then by definition $C_a(M) = \mathbf{f}$. Since $a \in A$ was arbitrary, M is a model of D^{su}_X. □

When the acceptance conditions are written as propositional formulas, the construction in Theorem 1 simply sets

$$\varphi_a = \bigvee_{M \in X, a \in M} \varphi_M \vee \bigvee_{M \subseteq A, M \notin X, a \notin M} \varphi_M$$

$$\varphi_M = \bigwedge_{a \in M} a \wedge \bigwedge_{a \in A \setminus M} \neg a$$

Since ADFs under supported semantics can be faithfully translated into logic programs, which can be likewise further translated to propositional logic, we have the following.

Corollary 2. *ADF*$^{su} \cong_e$ *LP*$^{su} \cong_e$ *PL*

While general ADFs under the supported model semantics can realise any set of models, the subclass of bipolar ADFs turns out to be less expressive. This is shown using the next result, which allows us to decide realisability of a given model set $X \subseteq 2^A$ in non-deterministic polynomial time. We assume that the size of the input is in the order of $\left|2^A\right|$, that is, the input set X is represented directly. The decision procedure then basically uses the construction of Theorem 1 and an additional encoding of bipolarity to define a reduction to the satisfiability problem in propositional logic.

Theorem 3. *Let $X \subseteq 2^A$ be a set of sets. It is decidable in non-deterministic polynomial time whether there exists a bipolar ADF D with $su(D) = X$.*

Proof. We construct a propositional formula ϕ_X that is satisfiable if and only if X is bipolarly realisable. The propositional signature we use is the following: For each $a \in A$ and $M \subseteq A$, there is a propositional variable p^M_a that expresses whether $C_a(M) = \mathbf{t}$. This allows to encode all possible acceptance conditions for the statements in A. To enforce bipolarity, we use additional variables to model supporting and attacking links: for all $a, b \in A$, there is a variable $p^{a,b}_{sup}$ saying that a supports b, and a variable $p^{a,b}_{att}$ saying that a attacks b. So the vocabulary of ϕ_X is given by

$$P = \left\{ p^M_a, p^{a,b}_{sup}, p^{a,b}_{att} \;\middle|\; M \subseteq A, a \in A, b \in A \right\}$$

To guarantee the desired set of models, we constrain the acceptance conditions as dictated by X: For any desired set M and statement a, the containment of a in M must correspond exactly to whether $C_a(M) = \mathbf{t}$; this is encoded in $\phi^{\in}_X$. Conversely, for any undesired set M and statement a, there must not be any such correspondence, which $\phi^{\notin}_X$ expresses.

$$\phi^{\in}_X = \bigwedge_{M \in X} \left(\bigwedge_{a \in M} p^M_a \wedge \bigwedge_{a \in A \setminus M} \neg p^M_a \right)$$

$$\phi^{\notin}_X = \bigwedge_{M \subseteq A, M \notin X} \left(\bigvee_{a \in M} \neg p^M_a \vee \bigvee_{a \in A \setminus M} p^M_a \right)$$

To enforce bipolarity, we state that each link must be supporting or attacking. To model the meaning of support and attack, we encode all ground instances of their definitions.

$$\phi_{bipolar} = \bigwedge_{a,b \in A} \left(\left(p^{a,b}_{sup} \vee p^{a,b}_{att} \right) \wedge \phi^{a,b}_{sup} \wedge \phi^{a,b}_{att} \right)$$

$$\phi^{a,b}_{sup} = p^{a,b}_{sup} \rightarrow \bigwedge_{M \subseteq A} \left(p^M_b \rightarrow p^{M \cup \{a\}}_b \right)$$

$$\phi^{a,b}_{att} = p^{a,b}_{att} \rightarrow \bigwedge_{M \subseteq A} \left(p^{M \cup \{a\}}_b \rightarrow p^M_b \right)$$

The overall formula is given by $\phi_X = \phi^{\in}_X \wedge \phi^{\notin}_X \wedge \phi_{bipolar}$. The rest of the proof – showing that X is bipolarly realisable if and only if ϕ_X is satisfiable – is delegated to Lemma 12 in the Appendix. □

Remarkably, the decision procedure does not only give an answer, but in the case of a positive answer we can read off the BADF realisation from the satisfying evaluation of the constructed formula. We illustrate the construction with an example that will subsequently be used to show that general ADFs are strictly more expressive than bipolar ADFs.

Example 1. Consider $A = \{x, y, z\}$ and this model set:

$$X_1 = \{\emptyset, \{x, y\}, \{x, z\}, \{y, z\}\}$$

The construction of Theorem 3 yields these formulas:

$$\begin{aligned}\phi^{\in}_{X_1} &= \neg p^{\emptyset}_x \wedge \neg p^{\emptyset}_y \wedge \neg p^{\emptyset}_z \wedge \\ &\quad p^{\{x,y\}}_x \wedge p^{\{x,y\}}_y \wedge \neg p^{\{x,y\}}_z \wedge \\ &\quad p^{\{x,z\}}_x \wedge \neg p^{\{x,z\}}_y \wedge p^{\{x,z\}}_z \wedge \\ &\quad \neg p^{\{y,z\}}_x \wedge p^{\{y,z\}}_y \wedge p^{\{y,z\}}_z \\ \phi^{\notin}_{X_1} &= (\neg p^{\{x\}}_x \vee p^{\{x\}}_y \vee p^{\{x\}}_z) \wedge \\ &\quad (p^{\{y\}}_x \vee \neg p^{\{y\}}_y \vee p^{\{y\}}_z) \wedge \\ &\quad (p^{\{z\}}_x \vee p^{\{z\}}_y \vee \neg p^{\{z\}}_z) \wedge \\ &\quad (\neg p^{\{x,y,z\}}_x \vee \neg p^{\{x,y,z\}}_y \vee \neg p^{\{x,y,z\}}_z)\end{aligned}$$

The remaining formulas about bipolarity are independent of X_1, we do not show them here. We have implemented the translation of Theorem 3 and used the solver clasp (Gebser et al., 2011) to verify that ϕ_{X_1} is unsatisfiable.

A manual proof of bipolar non-realisability of X_1 seems to amount to a laborious case distinction that explores the mutual incompatibility of the disjunctions in $\phi^{\notin}_{X_1}$ and bipolarity, a task that is better left to machines. Together with the straightforward statement of fact that X_1 can be realised by a non-bipolar ADF, the example leads to the next result.

Theorem 4. *BADF^{su} $<_e$ ADF^{su}*

Proof. The model set from **??** 1 is realisable under model semantics by ADF D_{X_1} with acceptance conditions

$$\varphi_x = (y \nleftrightarrow z), \quad \varphi_y = (x \nleftrightarrow z), \quad \varphi_z = (x \nleftrightarrow y)$$

where "$\nleftrightarrow$" denotes exclusive disjunction XOR. However, there is no bipolar ADF realising the model set X_1, as is witnessed by unsatisfiability of ϕ_{X_1} and Theorem 3. □

Clearly ADF D_{X_1} is not bipolar since in all acceptance formulas, all statements are neither supporting nor attacking. It is not the only realisation, some alternatives are given by

$$\begin{aligned} D'_{X_1}: &\quad \varphi_x = (y \nleftrightarrow z), \quad \varphi_y = y, \quad \varphi_z = z \\ D''_{X_1}: &\quad \varphi_x = x, \quad \varphi_y = (x \nleftrightarrow z), \quad \varphi_z = z \\ D'''_{X_1}: &\quad \varphi_x = x, \quad \varphi_y = y, \quad \varphi_z = (x \nleftrightarrow y) \end{aligned}$$

This shows that we cannot necessarily use the model set X_1 to determine a *single* reason for bipolar non-realisability, that is, a *single* link (b, a) that is neither supporting nor attacking in *all* realisations. Rather, the culprit(s) might be different in each realisation, and to show bipolar non-realisability, we have to prove that *for all* realisations, there necessarily *exists some* reason for non-bipolarity. And the number of different ADF realisations of a given model set X can be considerable, as our next result shows.

Proposition 5. *Let $|A| = n$, $X \subseteq 2^A$ with $\left|2^A \setminus X\right| = m$. The number of distinct ADFs D with $su(D) = X$ is*

$$r(n, m) = (2^n - 1)^m$$

Proof. We have to count the number of distinct models of the formula $\phi'_X = \phi^{\in}_X \wedge \phi^{\notin}_X$ from the proof of Theorem 3. We first observe that for each $a \in A$ and $M \subseteq A$, the propositional variable p^M_a occurs exactly once in ϕ'_X. Formula $\phi^{\in}_X$ is a conjunction of literals and does not contribute to combinatorial explosion. Formula $\phi^{\notin}_X$ contains m conjuncts. Each of the conjuncts is a disjunction of n distinct literals. There are $2^n - 1$ ways to satisfy such a disjunction. The claim now follows since for each of m conjuncts, we can choose one of $2^n - 1$ different ways to satisfy it. □

So the main contributing factor is the number m of interpretations that are excluded from the desired model set X. For **??** 1, for instance, there are $(2^3 - 1)^4 = 7^4 = 2401$ ADFs with the model set X_1. According to Theorem 4, none of them is bipolar. Obviously, the maximal number of realisations is achieved by $X = \emptyset$ whence $r(n, 2^n) = (2^n - 1)^{2^n}$. On the other hand, the model set $X = 2^A$ has exactly one realisation, $r(n, 0) = 1$.

It is comparably easy to show that BADF models are strictly more expressive than AFs, since sets of supported models of bipolar ADFs do not have the antichain property.

Proposition 6. *AF $<_e$ BADF^{su}*

Proof. Consider the vocabulary $A = \{a\}$ and the BADF $D = (A, \{(a, a)\}, \{\varphi_a\})$ with $\varphi_a = a$. It is straightforward to check that its model set is $su(D) = \{\emptyset, \{a\}\}$. Since model sets of AFs under stable extension semantics satisfy the antichain property, there is no equivalent AF over A. □

This yields the following overall relationships:

$$\text{AF} <_e \text{BADF}^{su} <_e \text{ADF}^{su} \cong_e \text{LP}^{su} \cong_e \text{PL}$$

Stable semantics

As before, we recall the current state of knowledge:

$$\text{AF} \leq_e \text{BADF}^{st} \leq_e \text{ADF}^{st} <_e \text{PL} \text{ and } \text{AF} \leq_e \text{LP}^{st} <_e \text{PL}$$

We first show that BADFs are strictly more expressive than AFs.

Proposition 7. *AF $<_e$ BADFst*

Proof. Consider the BADF from **??** 6, where the acceptance formula of the single statement a is given by $\varphi_a = a$. Its only stable model is $\emptyset$. However there is no AF with a single argument with the same set of stable extensions: the only candidates are $(\{a\}, \emptyset)$ and $(\{a\}, \{(a, a)\})$; their respective stable-extension sets are $\{\{a\}\}$ and $\emptyset$. □

Even if we discount for this special case of realising the empty stable extension, there are non-trivial extension-sets that AFs cannot realise.

Example 2 ((Dunne et al., 2014)). Consider the model set $X_2 = \{\{x, y\}, \{x, z\}, \{y, z\}\}$. Dunne et al. (2014) proved that X_2 is not realisable with stable AF semantics. Intuitively, the argument is as follows: Since x and y occur in an extension together, there can be no attack between them. The same holds for the pairs x, z and y, z. But then the set $\{x, y, z\}$ is conflict-free and thus there must be a stable extension containing all three arguments, which is not allowed by X_2. The reason is AFs' restriction to individual attack, as set attack (also called joint or collective attack) suffices to realise X_2 with BADF D under stable model semantics:

$$\varphi_x = \neg y \vee \neg z, \qquad \varphi_y = \neg x \vee \neg z, \qquad \varphi_z = \neg x \vee \neg y$$

Let us exemplarily show that $M = \{x, y\}$ is a stable model (the other cases are completely symmetric): The reduct D^M is characterised by the two acceptance formulas $\varphi_x = \neg y \vee \neg \mathbf{f}$ and $\varphi_y = \neg x \vee \neg \mathbf{f}$. We then easily find that $\Gamma_{D^M}(\emptyset, \emptyset) = (M, \emptyset) = \Gamma_{D^M}(M, \emptyset)$.

The construction from the previous example model set comes from logic programming (Eiter et al., 2013) and can be generalised to realise any non-empty model set satisfying the antichain property.

Definition 2. Let $X \subseteq 2^A$. Define the following BADF $D^{st}_X = (A, L, C)$ where C_a for $a \in A$ is given by

$$\varphi_a = \bigvee_{M \in X, a \in M} \left(\bigwedge_{b \in A \setminus M} \neg b \right)$$

and thus $L = \{(b, a) \mid M \in X, a \in M, b \in A \setminus M\}$.

We next show that the construction indeed works.

Theorem 8. *Let X with $\emptyset \neq X \subseteq 2^A$ be a $\subseteq$-antichain. We find that $st(D^{st}_X) = X$.*

Proof. Let $M \subseteq A$.

"$\subseteq$": Let $M \notin X$. We show that $M \notin su(D^{st}_X) \supseteq st(D^{st}_X)$.

1. There is an $N \in X$ with $M \subsetneq N$. Then there is an $a \in N \setminus M$. Consider its acceptance formula φ_a. Since $a \in N$ and $N \in X$, the formula φ_a has a disjunct $\psi_{a,N} = \bigwedge_{b \in A \setminus N} \neg b$. Now $M \subseteq N$ implies $A \setminus N \subseteq A \setminus M$ and M is a model for $\psi_{a,N}$. Thus M is a model for φ_a although $a \notin M$, hence $M \notin su(D^{st}_X)$.
2. For all $N \in X$, we have $M \not\subseteq N$. Obviously $M \neq \emptyset$ since $X \neq \emptyset$. Let $a \in M$. For each $N \in X$ with $a \in N$, the acceptance formula φ_a contains a disjunct $\psi_{a,N} = \bigwedge_{b \in A \setminus N} \neg b$. By assumption, for each $N \in X$ there is a $b_N \in M \setminus N$. Clearly $b_N \in A \setminus N$ and b_N is evaluated to true by M. Hence for each $N \in X$ with $a \in N$, the disjunct $\psi_{a,N}$ is evaluated to false by M. Thus φ_a is false under M and $M \notin su(D^{st}_X)$.

"$\supseteq$": Let $M \in X$. We first show that M is a model of D^{st}_X, that is: for all $a \in A$, $a \in M$ iff M is a model for φ_a.

1. Let $a \in M$. By construction, we have that φ_a in D^{st}_X contains a disjunct of the form $\psi_{a,M} = \bigwedge_{b \in A \setminus M} \neg b$. According to the interpretation M, all such $b \in A \setminus M$ are false and thus $\psi_{a,M}$ is true whence φ_a is true.
2. Let $a \in A \setminus M$ and consider its acceptance formula φ_a. Assume to the contrary that M is a model for φ_a. Then there is some $N \in X$ with $a \in N$ such that M is a model for $\psi_{a,N} = \bigwedge_{b \in A \setminus N} \neg b$, that is, $A \setminus N \subseteq A \setminus M$. Hence $M \subseteq N$ and X is not a $\subseteq$-antichain. Contradiction. Thus M is no model for φ_a.

Now consider the reduct D^M of D^{st}_X with respect to M. There, φ^M_a contains the disjunct $\psi^M_{a,M} = \psi_{a,M}[b/\mathbf{f} : b \notin M]$ where all $b \in A \setminus M$ have been replaced by false, whence $\psi^M_{a,M} = \neg \mathbf{f} \wedge \ldots \wedge \neg \mathbf{f}$ and φ^M_a is equivalent to true. Thus each $a \in M$ is true in the least fixpoint of Γ_{D^M} and thus $M \in st(D^{st}_X)$. □

The restriction to non-empty model sets is immaterial, since we can use the construction of Theorem 1 to realise the empty model set.

Since the stable model semantics for both ADFs and normal logic programs have the antichain property, the following is clear.

Corollary 9. *ADFst $\leq_e$ BADFst and LPst $\leq_e$ BADFst*

For the family of stable semantics, this leads to the following overall expressiveness relationships:

$$\text{AF} <_e \text{BADF}^{st} \cong_e \text{ADF}^{st} \cong_e \text{LP}^{st} <_e \text{PL}$$

Supported vs. stable semantics

Now we put the supported and stable pictures together. From the proof of Theorem 8, we can read off that for the canonical realisation D^{st}_X of an antichain X, the supported and stable semantics coincide, that is, $su(D^{st}_X) = st(D^{st}_X) = X$. With this observation, also bipolar ADFs under the supported semantics can realise any antichain, and we have this:

Proposition 10. *BADFst $\leq_e$ BADF^{su}*

As we have seen in **??** 6, there are bipolar ADFs with supported-model sets that are not antichains. Thus we get the following result.

Corollary 11. *BADFst $<_e$ BADF^{su}*

This result allows us to close the last gap and put together the big picture in Figure 1 below.

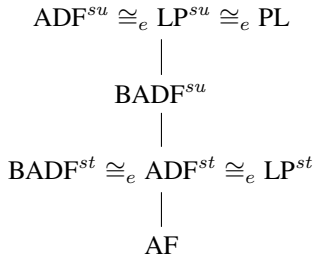


Figure 1: The expressiveness hierarchy. Expressiveness strictly increases from bottom to top. L^σ denotes language L under semantics σ, where "su" is the supported and "st" the stable model semantics; languages are among AFs (argumentation frameworks), ADFs (abstract dialectical frameworks), BADFs (bipolar ADFs), LPs (normal logic programs) and PL (propositional logic).

Discussion

We compared the expressiveness of abstract argumentation frameworks, abstract dialectical frameworks, normal logic programs and propositional logic. We showed that expressiveness under different semantics varies for the formalisms and obtained a neat expressiveness hierarchy. These results inform us about the capabilities of these languages to encode sets of two-valued interpretations, and help us decide which languages to use for specific applications.

For instance, if we wish to encode arbitrary model sets, for example when using model-based revision, then ADFs

and logic programs under supported semantics are a good choice. If we are happy with the restricted class of model sets having the antichain property, then we would be ill-advised to use general ADFs under stable model semantics with their Σ_2^P-hard stable model existence problem; to realise an antichain, it suffices to use bipolar ADFs or normal logic programs, where stable model existence is in NP.

There is much potential for further work. First of all, for results on non-realisability, it would be better to have necessary conditions than having to use a non-deterministic decision procedure. For this, we need to obtain general criteria that all model sets of a given formalism must obey, given the formalism is not universally expressive. This is non-trivial in general, and for AFs it constitutes a major open problem (Dunne et al., 2014; Baumann et al., 2014). Likewise, we sometimes used semantical realisations instead of syntactic ones; for example, to show universal realisability of ADFs under supported models we started out with model sets. It is an interesting question whether a realising ADF can be constructed from a given propositional formula without computing the models of the formula first. Second, there are further semantics for abstract dialectical frameworks whose expressiveness could be studied; Dunne et al. (2014) already analyse many of them for argumentation frameworks. This work is thus only a start and the same can be done for the remaining semantics, for example admissible, complete, preferred and others, which are all defined for AFs, (B)ADFs and LPs (Strass, 2013; Brewka et al., 2013). Third, there are further formalisms in abstract argumentation (Brewka, Polberg, and Woltran, 2013) whose expressiveness is by and large unexplored to the best of our knowledge. Fourth, the requirement that realisations may only use a fixed vocabulary without any additional symbols is quite restrictive. Intuitively, it should be allowed to add a reasonable number of additional atoms, for example a constant number or one that is linear in the original vocabulary. Finally, our study only considered *if* a language can express a model set, but not *to what cost* in terms of representation size. So the natural next step is to consider the succinctness of formalisms, "How large is the smallest knowledge base expressing a given model set?" (Gogic et al., 1995). A landmark result in this direction has been obtained by Lifschitz and Razborov (2006), who have shown that logic programs (with respect to two-valued stable models) are exponentially more succinct than propositional logic. That is, there are logic programs whose respective sets of stable models cannot be expressed by a propositional formula whose size is at most polynomial in the size of the logic program, unless a certain widely believed assumption of complexity theory is false. With the results of the present paper, we have laid the groundwork for a similar analysis of the other knowledge representation languages considered here, perhaps working towards a "map" of these languages in the sense of Darwiche and Marquis' knowledge compilation map [2002].

Acknowledgements. The author wishes to thank Stefan Woltran for providing a useful pointer to related work on realisability in logic programming, and Frank Loebe for several informative discussions. This research was partially supported by DFG (project BR 1817/7-1).

References

Baumann, R.; Dvořák, W.; Linsbichler, T.; Strass, H.; and Woltran, S. 2014. Compact argumentation frameworks. In Konieczny, S., and Tompits, H., eds., *Proceedings of the Fifteenth International Workshop on Non-Monotonic Reasoning (NMR)*.

Bidoit, N., and Froidevaux, C. 1991. Negation by default and unstratifiable logic programs. *Theoretical Computer Science* 78(1):85–112.

Brewka, G., and Woltran, S. 2010. Abstract Dialectical Frameworks. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 102–111.

Brewka, G.; Ellmauthaler, S.; Strass, H.; Wallner, J. P.; and Woltran, S. 2013. Abstract Dialectical Frameworks Revisited. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, 803–809. IJCAI/AAAI.

Brewka, G.; Dunne, P. E.; and Woltran, S. 2011. Relating the Semantics of Abstract Dialectical Frameworks and Standard AFs. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 780–785. IJCAI/AAAI.

Brewka, G.; Polberg, S.; and Woltran, S. 2013. Generalizations of Dung frameworks and their role in formal argumentation. *IEEE Intelligent Systems* PP(99). Special Issue on Representation and Reasoning. In press.

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

Coste-Marquis, S.; Konieczny, S.; Mailly, J.-G.; and Marquis, P. 2013. On the revision of argumentation systems: Minimal change of arguments status. *Proceedings of TAFA*.

Darwiche, A., and Marquis, P. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research (JAIR)* 17:229–264.

Dimopoulos, Y.; Nebel, B.; and Toni, F. 2002. On the computational complexity of assumption-based argumentation for default reasoning. *Artificial Intelligence* 141(1/2):57–78.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence* 77:321–358.

Dunne, P. E.; Dvořák, W.; Linsbichler, T.; and Woltran, S. 2014. Characteristics of Multiple Viewpoints in Abstract Argumentation. In *Proceedings of the Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*. To appear.

Eiter, T.; Fink, M.; Pührer, J.; Tompits, H.; and Woltran, S. 2013. Model-based recasting in answer-set programming. *Journal of Applied Non-Classical Logics* 23(1–2):75–104.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications* 24(2):105–124. Available at `http://potassco.sourceforge.net`.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*, 1070–1080. The MIT Press.

Gogic, G.; Kautz, H.; Papadimitriou, C.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 862–869. Morgan Kaufmann.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM Transactions on Computational Logic* 7(2):261–268.

Lin, F., and Zhao, Y. 2004. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence* 157(1-2):115–137.

Marek, V. W., and Truszczyński, M. 1991. Autoepistemic logic. *Journal of the ACM* 38(3):587–618.

Osorio, M.; Zepeda, C.; Nieves, J. C.; and Cortés, U. 2005. Inferring acceptable arguments with answer set programming. In *Proceedings of the Sixth Mexican International Conference on Computer Science (ENC)*, 198–205. IEEE Computer Society.

Strass, H., and Wallner, J. P. 2014. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. In *Proceedings of the Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*. To appear.

Strass, H. 2013. Approximating operators and semantics for abstract dialectical frameworks. *Artificial Intelligence* 205:39–70.

Appendix

Lemma 12. *X is bipolarly realisable if and only if the formula ϕ_X from Theorem 3 is satisfiable.*

Proof. "if": Let $I \subseteq P$ be a model for ϕ_X. For each $a \in A$, we define an acceptance condition as follows: for $M \subseteq A$, set $C_a(M) = \mathbf{t}$ iff $p_a^M \in I$. It is easy to see that $\phi_{bipolar}$ guarantees that these acceptance conditions are all bipolar. The ADF is now given by $D_X^{su} = (A, A \times A, C)$. It remains to show that any $M \subseteq A$ is a model of D_X^{su} if and only if $M \in X$.

"if": Let $M \in X$. We have to show that M is a model of D_X^{su}. Consider any $a \in A$.

1. $a \in M$. Since I is a model of $\phi_X^{\in}$, we have $p_a^M \in I$ and thus by definition $C_a(M) = \mathbf{t}$.
2. $a \in A \setminus M$. Since I is a model of $\phi_X^{\in}$, we have $p_a^M \notin I$ and thus by definition $C_a(M) = \mathbf{f}$.

"only if": Let $M \notin X$. Since I is a model of $\phi_X^{\notin}$, there is an $a \in M$ such that $C_a(M) = \mathbf{f}$ or an $a \notin M$ such that $C_a(M) = \mathbf{t}$. In any case, M is not a model of D_X^{su}.

"only if": Let D be a bipolar ADF with $su(D) = X$. We use D to define a model I for ϕ_X. First, for $M \subseteq A$ and $a \in A$, set $p_a^M \in I$ iff $C_a(M) = \mathbf{t}$. Since D is bipolar, each link is supporting or attacking and for all $a, b \in A$ we can find a valuation for $p_{sup}^{a,b}$ and $p_{att}^{a,b}$. It remains to show that I is a model for ϕ_X.

1. I is a model for $\phi_X^{\in}$: Since D realises X, each $M \in X$ is a model of D and thus for all $a \in A$ we have $C_a(M) = \mathbf{t}$ iff $a \in M$.
2. I is a model for $\phi_X^{\notin}$: Since D realises X, each $M \subseteq A$ with $M \notin X$ is not a model of D. Thus for each such M, there is an $a \in A$ witnessing that M is not a model of D: (1) $a \in M$ and $C_a(M) = \mathbf{f}$, or (2) $a \notin M$ and $C_a(M) = \mathbf{t}$.
3. I is a model for $\phi_{bipolar}$: This is straightforward since D is bipolar by assumption. □