Proceedings of the Twelfth International Workshop on Non-Monotonic Reasoning

Editors:

Maurice Pagnucco
ARC Centre of Excellence in Autonomous Systems
and National ICT Australia
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052
Australia
morri@cse.unsw.edu.au

Michael Thielscher
Computational Logic Group
Artificial Intelligence Institute
Department of Computer Science
Dresden University of Technology
D-01062 Dresden
Germany
mit@inf.tu-dresden.de

Technical Report
UNSW-CSE-TR-0819
September 2008

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Preface

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

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

The workshop programme is chaired by Maurice Pagnucco and Michael Thielscher, and the programme is composed of the following sessions (with session chairs).

1. Actions and Belief Change (Sebastian Sardina and Dongmo Zhang)
2. Applications (Tomi Janhunen and Eugenia Ternovska)
3. Argument, Dialogue and Decision (Yannis Dimopoulos and Gerard Vreeswijk)
4. Declarative Programming Paradigms and Systems for NMR (Esra Erdem and João Leite)
5. Preferences (Hans Tompits and Kewen Wang)
6. Foundations of NMR and Uncertainty (Alberto Finzi and Frank Wolter)

Authors have been invited to submit papers directly to any of the above sessions, and all papers have been reviewed by two or three experts in the field.

In addition to the paper sessions, this year's workshop featured joint invited talks by two internationally renowned researchers: Alexander Bochman (Holon Academic Institute of Technology) and Fangzhen Lin (Hong Kong University of Science and Technology).

The programme chairs are very grateful to the session chairs for organizing each session, and for arranging the reviewing of the submissions. The programme chairs are also very grateful to the reviewers for their hard work in assessing the submissions and for providing excellent feedback to the authors.

We would also like to thank Mirek Truszczynski for his financial support for the workshop. Our special thanks go to Sandra Grossmann who put these Proceedings together. This turned out to be an enormous effort and we appreciate their work very much.

August 2008

Maurice Pagnucco University of New South Wales
Michael Thielscher Dresden University of Technology

Table of Contents

Special Session on Actions and Belief Change

Reasoning about actions, causality, and belief change are well established research areas in nonmonotonic reasoning. Thanks to advances in these areas in recent years, it becomes evident that the boundaries between these research areas are hardly discernible and that they have more in common than was previously believed. For this reason, from the last NRAC workshop, the traditional NMR "Action and Change" and "Belief Change" tracks have been merged into a new "Actions and Belief Change" sub-workshop.

In addition to the traditional topics in the areas of reasoning about actions and belief change, we particularly encourage papers exploring the common territory that will further promote the cross-fertilization between these two areas. This would include, for example, reasoning about complex and dynamic environments, belief and knowledge merging under actions, multi-agent belief revision through communication, and so on.

Session Chairs

Sebastian Sardina, RMIT University, Australia
Dongmo Zhang, University of Western Sydney, Australia

Program Committee

Chitta Baral, Arizona State University, USA
Richard Booth, Mahasarakham University, Thailand
James Delgrande, Simon Fraser University, Canada
Eduardo Fermé, University of Madeira, Portugal
Alfredo Gabaldon, National ICT Australia, Australia
Aditya Ghose, University of Wollongong, Australia
Koen Hindriks, Delft University of Technology, The Netherlands
John-Jules Meyer, Utrecht University, The Netherlands
Pavlos Peppas, University of Patras, Greece
Laurent Perrussel, IRIT – Toulouse, France
Guilin Qi, University of Karlsruhe, Germany
Hans Rott, University of Regensburg, Germany
Mikhail Soutchanski, Ryerson University, Canada

Judgment Aggregation with Rule Confidence Scores

Farah Benamara
IRIT-CNRS
118 rt de Narbonne, 31062
Toulouse, France
benamara@irit.fr

Souhila Kaci
Université Lille-Nord de France, Artois
CRIL, CNRS UMR 8188
F-62307 - IUT de Lens
kaci@cril.univ-artois.fr

Gabriella Pigozzi
University of Luxembourg
Computer Science and Communications
Luxembourg
gabriella.pigozzi@uni.lu

Abstract

Judgment aggregation is a recent formal discipline that studies how to aggregate individual judgments to form collective decisions. Examples are expert panels, legal courts, boards, and councils. The problems investigated in this new field are relevant and common to many situations. Nevertheless, the existing procedures are idealized and, as for the related problems of preference aggregation in social choice theory, the field is plagued by impossibility theorems. In order to escape the impossibility results, a more realistic framework is needed. The goal of this paper is to extend standard judgment aggregation to take into account the judgment status and the confidence a group member may have in the decision rule. We propose to distinguish between abstainers and neutral judgment as well as to model the notion of confidence by assigning to each criterion a normalized weight. We then show how this additional information may help us to avoid indecision.

Introduction

Judgment aggregation is a recent formal discipline that studies how to aggregate individual judgments to form collective decisions. Examples are expert panels, legal courts, boards, and councils (List 2007). This field has recently attracted attention in multi-agent systems and artificial intelligence, in particular due to the relations with belief merging (Pigozzi 2006), for example for the combination of opinions of equally reliable individuals.

Judgment aggregation problems consider a group of people stating their views (in the binary form of 1 or 0) on some logically interconnected propositions. An example is the problem of choosing a candidate for a professor position in a university (Bovens & Rabinowicz 2006). A candidate is offered the job (*conclusion* R) only if she is good at teaching (*premise* P) and good at research (*premise* Q), that is the decision rule can be expressed as $(P \wedge Q) \leftrightarrow R$. As we will see, problems arise because a seemingly reasonable aggregation procedure leads to paradoxical outcomes.

Clearly, the problems investigated in this new field are relevant and common to many situations. Nevertheless, the existing procedures are idealized and, as for the related problems of preference aggregation in social choice theory (Dietrich & List 2006), the field is plagued by impossibility theorems. To provide a more realistic framework, and to escape the impossibility results are among the goals of the paper. More specifically, we introduce the following changes:

1. An agent may not vote and thus abstain. In the previous example, a committee member may abstain because she believes that the decision rule is inappropriate, or because there is no suitable candidate.
2. It is not realistic to impose that the agents always have a clear position on every proposition. Our model allows the individuals to express a neutral judgment. It is worth noticing that abstention and neutral judgments are distinct. The difference will be clarified later in the paper.
3. Borrowing the terminology from the field of multiple-criteria decision making (Keeney & Raiffa 1976; Figueira, Greco, & Ehrgott 2004), we call the propositions that support a certain conclusion *criteria* (instead of premises). This is justified by the fact that, in many decision problems, agents make their evaluations by taking into account different criteria and, even when the individuals agree on the criteria, they may assign them different weights. For example, some agents may deem research to be more important than teaching. Moreover, our framework allows the group members to state whether or not they agree on the rule governing the decision.
4. Some procedures can avoid paradoxical outcomes at the price of indecision (Pigozzi 2006). However, indecision is a very tedious problem. Our more refined procedure is an attempt to escape the impossibility results in judgment aggregation problems while, at the same time, resolves indecision.

In this paper, we propose to extend standard judgment aggregation to take into account the above considerations. More precisely, we will answer the following research questions:

- How to model *judgment status*? We will distinguish three cases: (a) classical binary evaluations, (b) neutral judgments and (c) abstentions.
- How to model *the confidence a member has in the decision rule*? We propose to model the notion of confidence

by assigning to each criterion a normalized weight.

- How to adapt standard aggregation procedures to take into account the judgment status as well as the confidence? We then show how this additional information may help us to avoid most cases of indecision.
- How to model *degree of support*? We propose the notion of *legitimacy*.

The remainder of this paper is organized as follows. After necessary background on the problem of judgment aggregation, we first recall some related works. We then present our general framework extending classical judgment aggregation procedure with confidence in the decision rule, judgment status and legitimacy of the result. After that we introduce the formal representation, the aggregation procedure and show that classical judgment aggregation is a special case of our more generalized framework. Lastly, we conclude.

Judgment aggregation

In the original problem of judgment aggregation (Kornhauser & Sager 1986; 1993), a court has to make a decision on whether a person is liable of breaching a contract (proposition R). The judges have to reach a verdict following the legal doctrine. This states that a person is liable if and only if there was a contract (P) and there was a conduct constituting breach of such a contract (Q). The legal doctrine can be formally expressed by the rule $(P \wedge Q) \leftrightarrow R$. Each member of the court expresses her judgment on the propositions P, Q and R such that the rule $(P \wedge Q) \leftrightarrow R$ is satisfied.

Suppose now that the three members of the court make their judgments according to Table 1.

	P	Q	$R = (P \wedge Q)$
Judge A	1	0	0
Judge B	0	1	0
Judge C	1	1	1
Majority	1	1	0

Table 1: Doctrinal paradox. Premises: P = There was a contract, Q = There was conduct constituting breach of such a contract. Conclusion: $R = (P \wedge Q)$ = There was a breach of contract.

Each judge expresses a consistent opinion, i.e. she says yes to R if and only if she says yes to both P and Q. However, *proposition-wise* majority voting (consisting in the separate aggregation of the votes for each proposition P, Q and R via majority rule) results in a majority for P and Q and yet a majority for $\neg R$. This is an inconsistent collective result, in the sense that $\{P, Q, \neg R, (P \wedge Q) \leftrightarrow R\}$ is inconsistent in propositional logic. The paradox lies in the fact that majority voting can lead a group of rational agents to endorse an irrational collective judgment, i.e. to have a majority believing that the defendant should be left free while *another* majority deems there are reasons to sentence her. The literature on judgment aggregation refers to such problem as the *doctrinal paradox*. Clearly, the relevance of such aggregation problems goes beyond the specific court example and affects all collective decisions on logically interconnected propositions.

The first two ways to avoid the inconsistency that have been suggested are the *premise-based procedure* and the *conclusion-based procedure* (Pettit 2001; Chapman 2002). According to the premise-based procedure, each member casts her vote on each premise. The conclusion is then inferred from the judgment of the majority of the group on the premises using the rule $(P \wedge Q) \leftrightarrow R$. In the example above, the premise-based procedure would declare the defendant liable of breaching the contract.

According to the conclusion-based procedure, the members decide privately on P and Q and only express their opinions on R publicly. The judgment of the group is then inferred from applying the majority rule to the agent judgments on the conclusion. The defendant will be declared liable if and only if a majority of the judges actually believes that she is liable, and no reasons for the court decision could be supplied. In the example, contrary to the premise-based procedure, the application of the conclusion-based procedure would free the defendant.

In order to investigate how strong the paradoxical outcomes are, some seemingly reasonable conditions were assumed on the aggregation function. Unfortunately, most of the results obtained in the field are negative (List & Pettit 2002; 2004; Pauly & van Hees 2006; Dietrich 2006).

To give a flavor of a typical impossibility result in the judgment aggregation field, and to introduce some terminology, we state the first impossibility theorem (List & Pettit 2002). A set of agents $N = \{1, 2, \ldots, n\}$, with $n \geq 3$, has to make judgments on logically interconnected propositions of a language $\mathcal{L}$. The set of propositions on which the judgments have to be made is called *agenda*. A (individual or collective) *judgment set* is the set of propositions believed by the agents or the group. An n-tuple $(J_1, J_2, \ldots, J_n)$ of agent judgment sets is called *profile*. A *judgment aggregation rule* F assigns a collective judgment set J to each profile $(J_1, J_2, \ldots, J_n)$ of agent judgment sets. A judgment set is *consistent* if it is a consistent set in $\mathcal{L}$, and is *complete* if, for any $P \in \mathcal{L}$, $P \in J$ or $\neg P \in J$.

A set of seemingly rational and desirable conditions are imposed on the aggregation rules and then, typically, an impossibility result is derived. The first impossibility theorem of judgment aggregation states that there exists no aggregation rule F satisfying the following conditions:

Universal Domain: The domain of F is the set of all profiles of consistent and complete judgment sets.

Anonymity: Intuitively, this means that all agents have equal weight.

Systematicity: This condition ensures that the collective judgment on each proposition depends only on the agent judgments on that proposition, and that the aggregation rule is the same across all propositions. Systematicity is clearly a very strong condition. In subsequent impossibil-

ity results, systematicity has been weakened to the independence of irrelevant alternatives:

Independence of Irrelevant Alternatives (IIA): IIA is systematicity without the neutrality condition, requiring that all propositions are equally treated.

Related works

In this section we refer to works that proposed to relax some of the assumptions made in the classical judgment aggregation framework. However, our model is the first that combines all these different aspects and introduces new ones.

Abstention and neutral judgments

Results in judgment aggregation usually assume complete judgment sets both at the individual and collective level. Gärdenfors (2006) was the first to criticize such assumption as being too strong and unrealistic. He allows voters to abstain from expressing judgments on some propositions in the agenda. He proves that, if the judgment sets may not be complete (but logically closed and consistent), then every aggregation function that is IIA and Paretian[1], must be oligarchic.[2]

Gärdenfors' framework requires the agenda to have a very rich logical structure (with an infinite number of issues). More recently, Dokow and Holzman (Dokow & Holzman 2007) extended Gärdenfors' result and consider finite agendas. Again, impossibility results are obtained. Hence, relaxing the completeness assumption does not avoid the impossibility results.

Nevertheless, allowing the voters to not express their judgments on some of the issues in the agenda provides a more realistic model of judgment aggregation, which is the aim of our paper. In order to avoid confusion, we must observe that we distinguish abstaining from being neutral with respect to an issue in the agenda. Abstentions in Gärdenfors and Dokow and Holzman' works correspond to what we call "neutral judgments". In our model, a voter abstains when she does not state her judgments on *any* issue in the agenda. Abstaining is a meaningful position, i.e. the refuse to participate in the decision process. This will affect the *legitimacy* of the decision outcome as an election with a high abstention rate is invalid.

On the other hand, being neutral on a certain issue captures those situations in which voters do not have a clear position on that issue, do not feel competent, or simply prefer not to take position on that matter. Unlike the abstainers, these voters wish to actively participate in the decision process. For example, given $(P \wedge Q) \leftrightarrow R$, if an individual believes P to be true but does not know about Q, then her judgment set will be $\{(1,0,0),(1,1,1)\}$.

[1] A *Paretian* aggregation function is such that, if all the individuals in the group adopt the same position on a certain issue, this position will be adopted at the collective level as well.

[2] An aggregation function is *oligarchic* if, for every issue in the agenda, the group adopts a position 0 (resp. 1) if and only if all the members of a subset of the group (the oligarchy) adopt position 0 (resp. 1) on that issue. Clearly, when there is only one individual in the oligarchy, it corresponds to dictatorship.

Weighted criteria

Borrowing the terminology from multi-criteria decision making, we will refer to the premises as *criteria* for the decision. Like multiple criteria decision methods, group members in a judgment aggregation setting evaluate a candidate on the basis of a finite number of premises. However, the two fields have some differences. The first is that multi-criteria decision methods study how a group can select (possibly) one candidate from a set of alternatives by attributing weights to each criterion. Instead, in judgment aggregation, group members are asked to express their judgments on propositions that refer to a single candidate per time. The second noticeable difference is in the distinction between criteria/premises and conclusion in judgment aggregation, which is missing in multi-criteria literature, as well as the logical interrelations among those propositions.

The main novelty we introduce in the judgment aggregation framework is that in our model group members assign *weights* to the criteria in the decision rule. As we will see, this amounts to allocate a *confidence score* to the rule governing the decision. Group members not only participate in the decision process but they also express how well-suited the adopted rule is for the decision at hand.

So far the only approach that resembles ours is that of Dietrich and List (2005), where they investigate judgment aggregation problems using quota rules. A threshold is fixed for each proposition in the agenda and a proposition is collectively accepted only if the number of group members accepting it is at least equal to the threshold for that proposition. The motivation for quota rules was to capture many real-world decision-making situations, where the propositions may differ in status and importance, which is taken into consideration by fixing different threshold values.

We sympathize with that approach but we want to go further. Instead of having an a priori fixed threshold for accepting each proposition, we want to allow the group members to state their individual views on the relevance of each criterion to the decision. In this way, people can also express how much they agree with (resp. dissent from) the rule, by assigning high (resp. low) weights to the criteria. The closer the sum of the weights is to 1, the more an individual believes that the rule is the correct one to assess that decision problem. If, on the other hand, she assigns low weights, it means that she deems that the more important criteria have been overlooked or dismissed.

As an example, consider the board of a research funding agency whose members have to decide which research project to support on the basis of three criteria: quality (P), originality (Q), and applicability (S). Assume as well that the applicability criterion has been introduced only recently following some new regulation that impose all research funding agency to be evaluated on the basis of likeness to attract the interest of private funding. If a good part of the board members dissent with the criterion S because they believe that this will damage pure theoretical projects to the benefit of pure applied ones, they will cast their votes on the propositions, but assign a very low weight to S. This will be reflected at the end of the process, when a certain decision will be made, but also the information about how the

group views the criteria selected for the rule will be publicly available.

General Framework

A formula representing judgment aggregation has the form

$$P \leftrightarrow R,$$

where P is a general propositional formula built on literals representing criteria and R is a literal representing the conclusion. Since any propositional formula P can be written in a disjunctive form, i.e. $P = (P_1 \wedge \cdots \wedge P_n) \vee (P'_1 \wedge \cdots \wedge P'_m) \vee \cdots$ then handling $P \leftrightarrow R$ turns to handling $(P_1 \wedge \cdots \wedge P_n) \leftrightarrow R$ or $(P'_1 \wedge \cdots \wedge P'_m) \leftrightarrow R$ or $\cdots$. Indeed we describe the judgment aggregation procedure corresponding to

$$(P_1 \wedge \cdots \wedge P_n) \leftrightarrow R \quad (1)$$

where P_i are criteria and R is the conclusion.
The other decision rules $(P'_1 \wedge \cdots \wedge P'_m) \leftrightarrow R$, $\cdots$ are treated in a similar way.

In the following (1) is referred to as the *decision rule*. Let us now formalize the extensions we intend to give to classical judgments aggregation, namely: *confidence in the decision rule*, *judgment status* and *legitimacy of the result*.

Confidence in the decision rule

The confidence in the decision rule is represented locally at the level of each criterion. A weight $\alpha_{ij} \in [0, 1]$ associated to a criterion P_i expresses how much P_i is relevant for the conclusion in the decision rule for the member j[3]. Thus (1) is generalized as follows:

$$(P_1, \alpha_{1j}) \Lambda \cdots \Lambda (P_n, \alpha_{nj}) \text{ iff } R, \quad (2)$$

with $0 \leq \alpha_{1j} + \cdots + \alpha_{nj} \leq 1$, where Λ stands for the conjunction between weighted criteria.

Note that when $\alpha_{ij} = 0$, the judgment corresponding to the associated criterion P_i is simply ignored and the value of R is decided only using the remaining criteria. This is intuitively meaningful since $\alpha_{ij} = 0$ means that the member j judges that the criterion P_i should not be considered.

Depending on the values of the weights $\alpha_{1j}, \cdots, \alpha_{nj}$, we distinguish the following cases:

a) Full agreement "$\alpha_{1j} + \cdots + \alpha_{nj} = 1$".
This means that, for member j, either the criteria $P_1, \cdots, P_n$ are the all and only relevant ones to make a judgment on R, or they *include* all the relevant criteria together with some completely irrelevant ones. Thus j completely agrees on (1). Indeed (2) reduces to (1) such that P_i is ignored if $\alpha_{ij} = 0$, i.e. the decision rule is $(P_1 \wedge \cdots \wedge P_{i-1} \wedge P_{i+1} \wedge \cdots \wedge P_n) \leftrightarrow R$.

It is worth observing that the original legal paradox of judgment aggregation is an instance of the full agreement case, where all group members (the judges) have to fully endorse the legal code, or behave as if this is the case.

b) Partial agreement "$\alpha_{1j} + \cdots + \alpha_{nj} < 1$".
This case means that member j doesn't fully agree on the decision rule, i.e. she deems that (all or some of) the relevant criteria have been dismissed (and, possibly, that the rule includes some irrelevant criteria for the decision).

Example 1 *Let us consider the example of the board of a research funding agency again. We recall that the decision on whether to support a research project is taken by looking at three criteria: quality (P), originality (Q), and applicability (S). The five members state their judgments on P, Q and S as in Table 2.*

	P	Q	S
M_1	(0, .33)	(0, .33)	(1, .34)
M_2	(1, .3)	(1, .3)	(1, .4)
M_3	(0, .5)	(0, .5)	(0, 0)
M_4	(1, .3)	(1, .3)	(0, 0)
M_5	(1, .2)	(1, .1)	(1, .1)

Table 2: Individual judgments and weights assignments on the criteria.

The first two members deem the criteria P, Q, and S to be the all and only relevant attributes for funding a project. Since for them $\alpha_{1j} + \alpha_{2j} + \alpha_{3j} = 1$ (for $j = 1, 2$), they fully agree with the decision rule $(P \wedge Q \wedge S) \leftrightarrow R$. The third member also fully agrees with the rule but, unlike the first two, she believes that P and Q are the only relevant criteria and S is completely irrelevant for the decision. Like the third agent, M_4 thinks that the applicability criterion should not play a role in the decision of which project to fund. However, she believes that one or more relevant criteria have not been taken into consideration ($\alpha_{14} + \alpha_{24} + \alpha_{34} < 1$). Finally, M_5 agrees with M_4 that other important criteria for the decision have been ignored, but she thinks that the applicability of a project should be taken into account, though it is not a very important aspect for the final decision.

The criteria weights should play a role in the way group members express their judgments on the conclusion. Moreover, the information about how relevant the members deem the criteria to be for the decision has to be taken into account when the individual judgments are aggregated to derive a collective decision.
We distinguish the following sub-cases depending on the level of global non-agreement on the decision rule. Let t $(1 > t > 0)$ be a threshold.

b.1) High partial agreement "$t \leq \alpha_{1j} + \cdots + \alpha_{nj} < 1$".
Even if member j does not fully agree on the decision

[3] It is important to notice that α_{ij} does not express how much a member is confident when expressing her judgment w.r.t. a criterion P_i, but how much member judges that P_i is relevant in the decision rule.

rule, she believes that this includes enough relevant criteria ($\alpha_{1j} + \cdots + \alpha_{nj} \geq t$). This means that j deems the rule sufficiently appropriate to decide R on the basis of the given criteria. Hence, the judgment on the conclusion is obtained following the given rule: (2) reduces to (1) such that P_i is ignored if $\alpha_{ij} = 0$.
Note that **a** and **b.1** can be both represented by $t \leq \alpha_{1j} + \cdots + \alpha_{nj} \leq 1$.

b.2) Low partial agreement "$0 \leq \alpha_{1j} + \cdots + \alpha_{nj} < t$". In this case, the confidence in the decision rule is very low, i.e. criteria P_i are not adequate or some very important criteria are missing. In this case, the member fixes the value of R according also to the missing criteria. The intuition is that, if an individual wants to have her saying in a decision process, but considers the adopted rule unable to capture the relevant criteria for the decision, she must be able to express her judgment on the conclusion while making explicit that she deems the rule to be not completely appropriate. Note, however that if a group member assigns a judgment 0 to a criterion P_i and $\alpha_{ij} \neq 0$ then R should be 0. This is to ensure coherence. Indeed the decision rule for that member is $(P_1 \wedge \cdots \wedge P_m \wedge T_1 \wedge \cdots \wedge T_l) \leftrightarrow R$, where $P_1, \cdots, P_m$ are criteria whose associated α_{ij} is different from 0 and $T_1, \cdots, T_l$ are missing criteria. Since only criteria P_i are present in the decision process, we may for example have $R = 0$ while $P_1 = \cdots = P_m = 1$[4] because the judgment of the group member j is 0 w.r.t. at least one missing criterion. Now, if at least one P_i has a judgment 0 then R is necessarily equal to 0. Indeed a group member is free to fix the value of R only in case $0 \leq \alpha_{1j} + \cdots + \alpha_{nj} < t$ and all criteria whose associated weight is different from 0 are assigned a judgment 1.

The weights $\alpha_{1j}, \cdots, \alpha_{nj}$ are then used to compute the confidence score CS_j of the rule for each group member j; namely $CS_j = \alpha_{1j} + \cdots + \alpha_{nj}$.

Example 2 *Following* **b.1** *and* **b.2***, the table below summarizes the judgments of the members of our funding board example:*

	P	Q	S	CS	R
M_1	(0, .33)	(0, .33)	(1, .34)	1	0
M_2	(1, .3)	(1, .3)	(1, .4)	1	1
M_3	(0, .5)	(0, .5)	(0, 0)	1	0
M_4	(1, .3)	(1, .3)	(0, 0)	.6	1
M_5	(1, .2)	(1, .1)	(1, .1)	.4	0

Table 3: Individual judgments on all propositions in the agenda.

[4] $P_i = 1$ is a short hand notation to express that the judgment w.r.t. criterion P_i is 1.

The first three members fully agree with the decision rule $(P \wedge Q \wedge S) \leftrightarrow R$. Hence, the judgment on the conclusion R is logically derived from the values assigned to P, Q, S and the rule. Things are different for the last two group members. Suppose that $t = .5$. M_4 assigned zero weight to S but her CS is above the threshold. Hence, M_4 assigns R a value according to the values of only P and Q and the decision rule. Instead, M_5 has a low CS in the rule. In order to capture the intuition that she should cast her vote on R sincerely (according to what she considers the missing criteria), M_5 can decide whether the research project without following the rule. For instance, she can refuse the funding ($R = 0$).

When $CS_j = \alpha_{1j} + \cdots + \alpha_{nj} = 0$ the judgments of the group member j are ignored but this is not considered as an abstention since the given α_{ij} are considered in the aggregation process, as explained below.

Judgment status

We distinguish three possible judgments: classical binary judgment 1 (for) or 0 (against), neutral judgment and abstention. As classical binary judgment is already used, we only detail abstention and neutral judgments.

- **Neutral judgment** We represent a neutral judgment by a question mark "?" which is interpreted as the judgment may be 1 or 0. A group member may express a neutral judgment w.r.t. some or all criteria (and - possibly - on the conclusion as well).
- **Abstention** In case of abstention, a group member does not give any judgment on $P_1, \cdots, P_n$ (and by consequence no values on $\alpha_1, \cdots, \alpha_n$) and R. Abstainers are not taken into account in the aggregation process but in the computation of the legitimacy of the conclusion (see below).

Legitimacy

The legitimacy, denoted lg, expresses to what extent the decision process is reliable. It is equal to the total number of voters over the number of authorized people to vote (i.e $0 \leq lg \leq 1$). The closer lg is to 1, more reliable the process is. The legitimacy does not play a role in the final outcome. However the legitimacy level may declare the decision outcome invalid. But this comes in a second step. First we aggregate using our aggregation procedure that we will present in the next section and then legitimacy considerations can play a role.

Representation and aggregation procedure

We represent a judgment expressed by a member j by the following tuple

$$J_j = ((P_{1j}, \alpha_{1j}), \cdots, (P_{nj}, \alpha_{nj}), R_j, CS_j),$$

$P_{ij}, R_j \in \{0, 1, ?\}$ and $\alpha_{ij}, CS_j \in [0, 1]$.

Note that CS_j can be computed from $\alpha_{1j}, \cdots, \alpha_{nj}$ (we have $CS_j = \alpha_{1j} + \cdots + \alpha_{nj}$). However we include it in

J_j for simplicity reading since it allows to see whether we apply the decision rule or not. This remarks also holds for CS_{agg} in D given below.
R_j is either derived following the decision rule or fixed by the group member depending on whether the confidence rule CS_j is above the threshold or not. In case of abstention we write $J_j = (X, \cdots, X, X, X)$.

Given a set of judgments $\{J_1, \cdots, J_k\}$, the collective decision is represented as follows [5]:

$$D = ((P_{agg_1}, \alpha_{agg_1}), \cdots, (P_{agg_n}, \alpha_{agg_n}), R_{agg}, CS_{agg}, lg),$$

such that:

- P_{agg_i} is the majority of $P_{i1}, \cdots, P_{ik}$ (with $\alpha_{ij} \neq 0$) following proposition-wise majority voting. So neutral judgments simply follow the majority.
  In case of indecision i.e. a tie between the number of $P_{ij} = 1$ and $P_{ij} = 0$, compute the sum of α_{ij} associated to $P_{ij} = 1$ and the sum of α_{ij} associated to $P_{ij} = 0$ taken individually and follow the judgment corresponding to the greatest sum. In this case, neutral judgments follow the will of judgments having a greater weight. In case of a tie, we put $P_{agg_i} =?$. Note that this is the only extreme case where our approach does not solve the indecision.
- α_{agg_i} (resp. CS_{agg}) is a numerical aggregation of $\alpha_{i1}, \cdots, \alpha_{ik}$ (resp. $CS_1, \cdots, CS_k$). In this paper, we use the average function but any other numerical aggregation function may be used as well.
  Note that $CS_{agg} = \alpha_{agg_1} + \cdots + \alpha_{agg_n}$. This is important since it means that expressing the confidence in the decision rule or relevance of each criterion leads to the same result.
- R_{agg} is computed by premise-based or conclusion-based procedure. The procedure is chosen w.r.t. CS_{agg} and t:
  - if $CS_{agg} < t$ then we use conclusion-based procedure and R is computed on the basis of $R_1, \cdots, R_k$. This is intuitively meaningful since $CS_{agg} < t$ means that the group members thought that the decision rule was not the right one for that decision, so the only reasonable thing they can say is the final conclusion, without giving reasons for that. R_{agg} is calculated by simple majority voting. In case of indecision we compute the sum of CS_j for which $R_j = 1$ and the sum of CS_j for which $R_j = 0$. Then we follow the judgment associated to the greatest sum. In case of a tie, we put $R_{agg} =?$. Again this is the only extreme case where indecision is not solved.
  - if $CS_{agg} \geq t$ then R_{agg} is computed following premise-based procedure. In fact $CS_{agg} \geq t$ means that we agree on the decision rule. Having $CS_{agg} = \alpha_{agg_1} + \cdots + \alpha_{agg_n}$ consolidates us in this choice since if the only information we have is $(P_{agg_1}, \alpha_{agg_1}), \cdots, (P_{agg_n}, \alpha_{agg_n})$ then we first compute $\alpha_{agg_1} + \cdots + \alpha_{agg_n}$. Then if the sum is above the threshold, we use the decision rule. The indecision is handled in the same way as in the previous item replacing CS_j by α_{agg_i}.
- lg is the legitimacy. It is equal to the total number of voters over the number of authorized people to vote.

We now illustrate the procedure with our running example.

Example 3 *Table 4 gives judgments expressed by five members of our funding board. Let $t = .8$. Since $CS_{agg} = .86 \geq .8$ we use premise-based procedure. We get $R = 1$. The legitimacy of this decision is equal to $4/5 = .8$. We can observe that the collective decision of the members of the funding board agree on the three criteria but with a very low confidence about the relevance of Applicability (S).*

	P	Q	S	CS	R
M_1	(1, .5)	(0, .5)	(?, 0)	1	0
M_2	(?, .4)	(1, .4)	(1, .1)	.9	?
M_3	X	X	X	X	X
M_4	(1, .3)	(1, .4)	(?, .1)	.8	?
M_5	(1, .4)	(1, .3)	(1, .05)	.75	1
collective decision	(1, .4)	(1, .4)	(1, .06)	.86	1

Table 4: Example of judgment aggregation with confidence scores.

Example 4 *Let us now consider individuals who have to make a collective decision using the rule $(P \wedge Q) \leftrightarrow R$, which they think is not appropriate, i.e. $CS < t$. Suppose that their judgments are as in Table 5 and that $t = .5$.*

	P	Q	CS	R
M_1	(1, .1)	(1, .2)	.3	0
M_2	(0, .1)	(1, .1)	.2	0
M_3	(1, .2)	(0, .2)	.4	0
M_4	(1, .2)	(1, .1)	.3	1
M_5	(0, .2)	(0, .2)	.4	0
collective decision	(1, .16)	(1, .16)	.32	0

Table 5: Example of judgment aggregation with low confidence scores.

All five members assign a very low confidence score to the decision rule. As we have seen, a low CS means that the group members believe that the most important criteria for the decision are missing. Therefore, they express their judgments on the criteria in the rule, but their decision on the conclusion R takes into account what they believe are the missing attributes. For example, M_1 states that $R = 0$ despite the fact that $P = 1$ and $Q = 1$. This suggests that M_1 considers that other criteria are not satisfied, so she is against R. In this situation, the group will conclude $R = 0$ and they will be able to provide only partial reasons in support of their decisions. The group cannot reach a decision

[5] The aggregation procedure doesn't assume that all members have to agree on the decision rule. Our extended framework allows each member to follow the rule or not depending on the weights she assign to the criteria.

by proposition-wise majority voting on the criteria, as P and Q do not exhaust the reasons for or against R.

Notice that the above line of reasoning also illustrates why our framework is less sensitive to paradoxical outcomes.

Classical judgment aggregation vs judgment aggregation with rule confidence score

In this section we compare how our approach behaves compared to standard judgment aggregation. More precisely, we describe the behavior of our approach and show that in extreme cases we recover the classical judgment aggregation procedure.

In our approach, the threshold governs our decision on whether we use the decision rule or not when computing R_j and also R_{agg}. The lower t is fixed, the more the group members are forced to adopt the decision rule, and the collective decision will be driven by the judgments on the criteria. This is coherent with our proposal to use premise-based procedure when $CS \geq t$. Symmetrically, setting a high t exposes the collective decision to be driven by the individual judgments on the conclusion, i.e. possibly less criteria will be considered important in the decision process. Again this is coherent with our proposal to use conclusion-based aggregation procedure when $CS < t$.

Our approach supposes to fix the value of the threshold t in order to choose the relevant aggregation function. The problem of setting a threshold is common to all frameworks that use such quantitative approach (see for example, the work by (Dietrich & List 2005) using quota rules). The following questions can then be asked: who fix the threshold and when ? In this work, we didn't pretend to answer these open questions since our aim is to study how weights can be used to model the confidence a member has in the decision rule. On the other hand, if all members disagree on the rule, we can conclude that the decision rule is note adequate. This can allow, for example, to update the next JA for other candidates by analysing which are the common irrelevant criteria in order to remove them from the decision rule.

The proposed extension has a nice behavior since it reduces into classical judgment aggregation procedure when basic hypothesis are considered. More precisely suppose that all weights are equal and $\alpha_{1j} + \cdots + \alpha_{nj} = 1$ (for $j = 1, \cdots, k$), there is no threshold and there are neither neutral judgments nor abstainers. In such a case:

(i) the notion of legitimacy is no longer meaningful since there are no abstainers,

(ii) P_{agg_i} is the majority of $P_{i1}, \cdots, P_{ik}$ as it is the case in classical judgment aggregation procedure,

(iii) $CS_j = \alpha_{1j} + \cdots + \alpha_{nj} = 1$ means that the group member j fully agrees on the decision rule so R_j is computed following the decision rule as it is the case in classical judgment aggregation procedure,

(iv) Since there is no threshold we recover the dilemma on whether we use premise-based or conclusion-based aggregation procedure.

In addition, our aggregation procedure verifies some desirable properties such as *anonimity*, *no dictatorship* and *universal domain*. *Anonimity* and *no dictatorship* are the same as in standard judgment aggregation. The first property requires that all group members who participated in the decision process (i.e. excluding abstainers) have equal weight in the aggregation. The absence of a dictator guarantees that there exists no single individual that always determines the collective decision. *Universal domain* guarantees that our aggregation rule takes the set of all *admissible* individual judgment sets and assigns a collective judgment set. In our framework, a judgment set is admissible if the weights of P_i, the judgments on the criteria P_i and on the conclusion R are assigned accordingly to the decision rule, the CS in the rule, and the individual judgment status.

Clearly, the controversial *independence of irrelevant alternatives (IIA)* condition is not satisfied by our aggregation procedure. Depending on whether $CS \geq t$ or $CS < t$, our aggregation turns to premise-wise or conclusion-based procedure. Not satisfying the IIA condition, our approach provides an escape from the impossibility results plaguing standard judgment aggregation.

Conclusion and Future Work

We extended classical judgment aggregation procedure in order to offer a more realistic framework, and to escape the impossibility results. We introduce two main changes. Firstly, we define judgment status where a member can abstain or give binary or neutral judgments. Secondly, members assign weights to the criteria. A confidence score is computed on the basis of these weights. It expresses how well-suited a group member thinks that the adopted rule is for the decision process. This new representation of criteria allows us to avoid most cases of indecision by using specific decision rule (conclusion-based or premises-based) according to the value of the confidence score. Lastly, we introduce the notion of legitimacy that expresses to what extent the conclusion tends for a consortium.

This work can be extended in different directions:

(i) refine the notion of abstention by allowing abstention at the level of a criterion and study its impact on the legitimacy of the decision. This is appropriate when the weight associated to a criterion is equal to 0,

(ii) extend our framework in order to treat extreme cases of indecision,

(iii) investigate the relationship between criteria having the highest weight in our framework and works on coalitions (Shehory & Kraus 1998). More precisely, we intend to study how group members can form coalitions and manipulate their confidence scores in order to drive the decision process in a particular direction.

(iv) Lastly, investigate the relationship with opinion aggregation in order to go beyond binary judgments (Ben-Arieh & Chen 2007).

Acknowledgments

Souhila Kaci has partially been supported by the French National Research Agency (Agence Nationale de la Recherche) under contract n^0 NT05-4_41833.

References

Ben-Arieh, D., and Chen, Z. 2007. Linguistic group decision-making: opinion aggregation and measures of consensus. *Fuzzy Optimization and Decision Making* 5(4):371–386.

Bovens, L., and Rabinowicz, W. 2006. Democratic answers to complex questions. an epistemic perspective. *Synthese* 150:131–153.

Chapman, B. 2002. Rational aggregation. *Politics, Philosophy and Economics* 1(3):337–354.

Dietrich, F., and List, C. 2005. Judgment aggregation by quota rules. *Journal of Theoretical Politics*.

Dietrich, F., and List, C. 2006. Arrow's theorem in judgment aggregation. *Social Choice and Welfare*.

Dietrich, F. 2006. Judgment aggregation: (im)possibility theorems. *Journal of Economic Theory* 126(1):286–298.

Dokow, E., and Holzman, R. 2007. Aggregation of binary evaluations with abstentions. *Working paper*.

Figueira, J.; Greco, S.; and Ehrgott, M., eds. 2004. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer.

Gärdenfors, P. 2006. A representation theorem for voting with logical consequences. *Economics and Philosophy* 22:181–190.

Keeney, R., and Raiffa, H. 1976. *Decision with Multiple Objectives: Preferences and Value Trade-Offs*. John Wiley, New York.

Kornhauser, L., and Sager, L. 1986. Unpacking the court. *Yale Law Journal* 96:82–117.

Kornhauser, L., and Sager, L. 1993. The one and the many: Adjudication in collegial courts. *California Law Review* 81:1–51.

List, C., and Pettit, P. 2002. Aggregating sets of judgments: An impossibility result. *Economics and Philosophy* 18:89–110.

List, C., and Pettit, P. 2004. Aggregating sets of judgments: Two impossibility results compared. *Synthese* 140(1-2):207–235.

List, C. 2007. Judgment aggregation - a bibliography on the discursive dilemma, the doctrinal paradox and decisions on multiple propositions. *http://personal.lse.ac.uk/LIST/doctrinalparadox.htm*.

Pauly, M., and van Hees, M. 2006. Logical constraints on judgment aggregation. *Journal of Philosophical Logic* 35:569–585.

Pettit, P. 2001. Deliberative democracy and the discursive dilemma. *Philosophical Issues* 11:268–299.

Pigozzi, G. 2006. Belief merging and the discursive dilemma: an argument-based account to paradoxes of judgment aggregation. *Synthese* 152(2):285–298.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence* 101(1-2):165–200.

Integrating Golog and Planning: An Empirical Evaluation

Jens Claßen[†] and **Viktor Engelmann**[†] and **Gerhard Lakemeyer**[†] and **Gabriele Röger**[‡]
[†] Dept. of Computer Science, RWTH Aachen University, Germany
[‡] Dept. of Computer Science, University of Freiburg, Germany

Abstract

The Golog family of action languages has proven to be a useful means for the high-level control of autonomous agents, such as mobile robots. In particular, the IndiGolog variant, where programs are executed in an online manner, is applicable in realistic scenarios where agents possess only incomplete knowledge about the state of the world, have to use sensors to gather necessary information at runtime and need to react to spontaneous, exogenous events that happen unpredictably due to a dynamic environment. Often, the specification of such an agent's program also involves that certain subgoals have to be solved by means of planning. IndiGolog supports this in principle by providing a variety of lookahead mechanisms, but when it comes to pure, sequential planning, these usually cannot compete with modern state-of-the-art planning systems, most of which being based on the Planning Domain Definition Language PDDL. Previous theoretical results provide insights on the semantical compatibility between Golog and PDDL and how they compare in terms of expressiveness. In this paper, we complement these results with an empirical evaluation that shows that equipping IndiGolog with a PDDL planner (FF in our case) pays off in terms of the runtime performance of the overall system. For that matter, we study a number of example application domains and compare the needed computation times for varying problem sizes and difficulties.

Introduction

The Golog (Levesque et al. 1997) family of action languages has already proven to be a suitable means when it comes to the high-level control of autonomous agents, such as mobile robots (Burgard et al. 1998). It provides the programmer with the flexibility to chose the right balance between deterministic, predefined behavior on the one hand and on the other hand leave certain non-deterministic parts to be solved by the system.

The IndiGolog (De Giacomo, Levesque, and Sardina 2001) variant, which is in turn based on ConGolog (De Giacomo, Lespérance, and Levesque 2000), possesses a number of features that makes it particularly suited for many practical scenarios. For one, it works in cases where agents only have incomplete knowledge about the state of the world, and where sensing actions can and most often have to be used in order to gather further information that is required to fulfill the task. Furthermore, so-called exogenous actions reflect changes in a dynamic environment that are not caused by an action of the agent.

For another, an important aspect about the language is that programs are executed in an incremental, online manner. Unlike in the original Golog, where the system searches for an action sequence that constitutes a legal execution of the entire input program, IndiGolog does not apply a general lookahead, but leaves it to the programmer to explicitly state which parts of the program ought to be solved by means of search. This helps to keep program execution tractable, in particular in the presence of incomplete knowledge and possible occurrences of exogenous actions.

However, in many application domains for which IndiGolog is suited in principle, one often encounters subproblems that are rather combinatorial in nature. Typical examples include scheduling currently pending requests to the system, finding a route through a certain topology, or a combination of these two. These are cases for classical planning, where the programmer only provides a list of available actions and a description of the goal state, and it is up to the system to search for an appropriate (and preferably short) sequence of action instances that achieves the goal. While Golog supports this in principle, it soon becomes infeasible for all but the smallest problem sizes.

However, sequential planning has received a lot of attention in recent years. PDDL, the Planning Domain Definition Language (Ghallab et al. 1998) was introduced as the common input language for the systems competing at the biennial International Planning Competition. It extends the well-known STRIPS language (Fikes and Nilsson 1971) by features from Pednault's (1989) ADL. Later extensions include (among other things) durative and concurrent actions (Fox and Long 2003; Edelkamp and Hoffmann 2004) as well as constraints on plan trajectories and preferences among goals (Gerevini and Long 2005). The language has become a de-facto standard for formulating planning benchmarks. Many efficient planners that use it have been developed by now, using a large variety of techniques and heuristics.

It suggests itself to benefit from these developments by embedding such a planner into IndiGolog. The idea is that

whenever a planning subproblem arises during the execution of a Golog program, it is translated into PDDL and the planner is called. The resulting plan is translated back and Golog resumes executing that plan. For the ADL fragment of PDDL, the theoretical foundations for such an embedding have been laid in previous work. Claßen et al. (2007) show that the state updates in PDDL can be understood as progression for a certain form of Golog action theories; Röger, Helmert and Nebel (2008; 2007) identify a maximal class of such theories that are equivalent to the ADL sub-language in terms of expressiveness.

Based on these theoretical results, we extended the current implementation of IndiGolog with the possibility of redirecting planning subgoals to a PDDL planner, in our case the FF system. In this paper, we do an empirical study that shows that such an extension is beneficial in terms of the overall computation time needed by the system. For this purpose we developed a number of example application domains, ran them in simulations and measured the corresponding runtimes for varying problem sizes.

The remainder of the paper is organized as follows. In the following section, we present details concerning the IndiGolog framework, the FF planning system, and our integration of the two. Next, we introduce the application domains that we developed, after which we discuss our experimental setup and the results obtained. We close with a brief conclusion.

Integrating FF into IndiGolog

IndiGolog

The *situation calculus* (McCarthy and Hayes 1969; Reiter 2001) is a dialect of first-order logic (with some second-order extensions) for reasoning about dynamic domains. Changes in the world are assumed to be the result of *primitive actions*, which are performed by some implicit agent and modelled by terms like $move(l_1, l_2)$. Properties that are affected by performing such actions are called fluents, which can be predicates like $Holding(obj_2, s)$ or functions like $position(robot, s)$. The last argument of a fluent is a *situation*, which should be understood as the current history of actions that have been executed. The constant S_0 is used to denote the initial situation, and when a is an action and s a situation, then $do(a, s)$ denotes the situation that results from performing a in s. For example $Holding(letter, do(pickup(letter, S_0)))$ means that the agent is holding the letter after picking it up. A particular domain is described by a *basic action theory*, which is a set of situation calculus formulas that define the fluents' values in the initial situation and preconditions and effects of actions.

Based on the situation calculus, the members of the Golog (Levesque et al. 1997) family of languages allow the definition of complex actions, also called *programs*. The ConGolog (De Giacomo, Lespérance, and Levesque 2000) variant supports the following constructs:

α	primitive action
$\phi?$	test
$\delta_1; \delta_2$	sequence
$\delta_1 \mid \delta_2$	nondeterministic choice
$\pi x.\delta(x)$	nondeterministic choice of argument
δ^*	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	loop
$\delta_1 \| \delta_2$	concurrent execution
$\delta_1 \rangle\!\rangle \delta_2$	prioritized concurrency
$\delta^{\|}$	concurrent iteration
$\langle \vec{x} : \phi(\vec{x}) \rightarrow \delta(\vec{x}) \rangle$	interrupt
$P(\vec{t})$	procedure call

Apart from conditionals, loops and recursive procedures, which are common to imperative programming languages, an important aspect is that parts of a program can also be nondeterministic. For instance, $\delta_1 \mid \delta_2$ means to do either δ_1 or δ_2, and δ^* performs δ zero or more times. The idea is that a program does not represent a complete solution to the problem, but only a sketch of it, where the nondeterministic parts constitute gaps which have to be filled by the system.

ConGolog models concurrency as (nondeterministic) interleavings of the involved processes, i.e. actions are always performed one at a time. This includes so-called *exogenous actions*, which are used to model spontaneous changes in the dynamic environment that do not constitute (direct) effects of the agent's actions. Interrupts can be used to define reactive responses that are triggered when such an event occurs or some other condition is met; the normal program execution then continues afterwards.

Programs are executed off-line in ConGolog. This means that the interpreter first analyzes the entire program to find a conforming execution trace before the first action is actually executed in the real world. This soon becomes a problem when the program is large; furthermore in many scenarios, the agent has only incomplete knowledge about its environment, making it necessary to gather information at runtime. IndiGolog (De Giacomo, Levesque, and Sardina 2001) is an extension where these issues are tackled. Programs are executed on-line, which means that there is no general lookahead; the interpreter simply executes the next possible action in each step (treating nondeterminism like random choices). A new operator $\Sigma(\delta)$ is introduced which has to be used to explicitly mark subprograms which have to be solved by means of search. This does not represent a loss of generality since one might encapsulate the entire program, but gives the programmer much more control over where the system spends its computational effort. In addition, programs may contain *sensing actions* for acquiring needed information at runtime. When such an action is executed, a sensing result is obtained (which normally is the current value of some fluent) and used to update the agent's knowledge base. Thus, a subsequent choice in the program that depends on this sensed value can be made on-line.

The features mentioned above make IndiGolog applicable in many practical scenarios, where we often are confronted with dynamically changing environments, where sensing has to be used to fill gaps in the agent's information about the world, and where computational power usually is limited. A PROLOG-based implementation of an IndiGolog agent architecture is available at Sourceforge[1]

[1] http://sourceforge.net/projects/indigolog/

and has already been successfully applied for controlling robots of the LEGO MINDSTORMS system, the ER1 EVOLUTION robot and other software agents.

Golog is therefore an appropriate means for the overall control of an agent in many application domains. However, the general task often involves certain combinatorial subproblems, like finding a route through a certain topology, scheduling currently pending requests or a combination of these two. When one is unable (or it is to tedious) to specify some (partially nondeterministic) program to restrain the space of possible execution traces, these examples correspond to classical planning tasks where only the current state, a goal, and a set of available actions are provided, and where it is the job of the system to search for an action sequence that achieves that goal.

In principle, even classical planning can be done in Golog with a completely nondeterministic program as follows:

while $(\neg Goal)$ **do** $(\pi a)\ a$ **endWhile**.

A (successful) execution trace of this program corresponds to a plan that reaches a situation where $Goal$ holds. However, since the Golog interpreter uses the PROLOG backtracking mechanism to resolve nondeterminism, performing planning like this basically amounts to do a blind search. The IndiGolog system further contains a number of built-in planning mechanisms, but these are merely proof of concepts for different kinds of conditional planning (Sardina et al. 2004), which use very basic, unguided search strategies. In any of these cases, planning soon becomes infeasible for all but the smallest problem sizes.

The FF Planning System

A large research community engaged in developing sophisticated techniques and domain-independent heuristics for solving classical planning problems has evolved in the last decade, where the PDDL language has become a de-facto standard for formalizing benchmarks that allow the comparison of different approaches.

The FF Planning System developed by Hoffmann (2001) is a fully automated system for classical planning. It supports the ADL fragment of PDDL which is sufficient for our purposes. Furthermore, it has proven its quality by winning the automated track of the planning competition in 2000 and still being part of state-of-the-art planning systems that support a wider fragment of PDDL.

FF performs a forward search in the state space, guided by a heuristic function that is automatically extracted from the domain description. The heuristic gets derived from the corresponding *relaxed* planning task that results from the original one by ignoring the delete effects of the actions. This relaxed task can be solved in polynomial time and the number of actions in the resulting plan provides an estimate for the goal distance in the original task. Furthermore, the relaxed task is used to identify so-called *helpful actions* that are expected to be a good choice for the next action and which hence are considered first during search.

The search method used by the FF system is a variant of hill-climbing called *enforced* hill climbing. The main difference to the standard algorithm is that in the case of a plateau it switches to best first search until a node with a strictly better evaluation is found.

Integration

We embedded the FF planning system into the IndiGolog framework to benefit from these developments. For that purpose, we introduced a new program construct `achieve_ff(G,A)`, where `G` is a goal formula and `A` is a list of actions. Whenever an `achieve_ff` statement is encountered during the execution of a program, it causes the current state, the goal and the available actions to be translated into a PDDL planning problem which is referred to FF. The resulting plan is translated back and Golog continues with executing that action sequence.

The semantical soundness of this proceeding has been laid in previous work. In a first paper (Claßen et al. 2007) it was shown that the state updates in PDDL's ADL fragment (i.e. the language we obtain when only the `:adl` requirement flag is set) can be understood as progression for a certain form of Golog action theories; in two further papers (Röger and Nebel 2007; Röger, Helmert, and Nebel 2008) a maximal class of such theories is identified that are equivalent to the ADL sub-language in terms of expressiveness. ADL extends basic STRIPS with conditional effects as well as negated, disjunctive and quantified preconditions.

We will not discuss the theoretical details here, but provide an intuition by means of an example translation, taken from one of our test applications. A PDDL planner requires two files: a *domain description*, containing types, predicates and operator definitions, and a *problem description*, which specifies the objects in the domain, the initial values of predicates and the goal formula.

Since in PDDL the closed world assumption holds, fluents and action parameters can only take values from a finite set of object constants. These may be divided into types and subtypes that are used to restrict the possible values of parameters. We use unary PROLOG predicates with finite extensions to represent types and declare the subtypes of the general supertype `object` as follows:

```
object(X) :- passenger(X) ; floor(X).
```

In general, subtypes can of course be further subdivided, using similar clauses. These Golog declarations are directly mapped to type declarations in the PDDL domain file:

```
(:types passenger floor - object)
```

A relational fluent with type restrictions on its arguments is in the following way declared in the Golog axiomatization:

```
rel_fluent(lift_at(F)) :- floor(F).
```

The compiled PDDL domain definition then contains:

```
(:predicates
   (lift_at ?f - floor) ...
```

An action is given by a declaration (with type restrictions), a precondition and a number of positive and negative effect axioms:

```
action(move(F1,F2)) :-
 floor(F1), floor(F2).
```

```
poss(move(F1,F2),
 and(lift_at(F1),
     or(above(F1,F2),above(F2,F1)))).

causes_false(move(F1,F2),
             lift_at(F1),true).
causes_true(move(F1,F2),
             lift_at(F2),true).
```

The last argument of an effect axiom may contain a condition that must hold for the effect to actually take place. In case of `true`, the translation is straightforward:

```
(:action move
 :parameters
  (?f1 - floor ?f2 - floor)
 :precondition
  (and (lift_at ?f1)
       (or (above ?f1 ?f2)
           (above ?f2 ?f1)))
 :effect
  (and (lift_at ?f2)
       (not (lift_at ?f1)))
)
```

The following example shows how an effect is translated that involves a non-trivial condition and additional variables that are not arguments of the action:

```
causes_true(stop(F),
            boarded(P), origin(P,F)).
```

Such conditions result in conditional effects in the PDDL action (the type of the quantified variable is determined on the basis of the type definition of the arguments of `origin`):

```
(forall (?p - passenger)
    (when (origin ?p ?f)(boarded ?p)))
```

To generate the actual planning instance, we need to collect all objects of the involved types, e.g.

```
passenger(p1). passenger(p2).
floor(f1). floor(f2). floor(f3).
```

and declare them in the PDDL problem file:

```
(:objects p1 p2 - passenger
          f1 f2 f3 - floor)
```

We then determine which fluent ground atoms `F` evaluate to `true` given the current action history `H`, i.e. we collect all solutions of `has_value(F,H,true)`. For instance, if `lift_at(f2)` is one such atom, then the `:init` section of the problem file contains

```
(:init  (lift_at f2)
        ... )
```

Finally, the problem description contains the translation of the goal formula `G`. For example, a goal formula `all(p,passenger,served(p))` results in

```
(:goal (forall (?p - passenger)
               (served ?p)))
```

We remark that of course the specific, restricted form of clauses in the Golog action theory is only required for those parts that are relevant for the planning problem. The axiomatization may contain additional parts that PDDL does not understand, in particular exogenous and sensing actions.

Benchmark Domains

We designed three example application domains. The first two examples are representatives of so-called transportation domains (Helmert 2008). The characterizing property of such problems is that there are portables that should be transported from their origin to their destination location using mobiles that can move between some of the locations. This type of problem is especially interesting because such tasks arise very often in practice. This is probably also the reason why a large fraction of the benchmarks used in the International Planning Competitions is among these domains. The most interesting aspect of our last example domain is that it in addition involves sensing. In the following we will briefly introduce each of these domains.

A Logistics Domain

The first domain that we studied serves as a representative for all kinds of logistics applications. The task is to transport packages to their destination locations, using a number of trucks which can only hold one package at a time. The direct connections between locations form a (not necessarily complete) graph structure.

The domain has the dynamic aspect that new packages keep arriving at runtime, represented by exogenous actions, and have to be picked up and delivered in turn.

An Elevator Domain

The second test domain has been inspired by the miconic elevator domains of the International Planning Competition in 2000. There is an elevator moving between the floors of a building. At some floors passengers are waiting and should be transported to their respective destination floor. During the program execution new passengers arrive randomly.

There are three sorts of actions that can be used to serve the passengers: Movement actions move the elevator from one floor to an adjacent floor. Since an elevator can move faster if it does not have to stop at each floor, there are also actions for fast movements that overcome two floors within one step. The third sort of actions are the stop actions which cause all passengers waiting at the current floor to enter the elevator and drop off all boarded passengers whose destination is the current floor.

A Mail Delivery Robot Domain

The third domain is a variant of a common application example (Tam et al. 1997) of a mobile robot operating in an office environment, where it has to deliver letters and parcels between the workers' mailboxes. Here, the structure of the building is assumed to consist of a number of hallways, which are connected (e.g. by an elevator) to other hallways, and where there is a certain number of offices at each hallway. Each office may contain one or multiple different mailboxes, each of which serving for both incoming and outgoing mails.

This domain involves sensing since the robot must look into a mailbox in order to find out how many and which letters it currently contains. Furthermore, before the agent actually knows where to deliver a letter, it has to pick it up and read off the addressee.

Experiments and Results

For our experiments, we further have extended the IndiGolog framework by a simple simulator that plays the role of the outside world. It runs in a separate instance of PROLOG and communicates with Golog via TCP/IP sockets. The basic idea is to keep track of the (relevant part of the) world state using PROLOG's `assert` and `retract` mechanism. When an exogenous actions occurs and what sensing result is returned after the execution of an action can then be defined as conditions wrt to the simulated state. All experiments were performed on a PC with an Intel Core 2 Duo E6750 CPU running at 2.66 GHz with 2 GB of memory.

Logistics

In the logistics domain, we defined a control program using prioritized interrupts:

proc *mainControl*
$\langle$ *undeliveredPackages* $\rightarrow$ *deliverPackages* $\rangle \rangle\!\rangle$
$\langle \neg$ *finished* $\rightarrow$ `wait` $\rangle$

The program is to be understood as follows. In each cycle of the (implicit) main loop, if there are packages that have not been delivered yet, compute a plan to deliver them and execute it. If this is not the case but execution is not yet finished, do nothing for one cycle. Otherwise terminate.

Here, *finished* is a fluent that serves as a flag for signalling when program execution is supposed to halt. This is necessary to be able to perform finite experiments for a task that is indeed open-ended: While delivering the currently pending packages, new delivery requests keep arriving, each of which being modelled by an exogenous action `new_package(p,l,d)` that sets the current location of package `p` to `l` and its destination to `d`. Since the system does not know in advance when and how many new package arrivals will occur, a special exogenous action `no_more_packages` is used to set *finished* to `TRUE` after the last `new_package` event indicating that the experiment ends at this point.

Testing whether there are still packages to be delivered is done by procedure *undeliveredPackages*:

proc *undeliveredPackages*
$\exists p$: *package* $\exists l$: *location* $\exists d$: *location*.
$at(p, l) \wedge$ *destination*$(p, d) \wedge (l \neq d)$

The *deliverPackages* procedure is the part where planning comes into play:

proc *deliverPackages*
solve($\forall p$: *package* $\forall d$: *location*.
destination$(p, d) \supset at(p, d)$,
`[load,unload,drive]`)

Here, the first argument of **solve** is the goal formula and the second one the list of actions the planner has to consider. In our experiments, we tested two different versions of **solve**: one calling the external FF planner via `achieve_ff`, the other, `achieve`, being an internal, PROLOG-implemented construct of the IndiGolog framework that basically performs an iterative deepening search. The two planners were in each case given the same amount of information: a list of available actions, the fluent predicates involved (including their initial values) and objects' as well as fluent and action parameters' types. Whereas the internal `achieve` construct directly uses the appropriate part of the Golog domain axiomatization, FF is provided with the corresponding PDDL translation as described earlier.

Pack.	Trucks	Loc.	iN	eN	iR	eR
3	2	3	9	9	10	10
3	2	4	10	7	7	10
3	2	5	10	9	7	10
3	2	6	10	10	3	9
3	2	7	9	10	3	8
5	2	3	9	10	7	10
5	2	4	10	8	3	10
5	2	5	7	10	1	10
5	2	6	7	9	0	10
5	2	7	4	10	0	10
3	3	3	10	10	9	10
3	3	4	10	10	7	10
3	3	5	10	10	6	10
3	3	6	9	10	5	10
3	3	7	7	10	4	10
5	3	3	10	10	6	10
5	3	4	9	10	5	9
5	3	5	4	10	1	10
5	3	6	7	10	1	10
5	3	7	6	9	0	10

Table 1: Logistics: Number of instances solved

For both of the planners, we considered two variants. In the first one, once a plan is found it gets executed entirely. Packages arriving during that time are ignored until plan execution finishes and the next call to the planner is made. In the other variant, the system aborts the current plan and performs a re-planning after each `new_package` event.

We performed a series of experiments where the number of locations varied between 3 to 7, the number of trucks between 2 and 3 and the number of dynamically arriving packages among 3 and 5. For each combination, we created 10 different domain instances. The initial locations of trucks and packages as well as the destinations of the packages are chosen randomly. Two locations are connected with a probability of 50%; additional random edges ensure that the roadmap graph forms a single connected component. In each instance, there is one initial package, and the intervals between the arrival times of new packages vary between 2 and 8 steps, where one step corresponds to the execution of a primitive, non-exogenous action.

For each planner variant and domain instance we measured the overall runtime of the system and the number of steps (minus the number of `wait` actions) that were taken until termination. Runs that did not terminate within 300 seconds were aborted. The runtime includes a wait interval of 0.5 seconds after each executed action which was reserved to handle the communication with and the state update of the simulator.

Pack.	Trucks	Loc.	iN	eN	iR	eR
3	2	3	14.0	14.0	16.5	15.0
3	2	4	15.0	14.5	35.0	14.0
3	2	5	16.0	13.0	23.5	15.0
3	2	6	30.5	14.5	300.0	16.5
3	2	7	25.0	14.5	300.0	17.0
5	2	3	20.0	18.0	82.5	19.0
5	2	4	35.5	20.5	300.0	21.0
5	2	5	47.5	19.0	300.0	21.0
5	2	6	75.5	19.0	300.0	21.0
5	2	7	300.0	19.5	300.0	22.5
3	3	3	17.0	14.0	28.5	15.0
3	3	4	21.0	14.0	35.5	15.0
3	3	5	27.5	14.5	174.5	15.0
3	3	6	42.5	14.0	225.0	15.0
3	3	7	127.0	14.0	300.0	15.5
5	3	3	27.5	19.0	95.5	19.5
5	3	4	58.5	21.0	298.0	23.0
5	3	5	300.0	19.5	300.0	21.5
5	3	6	216.0	20.5	300.0	22.0
5	3	7	235.5	20.5	300.0	21.0

Table 2: Logistics: Median runtimes in seconds

Table 1 summarizes how many of the 10 instances were solved within the time limit by each method. Here, "e" refers to the variant where FF was used for planning while "i" means that the internal `achieve` was used. Further "R" means the variant where instant re-planning was done after the arrival of a new request and "N" the one where the current plan was not immediately aborted. The median runtimes for each combination are given in Table 2 and shown graphically in Figure 1, using a logarithmic scale. In those cases where the run did not finish within the timeout the value is set to the maximal time of 300 seconds. Table 3 contains the median number of steps that were taken, considering only instances that were solved by all methods.

The results clearly show that using FF instead of the internal planner has a large impact on the necessary computation time of the system, letting it solve instances within seconds which otherwise would require several minutes. One might object that our comparison is not equitable because we contrast FF which is a satisficing planner (i.e. which may return suboptimal plans in terms of plan length) with the internal planner that generates optimal plans (following an iterative deepening search strategy). We argue that in many cases optimal planning is not expedient: in fact, in the presence of unpredictable exogenous events or sensing it is not possible to plan really optimal. Furthermore, the results in Table 3 show that the number of steps required by the internal system is only slightly lower. Actually, our other benchmark domains show that using the external planning system also can result in a lower number of steps (Table 9). Nevertheless, there might be applications where optimal planning is more suitable, e.g. because there are no exogenous events and sensing is not required, or because the actual execution of an action takes a lot of time. Since we use PDDL to communicate with the external planner, our approach can easily

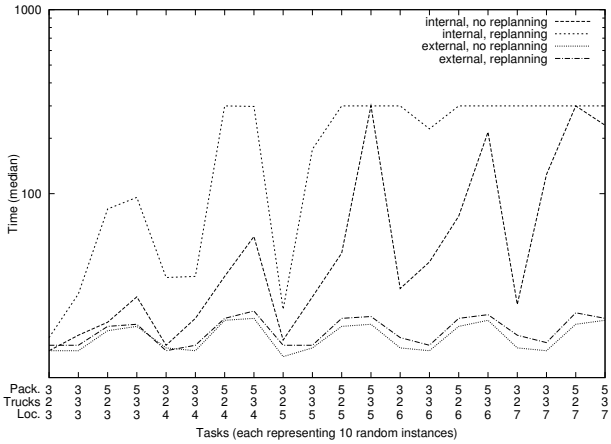


Figure 1: Logistics: Median runtimes in seconds

Pack.	Trucks	Loc.	Tasks	iN	eN	iR	eR
3	2	3	8	14.0	14.0	14.0	14.0
3	2	4	5	14.0	14.0	14.0	14.0
3	2	5	6	13.5	13.5	13.5	14.0
3	2	6	3	15.0	15.0	15.0	15.0
3	2	7	2	15.0	15.5	15.0	15.5
5	2	3	7	20.0	20.0	20.0	20.0
5	2	4	3	25.0	25.0	25.0	25.0
5	2	5	1	24.0	24.0	24.0	24.0
3	3	3	9	14.0	14.0	14.0	14.0
3	3	4	7	14.0	15.0	14.0	15.0
3	3	5	6	15.0	15.0	15.0	15.0
3	3	6	5	15.0	16.0	15.0	16.0
3	3	7	4	16.0	16.5	16.0	16.5
5	3	3	6	20.0	20.0	20.0	20.0
5	3	4	5	22.0	21.0	22.0	21.0
5	3	5	1	21.0	21.0	21.0	21.0
5	3	6	1	23.0	23.0	22.0	23.0

Table 3: Logistics: Median number of steps taken

be adapted to use any other planning system that handles PDDL, including optimal ones. As these planning systems are highly optimized, we would expect that they still would produce better results than the internal routine.

Elevator

The experimental setting for the elevator domain is analogous to the one of the logistics domain and uses the following main program.

proc *mainControl*
$\langle$ *unservedPassengers* $\rightarrow$ *servePassengers* $\rangle\rangle$
$\langle \neg$ *finished* $\rightarrow$ `wait` $\rangle$

Planning is used to serve the passengers that have not reached their destination yet:

proc *servePassengers*
solve($\forall p : passenger.\ served(p)$,
`[move_fast,move,stop]`)

Pass..	Floors	Tasks	iN	eN	iR	eR
3	5	10	10	10	10	10
3	6	10	8	10	8	10
3	7	10	10	9	4	10
3	8	10	6	10	3	10
5	5	10	10	9	7	9
5	6	10	8	10	1	9
5	7	10	4	9	0	9
5	8	10	5	7	0	10
7	5	10	7	10	1	10
7	6	10	1	10	0	10
7	7	10	0	9	0	8
7	8	10	0	10	0	10
9	5	10	4	9	0	10
9	6	10	0	9	1	10
9	7	10	0	7	0	10
9	8	10	0	8	0	9

Table 4: Elevator: Number of instances solved

Pass..	Floors	iN	eN	iR	eR
3	5	17.0	14.0	23.5	14.5
3	6	22.0	12.0	35.5	13.5
3	7	39.5	15.0	300.0	15.0
3	8	87.0	17.0	300.0	18.0
5	5	23.0	19.0	81.5	24.0
5	6	90.0	20.5	300.0	27.0
5	7	300.0	24.0	300.0	28.0
5	8	276.5	20.5	300.0	27.5
7	5	79.5	28.0	300.0	26.5
7	6	300.0	29.5	300.0	28.0
7	7	300.0	33.0	300.0	31.5
7	8	300.0	31.5	300.0	31.5
9	5	300.0	33.0	300.0	31.5
9	6	300.0	33.5	300.0	37.0
9	7	300.0	33.5	300.0	43.0
9	8	300.0	36.5	300.0	44.5

Table 5: Elevator: Median runtimes in seconds

Again, we tested two versions of **solve**, one calling the internal planner, the other calling the FF system, each with and without re-planning on the arrival of a new passenger.

For the benchmark instances of this domain we let the number of new passengers vary among 3, 5, 7 and 9 and the number of floors between 5 and 8. As above, we created 10 different instances for each combination, choosing the passengers' origins and destinations randomly. Initially there is always one passenger request and the intervals between newly arriving passengers lie between 2 and 8 steps.

In analogy to the previous domain, table 4 summarizes how many of the 10 instances were solved within the time limit by each method. The median runtimes for each combination are stated in Table 5 and shown graphically in Figure 2. Table 6 contains the median number of taken steps, considering only instances that were solved by all methods.

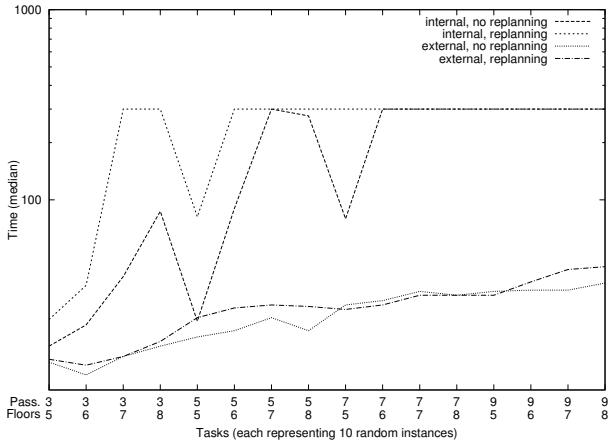


Figure 2: Elevator: Median runtimes in seconds

Pass..	Floors	Tasks	iN	eN	iR	eR
3	5	10	16.5	16.5	15.5	16.5
3	6	7	19.0	19.0	16.0	18.0
3	7	3	19.0	19.0	16.0	19.0
3	8	3	19.0	19.0	19.0	19.0
5	5	5	25.0	25.0	29.0	30.0
5	6	1	37.0	38.0	32.0	33.0
7	5	1	34.0	34.0	32.0	32.0

Table 6: Elevator: Median number of steps taken

Again, the variants with the external planner performed much better than the ones using the internal search method. The fifth row in Table 6 shows also an interesting detail: The re-planning strategy sometimes causes the overall number of steps to increase. This happens when a current plan is discarded to serve a new request, and when later another new request is made it turns out that following the original plan would have been less costly.

Mail Delivery

The mail delivery robot is controlled as follows:

proc *mainControl*
  while($\neg$*finished*) **do**
    (π_r m : *mailbox*) *getLettersFrom*(m);
    deliverLetters

Here, π_r denotes a variant of the non-deterministic choice of argument where the argument's instantiation is picked randomly. In case of the normal π construct, IndiGolog otherwise instantiates the variable always with the first applicable symbol, which would cause the program above to pick the same mailbox in each cycle of the loop. Once the next mailbox that should be visited is chosen, the path to it is determined by means of planning:

proc *getLettersFrom*(m)
  (π l : *location*)
    at(m, l)?;
    solve(*robotAt*(l), `[move]`);
    takeAllLetters(m)

Taking letters out of a mailbox requires sensing:

proc *takeAllLetters*(m)
  `look_into`(m);
  while$(\exists\, l : \textit{letter}.\ \textit{in}(l, m))$
    $\pi\ l : \textit{letter}$
      $\textit{in}(l, m)?$;
      `take_out`(l, m);
      `look_at`(l)
    `look_into`(m)

`look_into`(m) is a sensing action whose outcome is a constant l denoting one of the letters in the box (i.e. the robot can always only "see" the topmost one). Thus, the agent gets to know that fluent $\textit{in}(l, m)$ is currently true. In case the mailbox is empty, the return value is instead simply the special constant "`empty`". After picking up l, action `look_at`(l) is applicable and causes $\textit{addressee}(l, m')$ to become known to the agent for some mailbox m', which is the destination of letter l. For delivering the letters obtained like this, another call to the planner is made:

proc *deliverLetters*
  solve$(\forall\, l : \textit{letter}.$
      $(\exists\, m : \textit{mailbox}.\ \textit{addressee}(l, m))$
        $\supset \textit{delivered}(l),$
    `[put_in,move]`)

Again, we study the system's behavior for the case in which **solve** uses the internal planner and for the case where FF is called instead. Since there are no exogenous actions in this scenario, we do not consider dynamic re-planning.

In our benchmark scenarios the number of offices varies among 4, 8 and 16, the number of hallways among 2, 4 and 8, and the number of letters among 2, 4, 8 and 16. As in the other domains we created 10 instances for each combination, the offices being connected randomly to some hallway and hallways being connected to one another in a tree-like fashion. There are as many mailboxes as offices, but they are placed randomly. Therefore, it is possible that an office contains multiple mailboxes, only one, or even none at all. The origins and addressees of the letters are also chosen randomly.

Table 7 once again summarizes how many of the 10 instances were solved within the time limit by each method. The median runtimes for each combination are given in Table 8 and shown graphically in Figure 3. Table 9 contains the median number of steps that were taken, considering only instances that were solved by all methods.

The results for this domain are again quite conclusive. The controller using FF was able to solve more tasks and throughout required less computation time. In terms of steps, the two methods are comparable, but the results are somewhat erratic, which is mostly because of the randomized strategy that was used.

Conclusion

We empirically evaluated a system that integrates the FF planning system into the IndiGolog agent framework. For that purpose, we developed three example application domains in which classical planning subproblems arise in the course of the execution of a high-level program. A series of experiments with different scenarios and problem sizes shows that the integration of the external planner decreases the required computation time a lot, keeping the number of executed actions similar. We ran further experiments in order to examine the effect of re-planning after exogenous events. This strategy does not pay off if planning is done by the internal mechanism because the additional computation time dominates the savings. Using the external planning system, the results are more balanced but still not clearly indicating that re-planning pays off. This may be due to the relatively simple strategy that we used. A possible direction for future work therefore is to study what effect it has on our system to use a more sophisticated method for execution monitoring and re-planning.

Lett.	Off.	Hall.	Tasks	i	e
10	4	2	10	6	10
10	4	4	10	6	10
10	4	8	10	4	10
10	8	2	10	4	10
10	8	4	10	5	10
10	8	8	10	5	10
10	16	2	10	5	10
10	16	4	10	6	10
10	16	8	10	7	10
15	4	2	10	1	10
15	4	4	10	1	10
15	4	8	10	0	10
15	8	2	10	1	10
15	8	4	10	0	10
15	8	8	10	2	10
15	16	2	10	2	10
15	16	4	10	2	10
15	16	8	10	0	10

Table 7: Mail Delivery: Number of instances solved

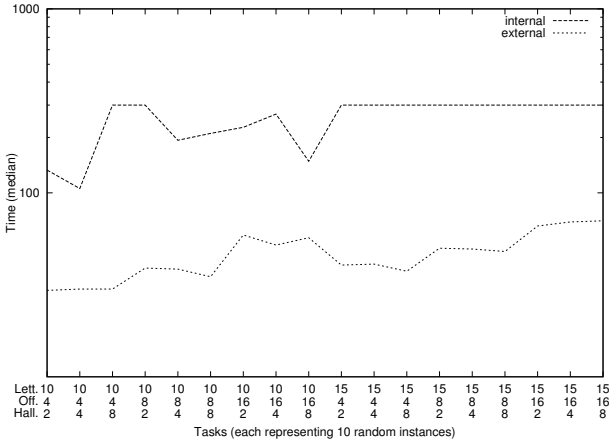


Figure 3: Mail Delivery: Median runtimes in seconds

Lett.	Off.	Hall.	i	e
10	4	2	133.0	29.5
10	4	4	105.5	30.0
10	4	8	300.0	30.0
10	8	2	300.0	39.0
10	8	4	193.0	38.5
10	8	8	210.5	35.0
10	16	2	227.0	59.0
10	16	4	268.0	52.0
10	16	8	148.5	57.0
15	4	2	300.0	40.5
15	4	4	300.0	41.0
15	4	8	300.0	37.5
15	8	2	300.0	50.0
15	8	4	300.0	49.5
15	8	8	300.0	48.0
15	16	2	300.0	66.0
15	16	4	300.0	69.5
15	16	8	300.0	70.5

Table 8: Mail Delivery: Median runtimes in seconds

Lett.	Off.	Hall.	Tasks	i	e
10	4	2	6	50.5	51.0
10	4	4	6	49.5	51.0
10	4	8	4	54.0	50.0
10	8	2	4	68.5	63.0
10	8	4	5	61.0	67.0
10	8	8	5	71.0	59.0
10	16	2	5	92.0	93.0
10	16	4	6	92.0	93.5
10	16	8	7	91.0	95.0
15	4	2	1	69.0	72.0
15	4	4	1	69.0	70.0
15	8	2	1	101.0	92.0
15	8	8	2	76.0	95.0
15	16	2	2	134.5	114.5
15	16	4	2	112.5	105.5

Table 9: Mail Delivery: Median number of steps taken

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft under grants La 747/14-1 and Ne 623/10-1. We thank Sebastian Sardina and Stavros Vassos for their help with the IndiGolog framework.

References

Burgard, W.; Cremers, A. B.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; and Thrun, S. 1998. The interactive museum tour-guide robot. In *Proc. AAAI-98*, 11–18.

Claßen, J.; Eyerich, P.; Lakemeyer, G.; and Nebel, B. 2007. Towards an integration of Golog and planning. In *Proc. IJCAI 2007*, 1846–1851.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.* 121(1–2):109–169.

De Giacomo, G.; Levesque, H. J.; and Sardina, S. 2001. Incremental execution of guarded theories. *Computational Logic* 2(4):495–525.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. report 195, Inst. f. Informatik, Univ. Freiburg.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, University of Brescia.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language.

Helmert, M. 2008. *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, volume 4929 of *LNAI*. Springer-Verlag.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *J. Log. Prog.* 31:59–84.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. New York: American Elsevier. 463–502.

Pednault, E. P. D. 1989. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR-89*, 324–332.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Röger, G., and Nebel, B. 2007. Expressiveness of ADL and Golog: Functions make a difference. In *Proc. AAAI 2007*.

Röger, G.; Helmert, M.; and Nebel, B. 2008. On the relative expressiveness of ADL and Golog: The last piece in the puzzle. In *Proc. KR 2008*. to appear.

Sardina, S.; De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in Indigolog—from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 41(2-4):259–299.

Tam, K.; Lloyd, J.; Lespérance, Y.; Levesque, H. J.; Lin, F.; Marcu, D.; Reiter, R.; and Jenkin, M. R. M. 1997. Controlling autonomous robots with GOLOG. In *Proc. IJCAI-97*, 1–12.

A diff-Based Merging Operator

Patricia Everaere
LIFL - CNRS
Université de Lille 1 - Lille - France
patricia.everaere@univ-lille1.fr

Sébastien Konieczny **Pierre Marquis**
CRIL - CNRS
Université d'Artois - Lens - France
{konieczny, marquis}@cril.fr

Abstract

Merging operators aim at defining the beliefs (resp. the goal) of a group of agents from a profile of bases, gathering the beliefs (resp. the goals) of each member of the group. In the propositional setting, a well-studied family of merging operators are distance-based ones: the models of the merged base are the closest interpretations to the given profile. Closeness is, in this context, measured as a number resulting from the aggregation of the distances to each base of the profile. In this work we define a new kind of propositional merging operators, close to such distance-based merging operators, but relying on a set-theoretic definition of closeness, already at work in several revision/update operators from the literature. We study a specific merging operator of this family, obtained by considering set-product as the aggregation function.

Introduction

Information merging is a very important task in artificial intelligence: the issue is to determine the beliefs, or the goals, of a group of agents from their individual points of view. Much work has been devoted to the definition of merging operators in the propositional case (Revesz 1997; Liberatore & Schaerf 1998; Baral *et al.* 1992; Konieczny & Pino Pérez 2002a; Meyer, Pozos Parra, & Perrussel 2005), and to the study of their properties with respect to different criteria, mainly logical properties, strategy-proofness, complexity. See for instance (Konieczny & Pino Pérez 2002a; Revesz 1997; Liberatore & Schaerf 1998; Konieczny, Lang, & Marquis 2004) for logic-based characterizations, (Everaere, Konieczny, & Marquis 2007) for an investigation of strategy-proofness issues, and (Konieczny, Lang, & Marquis 2004; Everaere, Konieczny, & Marquis 2007) for computational complexity results. There exist also works on merging in richer logical settings than propositional logic, see for instance (Meyer 2001; Benferhat *et al.* 2002; Chopra, Ghose, & Meyer 2006; Benferhat, Lagrue, & Rossit 2007).

In (Konieczny & Pino Pérez 2002a) a set of postulates is proposed to characterize different families of merging operators, and several families of operators satisfying these postulates are defined. Such operators are called model-based merging operators because basically they select the models of a given integrity constraint (i.e., a formula encoding laws, norms, etc., used for constraining the result of the merging) that are the closest ones to the given profile of belief/goal bases of the group. Often, these operators are defined from a distance between interpretations.This distance between interpretations induce a distance between an interpretation and a base, which indicates how plausible/satisfactory the interpretation is with respect to the base. Once such distances are computed, an aggregation function is used to define the overall distance of each model (of the integrity constraints) to the profile. The models of the result of the merging are the closest models of the integrity constraints to the profile.

A commonly-used distance between interpretations is the Hamming distance (also called Dalal distance (Dalal 1988)). The Hamming distance between two interpretations is the number of propositional variables the two interpretations disagree on. The closeness between two interpretations is thus assessed as the number of atoms whose truth values must be flipped in one interpretation in order to make it identical to the second one. Such a distance is meaningful when no extra-information on the epistemic states of the agents are available.

The major problem with distance-based merging operators is that evaluating the closeness between two interpretations as a number may lead to lose too much information. Thus, the conflicting variables themselves (and not only how many they are) can prove significant. Especially, when variables express real-world properties, it can be the case that some variables are more important than others, or that some variables are logically connected. In these cases, distances are not mandatory.

As an alternative to distance, an interesting measure used to evaluate the closeness of two interpretations is diff, the symetrical difference between them. Instead of evaluating the degree of conflict between two interpretations as the number of variables on which they differ (as it is the case with the Hamming distance), the diff measure assesses it as the set of such variables.

In this paper, we consider the family of propositional merging operators based on the diff measure. We specifically focus on the operator $\Delta^{\mathsf{diff},\oplus}$ from this family obtained by considering set-product as the aggregation function. We evaluate it with respect to three criteria: logical properties, strategy-proofness and complexity. Other operators from

this family are presented in (Everaere, Konieczny, & Marquis 2008).

The rest of the paper is as follows. In the following section, we give some formal preliminaries. Then, we define the family of model-based merging operators based on the diff measure of closeness, and make precise the specific operator $\Delta^{\mathsf{diff},\oplus}$ we focus on. In the next section, we report on the logical properties of $\Delta^{\mathsf{diff},\oplus}$ and we discuss the strategy-proofness issues for it. The computational complexity of $\Delta^{\mathsf{diff},\oplus}$ is given after, just before a discussion about some related work. The paper ends with some perspectives.

Preliminaries

We consider a propositional language $\mathcal{L}$ defined from a finite set of propositional variables $\mathcal{P}$ and the usual connectives.

An interpretation (or world) is a total function from $\mathcal{P}$ to $\{0,1\}$, denoted by a bit vector whenever a strict total order on $\mathcal{P}$ is specified. The set of all interpretations is noted $\mathcal{W}$. An interpretation ω is a model of a formula $\phi \in \mathcal{L}$ if and only if it makes it true in the usual truth functional way. $[\phi]$ denotes the set of models of formula ϕ, i.e., $[\phi] = \{\omega \in \mathcal{W} \mid \omega \models \phi\}$.

A *base* K denotes the set of beliefs or goals of an agent, it is a finite and consistent set of propositional formulas, interpreted conjunctively. Unless stated otherwise, we identify K with the conjunction of its elements.

A *profile* E denotes the group of agents involved in the merging process. It is a multi-set (bag) of belief/goal bases $E = \{K_1, \ldots, K_n\}$ (hence two agents are allowed to exhibit identical bases). We note $\sqcup$ the union of multi-sets. We denote by $\bigwedge E$ the conjunction of bases of E, i.e., $\bigwedge E = K_1 \wedge \ldots \wedge K_n$. A profile E is said to be consistent if and only if $\bigwedge E$ is consistent. We say that two profiles are equivalent, noted $E_1 \equiv E_2$, if there exists a bijection f from E_1 to E_2 such that for every $\phi \in E_1$, ϕ and $f(\phi)$ are logically equivalent.

The result of the merging of the bases of a profile E, under the integrity constraints μ, is the *merged base* denoted $\Delta_\mu(E)$. The *integrity constraints* consist of a formula the merged base has to satisfy.

Diff-Based Merging Operators

As a gentle introduction to diff-based merging operators, let us first recall how distance-based merging operators are defined. This calls for a notion of (pseudo-)distance between interpretations and a notion of aggregation function.

Definition 1 *A* (pseudo-)distance between interpretations *is a total function d from $\mathcal{W} \times \mathcal{W}$ to IN such that for every ω_1, $\omega_2 \in \mathcal{W}$:*

- $d(\omega_1, \omega_2) = d(\omega_2, \omega_1)$, *and*
- $d(\omega_1, \omega_2) = 0$ *if and only if* $\omega_1 = \omega_2$.

Any distance between interpretations d induces a "distance" between an interpretation ω and a base K defined by $d(\omega, K) = \min_{\omega' \models K} d(\omega, \omega')$.

Definition 2 *An* aggregation function *is a total function f associating a non-negative integer to every finite tuple of non-negative integers and verifying (non-decreasingness), (minimality) and (identity).*

- *if* $x \leq y$, *then* $f(x_1, \ldots, x, \ldots, x_n) \leq f(x_1, \ldots, y, \ldots, x_n)$. *(non-decreasingness)*
- $f(x_1, \ldots, x_n) = 0$ *if and only if* $x_1 = \ldots = x_n = 0$. *(minimality)*
- *for every non-negative integer x,* $f(x) = x$. *(identity)*

We can now define distance-based merging operators:

Definition 3 *Let d be a distance and f be an aggregation function. The* distance-based merging operator *induced by d and f is defined by: for any profile $E = \{K_1, \ldots, K_n\}$ and any integrity constraint μ,*

$$[\Delta_\mu^{d,f}(E)] = \{\omega \models \mu \mid f(d(\omega, K_1), \ldots, d(\omega, K_n)) \text{ is minimal}\}.$$

Such operators have been extensively studied, and many "standard" merging operators belong to this class (Revesz 1997; Liberatore & Schaerf 1998). Their logical properties are stated in (Konieczny & Pino Pérez 2002a), their strategy-proofness is studied in (Everaere, Konieczny, & Marquis 2007), and their computational complexity in (Konieczny, Lang, & Marquis 2004).

Let us now turn to diff-based merging operators. Basically, the idea consists in evaluating closeness between two interpretations ω and ω' as the set of variables on which they differ:

$$\mathsf{diff}(\omega, \omega') = \{p \in \mathcal{P} \mid \omega(p) \neq \omega'(p)\}.$$

This definition has already been used in the belief revision/update literature in order to define a number of operators (Katsuno & Mendelzon 1991; Weber 1986; Satoh 1988; Borgida 1985; Winslett 1988).

As for distances, we can straightforwardly define, using diff, a notion of closeness between an interpretation and a base, as the minimum closeness between the interpretation and the models of the base. Of course, since diff gives as output a set instead of a number, set-inclusion has to be considered as minimality criterion:

$$\mathsf{diff}(\omega, K) = min(\{\mathsf{diff}(\omega, \omega') \mid \omega' \models K\}, \subseteq).$$

So the closeness between an interpretation ω and a base K is measured as the set of all minimal sets (for set inclusion) of propositional variables which have to be flipped in ω to make it a model of K.

Now, we need to aggregate these measures in order to define a global notion of closeness between an interpretation and a profile. This is the aim of the aggregation functions. Of course, usual functions at work for distance-based operators cannot be used here simply because we do not work with numbers, but with sets.

Several aggregation functions can be considered in our setting. We focus on a single one in this paper. We consider set-product $\oplus$ as aggregation function: for two sets of sets E and E', $E \oplus E' = \{c \cup c' \mid c \in E \text{ and } c' \in E'\}$.

ω	$\mathsf{diff}(\omega, K_1)$	$\mathsf{diff}(\omega, K_2)$	$\mathsf{diff}(\omega, \{K_1, K_2\})$
0000	$\{\{p,q,r\}\}$	$\{\{p,s\}\}$	$\{\{p,q,r,s\}\}$
0001	$\{\{p,q,r\}\}$	$\{\{p\}\}$	$\{\{p,q,r\}\}$
0010	$\{\{p,q\}\}$	$\{\{p,s\}\}$	$\{\{p,q,s\}\}$
0011	$\{\{p,q\}\}$	$\{\{p\}\}$	$\{\{p,q\}\}$
0100	$\{\{p,r\}\}$	$\{\{p,q,s\}\}$	$\{\{p,q,r,s\}\}$
0101	$\{\{p,r\}\}$	$\{\{p,q\}\}$	$\{\{p,q,r\}\}$
0110	$\{\{p\}\}$	$\{\{p,q,s\}\}$	$\{\{p,q,s\}\}$
0111	$\{\{p\}\}$	$\{\{p,q\}\}$	$\{\{p,q\}\}$
1000	$\{\{q,r\}\}$	$\{\{s\}\}$	$\{\{q,r,s\}\}$
1001	$\{\{q,r\}\}$	$\emptyset$	$\{\{q,r\}\}$
1010	$\{\{q\}\}$	$\{\{s\}\}$	$\{\{q,s\}\}$
1011	$\{\{q\}\}$	$\emptyset$	$\{\{q\}\}$
1100	$\{\{r\}\}$	$\{\{q,s\}\}$	$\{\{q,r,s\}\}$
1101	$\{\{r\}\}$	$\{\{q\}\}$	$\{\{q,r\}\}$
1110	$\emptyset$	$\{\{q,s\}\}$	$\{\{q,s\}\}$
1111	$\emptyset$	$\{\{q\}\}$	$\{\{q\}\}$

Table 1: Computation of $\Delta_{\top}^{\mathsf{diff},\oplus}(E)$

Definition 4 *Let $E = \{K_1, \ldots, K_n\}$ be a profile and ω an interpretation. The closeness between ω and E is given by:*

$$\mathsf{diff}(\omega, E) = \min(\{\oplus_{K_i \in E}\ \mathsf{diff}(\omega, K_i)\}, \subseteq).$$

By construction, each element of $\mathsf{diff}(\omega, E)$ is a minimal set c of variables (a conflict set) such that for each base K_i, ω can be transformed into a model of K_i by flipping in ω the variables of c.

Finally, we define a merging operator $\Delta^{\mathsf{diff},\oplus}$ which picks up the models of the integrity constraints whose closeness to the profile E contains at least one of the minimal (w.r.t. $\subseteq$) conflict set:

Definition 5 *Let $E = \{K_1, K_2, \ldots, K_n\}$ be a profile, μ an integrity constraint. Then:*

$$\mathsf{diff}_\mu(E) = \min(\{\mathsf{diff}(\omega, E) \mid \omega \models \mu\}, \subseteq)$$

and

$$[\Delta_\mu^{\mathsf{diff},\oplus}(E)] = \{\omega \models \mu \mid \exists c \in \mathsf{diff}(\omega, E) \textit{ s.t. } c \in \mathsf{diff}_\mu(E)\}.$$

Example 1 *We consider a profile $E = \{K_1, K_2\}$ with $K_1 = \{p \wedge q \wedge r\}$ and $K_2 = \{p \wedge \neg q \wedge s\}$, there is no integrity constraint (i.e., $\mu \equiv \top$).* $\mathsf{diff}_\mu(E) = \min(\{\{p,q,r,s\}, \{p,q,r\}, \{p,q,s\}, \{p,q\}, \{q,r,s\}, \{q,r\}, \{q,\ s\}, \{q\}\}, \subseteq) = \{\{q\}\}$. $[\Delta_\top^{\mathsf{diff},\oplus}(E)] = \{1111, 1011\}$ *so* $\Delta_\top^{\mathsf{diff},\oplus}(E) \equiv p \wedge r \wedge s$ *(see Table 1).*

Just as many IC merging operators can be considered as generalizations of AGM revision operators (Konieczny & Pino Pérez 2002a), one can easily show that $\Delta^{\mathsf{diff},\oplus}$ can be viewed as a generalization of the well-known Satoh's revision operator (Satoh 1988), denoted $\circ_{\mathcal{S}}$:

Proposition 1 *Let K be a base and μ an integrity constraint. We have:*

$$\Delta_\mu^{\mathsf{diff},\oplus}(\{K\}) \equiv K \circ_{\mathcal{S}} \mu.$$

Logical Properties

Since we aim at investigating the logical properties of the merging operator $\Delta^{\mathsf{diff},\oplus}$, a set of properties must first be considered as a base line. The following set of postulates was proposed in (Konieczny & Pino Pérez 2002a):

Definition 6 *$\triangle$ is an* IC merging operator *if and only if it satisfies the following postulates:*

(IC0) $\triangle_\mu(E) \models \mu$

(IC1) *If μ is consistent, then $\triangle_\mu(E)$ is consistent*

(IC2) *If $\bigwedge E$ is consistent with μ, then $\triangle_\mu(E) \equiv \bigwedge E \wedge \mu$*

(IC3) *If $E_1 \equiv E_2$ and $\mu_1 \equiv \mu_2$, then $\triangle_{\mu_1}(E_1) \equiv \triangle_{\mu_2}(E_2)$*

(IC4) *If $K_1 \models \mu$ and $K_2 \models \mu$, then $\triangle_\mu(\{K_1, K_2\}) \wedge K_1$ is consistent if and only if $\triangle_\mu(\{K_1, K_2\}) \wedge K_2$ is consistent*

(IC5) $\triangle_\mu(E_1) \wedge \triangle_\mu(E_2) \models \triangle_\mu(E_1 \sqcup E_2)$

(IC6) *If $\triangle_\mu(E_1) \wedge \triangle_\mu(E_2)$ is consistent, then $\triangle_\mu(E_1 \sqcup E_2) \models \triangle_\mu(E_1) \wedge \triangle_\mu(E_2)$*

(IC7) $\triangle_{\mu_1}(E) \wedge \mu_2 \models \triangle_{\mu_1 \wedge \mu_2}(E)$

(IC8) *If $\triangle_{\mu_1}(E) \wedge \mu_2$ is consistent, then $\triangle_{\mu_1 \wedge \mu_2}(E) \models \triangle_{\mu_1}(E)$*

For explanations on these postulates see (Konieczny & Pino Pérez 2002a).

Proposition 2 *$\Delta^{\mathsf{diff},\oplus}$ satisfies* **(IC0)**, **(IC1)**, **(IC2)**, **(IC3)**, **(IC4)** *and* **(IC7)**. *It does not satisfy* **(IC5)**, **(IC6)** *and* **(IC8)**.

The reason why $\Delta^{\mathsf{diff},\oplus}$ does not satisfies **(IC8)** is that this property requires a total criterion (i.e., the corresponding syncretic assignment (Konieczny & Pino Pérez 2002b) must associates a **total** pre-order to each profile, so that any two interpretations can always be compared), whereas diff gives rise only to partial relations.

$\Delta^{\mathsf{diff},\oplus}$ also does not satisfy **(IC5)** and **(IC6)**, which are postulates capturing aggregation properties. This is not surprising since, unlike distance-based operators (as the ones based on Hamming distance), $\Delta^{\mathsf{diff},\oplus}$ keeps a justification of the minimality of an interpretation (as a conflict set). So, when joining two groups, it may happen that the justifications needed to motivate this choice become too weak. As an example, assume that two profiles E_1 and E_2 select independently as the result of the merging a formula with only two models ω and ω' (i.e., $[\Delta_\mu^{\mathsf{diff},\oplus}(E_1)] = [\Delta_\mu^{\mathsf{diff},\oplus}(E_2)] = \{\omega, \omega'\}$), and suppose that the conflict sets between the models and the profiles are the following ones: $\mathsf{diff}(\omega, E_1) = \mathsf{diff}(\omega', E_1) = \mathsf{diff}(\omega, E_2) = \{\{a, b\}\}$, and $\mathsf{diff}(\omega', E_2) = \{\{a, c\}\}$. Then, if we join the two groups we obtain $\mathsf{diff}(\omega, E_1 \sqcup E_2) = \{\{a, b\}\}$ and $\mathsf{diff}(\omega', E_1 \sqcup E_2) = \{\{a, b, c\}\}$. The conflict set associated to ω' is not minimal anymore. Since $\mathsf{diff}(\omega, E_1 \sqcup E_2) \subset \mathsf{diff}(\omega', E_1 \sqcup E_2)$, we have $\omega' \not\models \Delta_\mu^{\mathsf{diff},\oplus}(E_1 \sqcup E_2)$, whereas $\omega' \models \Delta_\mu^{\mathsf{diff},\oplus}(E_1) \wedge \Delta_\mu^{\mathsf{diff},\oplus}(E_2)$, which contradicts **(IC6)**.

On the following example, we illustrate how $\Delta^{\mathsf{diff},\oplus}$ can prove better than usual distance-based merging operators:

Example 2 *Consider four bases $[K_1] = \{0010, 0100\}$, $[K_2] = \{0001, 0100\}$, $[K_3] = \{0111, 0100\}$, and $[K_4] = \{1011, 0100\}$. $E = \{K_1, K_2, K_3, K_4\}$. The only possible worlds are $[\mu] = \{0011, 0000\}$.*

ω		0011		0000	
$\mathsf{diff}(\omega, K_1)$	$d_H(\omega, K_1)$	$\{\{d\}\}$	1	$\{\{b\},\{c\}\}$	1
$\mathsf{diff}(\omega, K_2)$	$d_H(\omega, K_2)$	$\{\{c\}\}$	1	$\{\{b\},\{d\}\}$	1
$\mathsf{diff}(\omega, K_3)$	$d_H(\omega, K_3)$	$\{\{b\}\}$	1	$\{\{b\}\}$	1
$\mathsf{diff}(\omega, K_4)$	$d_H(\omega, K_4)$	$\{\{a\},\{b,c,d\}\}$	1	$\{\{b\},\{a,c,d\}\}$	1
$\mathsf{diff}(\omega, E)$	$d_{H,\Sigma}(\omega, E)$	$\{\{b,c,d\}\}$	4	$\{\{b\}\}$	4

Table 2: Computations of $\Delta_\mu^{\mathsf{diff},\oplus}(E)$ and $\Delta_\mu^{d_H,\Sigma}(E)$

Computations of the merged bases for operators $\Delta^{\mathsf{diff},\oplus}$ and $\Delta_\mu^{d_H,\Sigma}$ are summed up in Table 2 ($\Delta_\mu^{d_H,\Sigma}$ is the distance-based merging operator relying on the Hamming distance and using sum as an aggregation function (Revesz 1997; Konieczny & Pino Pérez 2002a)).
We get $[\Delta_\mu^{\mathsf{diff},\oplus}(E)] = \{0000\}$, while $[\Delta_\mu^{d_H,\Sigma}(E)] = \{0011, 0000\}$.
Clearly the Hamming distance d_H does not discriminate between the two possible worlds, which can be problematic. Here all the agents agree on what they disagree with 0000 (i.e., the conflict is on b), while this is not the case for 0011. Operators based on the Hamming distance cannot make this distinction. As one can check in Table 2 the Hamming distances of the interpretations to the bases are all identical and equal to 1, whereas the diff distance exhibits the fact that there is less conflict on 0000 than on 0011 (while flipping the variable b in 0000 is enough to obtain a model of all the bases, it is not the case with 0011).

Beyond the IC postulates, $\Delta^{\mathsf{diff},\oplus}$ satisfies also an interesting additional logical property:

Definition 7 *A merging operator Δ satisfies the* temperance *property iff for every profile $\{K_1, \ldots, K_n\}$:*

$\Delta_\top(\{K_1, \ldots, K_n\})$ *is consistent with each* K_i **(temperance)**

Proposition 3 $\Delta^{\mathsf{diff},\oplus}$ *satisfies* **(temperance)**.

This proposition shows that the merged base obtained using $\Delta^{\mathsf{diff},\oplus}$ is consistent with every base of the profile (when there is no integrity constraint). This proposition also gives an additional explanation to the fact that $\Delta^{\mathsf{diff},\oplus}$ does not satisfy **(IC6)**, since temperance is not compatible with this postulate.

Proposition 4 *There is no merging operator satisfying all of* **(IC2)**, **(IC6)**, *and* **(temperance)**.

It is worth noting that the temperance property is not satisfied by many merging operators. In particular, as implied by the previous proposition, none of the IC merging operators satisfies temperance. Interestingly, the temperance property shows that $\Delta^{\mathsf{diff},\oplus}$ can be viewed as a kind of negotiation operator, which can be used for determining the most consensual parts of the bases of all agents. This can prove useful for defining new negotiation operators, as studied for instance in (Zhang *et al.* 2004; Meyer *et al.* 2004b; 2004a; Booth 2001; 2006; Konieczny 2004).

Strategy-Proofness

Let us now investigate how robust $\Delta^{\mathsf{diff},\oplus}$ is with respect to manipulation. Intuitively, a merging operator is strategy-proof if and only if, given the beliefs/goals of the other agents, reporting untruthful beliefs/goals does not enable an agent to improve her satisfaction. A formal counterpart of this idea is given in (Everaere, Konieczny, & Marquis 2004; 2007):

Definition 8 *Let i be a satisfaction index, i.e., a total function from $\mathcal{L} \times \mathcal{L}$ to $\mathbb{R}$. A merging operator Δ is* strategy-proof for i *if and only if there is no integrity constraint μ, no profile $E = \{K_1, \ldots, K_n\}$, no base K and no base K' such that*

$$i(K, \Delta_\mu(E \sqcup \{K'\})) > i(K, \Delta_\mu(E \sqcup \{K\})).$$

Clearly, there are numerous different ways to define the satisfaction of an agent given a merged base. While many *ad hoc* definitions can be considered, the following three indexes are meaningful when no additional information are available (Everaere, Konieczny, & Marquis 2004; 2007):

Definition 9

- $i_{d_w}(K, K_\Delta) = \begin{cases} 1 & \text{if } K \wedge K_\Delta \text{ is consistent,} \\ 0 & \text{otherwise.} \end{cases}$
- $i_{d_s}(K, K_\Delta) = \begin{cases} 1 & \text{if } K_\Delta \models K, \\ 0 & \text{otherwise.} \end{cases}$
- $i_p(K, K_\Delta) = \begin{cases} \frac{\#([K] \cap [K_\Delta])}{\#([K_\Delta])} & \text{if } \#([K_\Delta]) \neq 0, \\ 0 & \text{otherwise.} \end{cases}$

For the weak drastic index (i_{d_w}), the agent is considered satisfied as soon as its beliefs/goals are consistent with the merged base. For the strong drastic index (i_{d_s}), in order to be satisfied, the agent must impose her beliefs/goals to the whole group. The last index ("probabilistic index" i_p) is not a Boolean one, leading to a more gradual notion of satisfaction. The more compatible the merged base with the agent's base the more satisfied the agent. The compatibility degree of K with K_Δ is the (normalized) number of models of K that are models of K_Δ as well.

Proposition 5 *In the general case $\Delta^{\mathsf{diff},\oplus}$ is not strategy-proof for any of the three indexes i_{d_w}, i_{d_s} and i_p. When there is no integrity constraint (i.e., $\mu \equiv \top$), $\Delta^{\mathsf{diff},\oplus}$ is strategy-proof for i_{d_w}, but still not strategy-proof for i_{d_s} or i_p.*

Most of the model-based operators are not strategy-proof, even in very restricted situations (Everaere, Konieczny, & Marquis 2007). For example, $\Delta^{d_H,\Sigma}$ or $\Delta^{d_H,Gmin}$, which

are the best model-based operators with respect to strategy-proofness, are not strategy-proof for i_{d_w}, even if $\mu \equiv \top$. $\Delta^{\mathsf{diff},\oplus}$ performs slightly better than any of them with this respect.

Complexity Issues

Let us consider now the complexity issue for the inference problem from a $\Delta^{\mathsf{diff},\oplus}$-merged base. We assume the reader acquainted with basics of complexity theory (see (Papadimitriou 1994)).

Formally, let us consider the following decision problem $\text{MERGE}(\Delta^{\mathsf{diff},\oplus})$:

- **Input:** A triple $\langle E, \mu, \alpha\rangle$ where $E = \{K_1, \ldots, K_n\}$ is a profile, $\mu \in \mathcal{L}$ is a formula, and $\alpha \in \mathcal{L}$ is a formula.
- **Question:** Does $\Delta^{\mathsf{diff},\oplus}_{\mu}(E) \models \alpha$ hold?

Proposition 6 $\text{MERGE}(\Delta^{\mathsf{diff},\oplus})$ *is* Π^p_2*-complete.*

This result shows that $\Delta^{\mathsf{diff},\oplus}$ is computationally harder than usual distance-based operators, but is at the same complexity level as many formula-based operators (Konieczny, Lang, & Marquis 2004), and as complex as the corresponding revision operator (see Proposition 1) (Eiter & Gottlob 1992).

Related Work: Consistency-Based Operators

In (Delgrande & Schaub 2007) two consistency-based merging operators, based on a default inference relation, are proposed. The idea is to use a specific language for each of the bases (disjoint from all other), so as to make their union consistent, and then to add as much default equivalence as possible in order to identify the corresponding variables of the different languages.

At a first glance, these operators seem very close to $\Delta^{\mathsf{diff},\oplus}$, since they try to maximise the agreement between the bases at the variable level, whereas $\Delta^{\mathsf{diff},\oplus}$ tries to minimize the conflict. Furthermore, these two operators satisfy also the temperance property. However one can show that all three operators are actually distinct (and even incomparable as to their inferential power).

Let us first give a brief refresher on Delgrande and Schaub's operators. A i-renaming of a language $\mathcal{L}$ is the language $\mathcal{L}^i$, built from the set of propositional variables $\mathcal{P}^i = \{p^i \mid p \in \mathcal{P}\}$, where for each $\alpha \in \mathcal{L}$, α^i is the result of replacing in α each propositional variable $p \in \mathcal{P}$ by the corresponding propositional variable $p^i \in \mathcal{P}^i$. Given a base K, the i-renaming of (the formulas of) K, is denoted K^i.

Definition 10 *Let $E = \{K_1, K_2, \ldots, K_n\}$ be a profile.*

- *Let EQ be a maximal (w.r.t $\subseteq$) subset of $\{p^k \Leftrightarrow p^l \mid p \in \mathcal{L}$ and $k, l \in \{1 \ldots n\}\}$ such that $(\bigwedge_{K_i \in E} K_i^i) \wedge EQ$ is consistent.*
  Then $\{\alpha \mid \forall j \in \{1 \ldots n\}\ (\bigwedge_{K_i \in E} K_i^i) \wedge EQ \models \alpha^j\}$ is a consistent symmetric belief change extension *of E.*
  The skeptical merging $\Delta_s(E)$ *of E is the intersection of all the consistent symmetric belief change extensions of E.*

ω	$\mathsf{diff}(\omega, K_1)$	$\mathsf{diff}(\omega, K_2)$	$\mathsf{diff}(\omega, K_3)$	$\mathsf{diff}(\omega, E)$
000	$\{\{p,q\},\{q,r\}\}$	$\{\{p\},\{q\}\}$	$\{\emptyset\}$	$\{\{p,q\},\{q,r\}\}$
001	$\{\{q\}\}$	$\{\{p,r\},\{q,r\}\}$	$\{\{r\}\}$	$\{\{q,r\}\}$
010	$\{\{p\},\{r\}\}$	$\{\emptyset\}$	$\{\{q\}\}$	$\{\{p,q\},\{q,r\}\}$
011	$\{\emptyset\}$	$\{\{r\}\}$	$\{\{q,r\}\}$	$\{\{q,r\}\}$
100	$\{\{q\}\}$	$\{\emptyset\}$	$\{\emptyset\}$	$\{\{q\}\}$
101	$\{\{p,q\},\{q,r\}\}$	$\{\{r\}\}$	$\{\{r\}\}$	$\{\{q,r\}\}$
110	$\{\emptyset\}$	$\{\{q\},\{p\}\}$	$\{\{q\}\}$	$\{\{q\}\}$
111	$\{\{p\},\{r\}\}$	$\{\{q,r\},\{p,r\}\}$	$\{\{q,r\}\}$	$\{\{q,r\}\}$

Table 3: Example 2 - Computation of $\Delta^{\mathsf{diff},\oplus}_{\top}(E)$

- *Let EQ be a maximal (w.r.t $\subseteq$) subset of $\{p^j \Leftrightarrow p \mid p \in \mathcal{L}$ and $j \in \{1 \ldots n\}\}$ such that $(\bigwedge_{K_i \in E} K_i^i) \wedge EQ$ is consistent.*
  Then $(\bigwedge_{K_i \in E} K_i^i) \wedge EQ$ is a consistent projected belief change extension *of E.*
  The skeptical merging $\nabla_s(E)$ *of E is the intersection of all the consistent projected belief change extensions of E.*

Example 3 *We consider the profile $E = \{K_1, K_2, K_3\}$, with $K_1 = (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r)$, $K_2 = (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r)$ and $K_3 = \neg q \wedge \neg r$.*
The computation of $\Delta^{\mathsf{diff},\oplus}_{\top}(E)$ is described in Table 3. We have $\Delta^{\mathsf{diff},\oplus}_{\top}(E) \equiv p \wedge \neg r$.
There are four maximal sets of equivalences for $\Delta_s(E)$:

$EQ_1 = \{p^1 \Leftrightarrow p^2, p^1 \Leftrightarrow p^3, p^2 \Leftrightarrow p^3, r^1 \Leftrightarrow r^2, r^1 \Leftrightarrow r^3, r^2 \Leftrightarrow r^3, q^2 \Leftrightarrow q^3\}$
$EQ_2 = \{p^1 \Leftrightarrow p^3, r^1 \Leftrightarrow r^2, r^1 \Leftrightarrow r^3, r^2 \Leftrightarrow r^3, q^1 \Leftrightarrow q^2\}$
$EQ_3 = \{p^2 \Leftrightarrow p^3, r^1 \Leftrightarrow r^2, r^1 \Leftrightarrow r^3, r^2 \Leftrightarrow r^3, q^1 \Leftrightarrow q^2\}$
$EQ_4 = \{p^1 \Leftrightarrow p^2, p^1 \Leftrightarrow p^3, p^2 \Leftrightarrow p^3, r^2 \Leftrightarrow r^3, q^1 \Leftrightarrow q^2\}$

So, $\Delta_s(E) \equiv \neg r \vee (\neg p \wedge q)$, and $\Delta_s(E) \not\models \Delta^{\mathsf{diff},\oplus}_{\top}(E)$.
For ∇_s, the maximal sets of equivalences are the following ones ($p \Leftrightarrow p^1 \Leftrightarrow p^2 \Leftrightarrow p^3$ is used as a concise notation for $p \Leftrightarrow p^1, p \Leftrightarrow p^2, p \Leftrightarrow p^3$ and similarly for the other variables):

$EQ_1 = \{p \Leftrightarrow p^1 \Leftrightarrow p^2 \Leftrightarrow p^3, r \Leftrightarrow r^1 \Leftrightarrow r^2 \Leftrightarrow r^3, q \Leftrightarrow q^2 \Leftrightarrow q^3\}$
$EQ'_1 = \{p \Leftrightarrow p^1 \Leftrightarrow p^2 \Leftrightarrow p^3, r \Leftrightarrow r^1 \Leftrightarrow r^2 \Leftrightarrow r^3, q \Leftrightarrow q^1\}$,
$EQ_2 = \{p \Leftrightarrow p^1 \Leftrightarrow p^3, r \Leftrightarrow r^1 \Leftrightarrow r^2 \Leftrightarrow r^3, q \Leftrightarrow q^1 \Leftrightarrow q^2\}$
$EQ_3 = \{p \Leftrightarrow p^2 \Leftrightarrow p^3, r \Leftrightarrow r^1 \Leftrightarrow r^2 \Leftrightarrow r^3, q \Leftrightarrow q^1 \Leftrightarrow q^2\}$
$EQ_4 = \{p \Leftrightarrow p^1 \Leftrightarrow p^2 \Leftrightarrow p^3, r \Leftrightarrow r^2 \Leftrightarrow r^3, q \Leftrightarrow q^1 \Leftrightarrow q^2\}$
$EQ'_4 = \{p \Leftrightarrow p^1 \Leftrightarrow p^2 \Leftrightarrow p^3, r \Leftrightarrow r^1, q \Leftrightarrow q^1 \Leftrightarrow q^2\}$

So, $\nabla_s(E) \equiv (p \wedge \neg r) \vee (\neg p \wedge q)$, and $\nabla_s(E) \not\models \Delta^{\mathsf{diff},\oplus}_{\top}(E)$.

More generally, we can prove the following statement:

Proposition 7 *$\Delta^{\mathsf{diff},\oplus}_{\top}$, Δ_s, ∇_s are pairwise incomparable with respect to inferential power, i.e., it is not the case that for every profile E, the merged base obtained using one of these operators implies the merged base obtained using another operator among these three ones.*

Conclusion and Perspectives

In this paper we have introduced a family of model-based merging operators, relying on a set-theoretic measure of conflict. We focused on set-product as an aggregation function and considered the corresponding operator $\Delta^{\mathsf{diff},\oplus}$. A feature of this operator, typically not shared by existing model-based operators, is that it satisfies the temperance property, and as a consequence, it is strategy-proof for the weak drastic index when there are no integrity constraints. The price to be paid is a higher complexity than usual model-based operators (but similar to the one of formula-based merging operators (Everaere, Konieczny, & Marquis 2007)).

An important point of this work is that it illustrates the fact that the widely used Hamming distance (and more generally all distance-based operators whatever the distance), can be criticized for aggregation. We show through examples in this paper that using diff can allow to find subtler results.

The main perspective opened by this work is to characterize the merging scenarios requiring such subtler information, and to improve existing merging operators by taking it into account. This work calls for a number of other perspectives. Especially, there are several parameters used in the definition of $\Delta^{\mathsf{diff},\oplus}$ for which alternative choices could be made (especially, other aggregation functions, other minimality criteria for characterizing the models of the merged base). It would be interesting to determine whether some specific choices for these parameters would lead to majority-like operators or arbitration-like operators (Konieczny & Pino Pérez 2002a). Another issue for further research consists in determining rationality conditions on aggregation functions (as it has been achieved for distance-based merging operators). More generally, investigating the properties of the whole family of *diff*-based operators is an interesting issue.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. They have been partly supported by the ANR project PHAC (ANR-05-BLAN-0384).

References

Baral, C.; Kraus, S.; Minker, J.; and Subrahmanian, V. S. 1992. Combining knowledge bases consisting of first-order theories. *Computational Intelligence* 8(1):45–71.

Benferhat, S.; Dubois, D.; Kaci, S.; and Prade, H. 2002. Possibilistic merging and distance-based fusion of propositional information. *Annals of Mathematics and Artificial Intelligence* 34(1-3):217–252.

Benferhat, S.; Lagrue, S.; and Rossit, J. 2007. An egalitarist fusion of incommensurable ranked belief bases under constraints. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI'07)*, 367–372.

Booth, R. 2001. A negotiation-style framework for non-prioritised revision. In *Proceedings of the International Conference on Theoretical Aspects of Rationality and Knowledge (TARK'01)*, 137–150.

Booth, R. 2006. Social contraction and belief negotiation. *Information Fusion* 7(1):19–34.

Borgida, A. 1985. Language features for flexible handling of exceptions in information systems. *ACM Trans. on Database Syst.* 10:563–603.

Chopra, S.; Ghose, A. K.; and Meyer, T. 2006. Social choice theory, belief merging, and strategy-proofness. *Information Fusion* 7(1):61–79.

Dalal, M. 1988. Investigations into a theory of knowledge base revision: preliminary report. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI'88)*, 475–479.

Delgrande, J., and Schaub, T. 2007. A consistency-based framework for merging knowledge bases. *Journal of Applied Logic* 5(3):459–477.

Eiter, T., and Gottlob, G. 1992. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence* 57(2–3):227–270.

Everaere, P.; Konieczny, S.; and Marquis, P. 2004. On merging strategy-proofness. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, 357–367.

Everaere, P.; Konieczny, S.; and Marquis, P. 2007. The strategy-proofness landscape of merging. *Journal of Artificial Intelligence Research* 28:49–105.

Everaere, P.; Konieczny, S.; and Marquis, P. 2008. Conflict-based merging operators. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR'08)*. to appear.

Katsuno, H., and Mendelzon, A. O. 1991. Propositional knowledge base revision and minimal change. *Artificial Intelligence* 52:263–294.

Konieczny, S., and Pino Pérez, R. 2002a. Merging information under constraints: a logical framework. *Journal of Logic and Computation* 12(5):773–808.

Konieczny, S., and Pino Pérez, R. 2002b. On the frontier between arbitration and majority. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR'02)*, 109–118.

Konieczny, S.; Lang, J.; and Marquis, P. 2004. DA^2 merging operators. *Artificial Intelligence* 157:49–79.

Konieczny, S. 2004. Belief base merging as a game. *Journal of Applied Non-Classical Logics* 14(3):275–294.

Liberatore, P., and Schaerf, M. 1998. Arbitration (or how to merge knowledge bases). *IEEE Transactions on Knowledge and Data Engineering* 10(1):76–90.

Meyer, T.; Foo, N.; Zhang, D.; and Kwok, R. 2004a. Logical foundations of negotiation: Outcome, concession and adaptation. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI'04)*, 293–298.

Meyer, T.; Foo, N.; Zhang, D.; and Kwok, R. 2004b. Logical foundations of negotiation: Strategies and preferences. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR'04)*, 311–318.

Meyer, T.; Pozos Parra, P.; and Perrussel, L. 2005. Mediation using m-states. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'05)*, 489–500.

Meyer, T. 2001. On the semantics of combination operations. *Journal of Applied Non-Classical Logics* 11(1-2):59–84.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

Revesz, P. Z. 1997. On the semantics of arbitration. *International Journal of Algebra and Computation* 7(2):133–160.

Satoh, K. 1988. Non-monotonic reasoning by minimal belief revision. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 455–462.

Weber, A. 1986. Updating propositional formulas. In *Proceedings of First Conference on Expert Database Systems*, 487–500.

Winslett, M. 1988. Reasoning about action using a possible model approach. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI'88)*, 89–93.

Zhang, D.; Foo, N.; Meyer, T.; and Kwok, R. 2004. Negotiation as mutual belief revision. In *Proceedings of the American National Conference on Artificial Intelligence (AAAI'04)*, 317–322.

Activity Recognition with Intended Actions, Answer Set Programming Approach

Alfredo Gabaldon
NICTA & UNSW
Sydney, Australia
alfredo@cse.unsw.edu.au

Abstract

We consider an activity recognition problem as follows. We are given a description of the action capabilities of an agent being observed. This description includes a description of the preconditions and effects of atomic actions the agent may execute and a description of activities (sequences of actions). Given this description and a set of propositions about action occurrences and intended actions, called *history*, the problem is to determine what has already happened, what the intentions of the agent are, and what may happen as a result of the agent acting on those intentions. Our formalization is based on a recent approach to reasoning about intended actions in ($\mathcal{A}$-like) action languages with answer set programming implementations.

Introduction

We consider the following problem: given a partial record of what an agent being observed is doing, including a) intended actions, b) action executions, c) fluent values, all at various time points, determine a complete picture of what the agent has done, intends and may do in the future. This problem is what we will refer to as *activity recognition*, and our approach to it is based on logical reasoning about the observations with respect to a background knowledge base that includes a formal action theory representing how the world evolves as the agent executes actions, knowledge about non-elementary actions, called *activities*, that the agent might be executing, and a theory of intended actions.

Our approach to activity recognition is novel in that it is based on a formal theory of actions and reasoning with an explicit notion of intended actions. We discuss related work before Conclusions.

We will use a modified and extended version of the cooking example from (Kautz and Allen 1996) throughout the paper.

Example 1 The observed agent in this domain is capable of executing various meal preparation activities: it can *cook chicken marinara* which consists in *making marinara* sauce and putting it together with chicken by *mixing chicken marinara*. It can *cook fettuccini alfredo* by *making fettuccini*, *making alfredo* sauce and putting it together by *mixing fettuccini alfredo*. It can also *cook fettuccini marinara*, *cook spaghetti carbonara* and *cook chicken primavera* using similar steps. Then, if the agent declares that he intends to *make fettuccini* at time 1, and we observe that *mix chicken marinara* occurred at time 3, then the possible conclusions are that the agent intends to cook two dishes: one of the fettuccini dishes and chicken marinara. In the case that he is making fettuccini marinara, it is possible that he makes marinara sauce once, for both dishes, i.e. the two cooking activities share an action.

Reasoning about Intended Actions

The action description language $\mathcal{ALI}$, introduced in (Baral and Gelfond 2005), extends similar action languages, in particular $\mathcal{AL}$(Baral and Gelfond 2000), with the capability of reasoning about intended actions. In this section we give an overview of this language.

Like other $\mathcal{A}$-like languages, a domain description in $\mathcal{ALI}$ includes a set of *dynamic causal laws*, *static causal laws* and executability propositions. A set of such statements, called *action description*, describes a transition system which models how the world moves from one state to another as actions are executed. A separate set of constructs in the language is used to capture a *history*: a set of statements about observed values of fluents and occurrences of actions at specified time points. Given a domain description and a possibly incomplete history, the reasoning task is then to determine a complete trajectory that the world may have followed and that is compatible with the history. Additionally, the language $\mathcal{ALI}$ allows for reasoning about intended actions, thus it includes a construct for specifying in a history that at a given time the agent intends to execute a given action. The underlying principle incorporated in $\mathcal{ALI}$ states that *normally, unfulfilled intentions persist*, meaning that if the agent is not able to execute an intended action at a specified time (e.g. because the action was not executable at that time) then the intention persists until the agent successfully executes the action. The formal syntax and semantics of this language is as follows.

The signature of $\mathcal{ALI}$ consists of two disjoint, finite sets: a set of elementary action names A, and a set of symbols F, called *fluents*, which represent properties of the domain that change when actions are executed. A *fluent literal* is a

fluent f or its negation, denoted by $\neg f$. A set of literals Y is called *complete* if for every $f \in F$, $f \in Y$ or $\neg f \in Y$, and Y is called *consistent* if there is no f s.t. $f, \neg f \in Y$. A *state* is a complete and consistent set of fluent literals and represents one possible state of the domain. An *action* is a set $\{a_1, \ldots, a_n\}$ of elementary actions representing their simultaneous execution. Sequences of actions are lists of actions separated by commas and enclosed by $\langle \cdot \rangle$.

Given a signature $\Sigma = (A, F)$, a *transition diagram* over Σ is defined as a directed graph T where:

- the states of T are the states of Σ, which are denoted by σ_i's;
- the arcs of T are labeled by actions of Σ.

A path $\langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ of a transition diagram is called a *trajectory* of the domain.

As mentioned above, an action description in $\mathcal{ALI}$ consists of a set of statements understood as describing a transition diagram. These statements are of the following three forms (a_e denotes an elementary action and the l_i denote fluent literals):

1. *dynamic causal laws*: *causes*$(a_e, l_0, [l_1, \ldots, l_n])$, stating that executing a_e in a state where $l_1, \ldots, l_n$ hold causes l_0 to be true in the resulting state;
2. *static causal laws*: *caused*$(l_0, [l_1, \ldots, l_n])$, stating that l_0 is caused to hold in every state where $l_1, \ldots, l_n$ hold;
3. *executability propositions*:[1] *impossible_if*$(a_e, [l_1, \ldots l_n])$, stating that a_e cannot be executed in a state where $l_1, \ldots, l_n$ hold.

The definition of the transition diagram specified by an action description requires the following notions and notation: An action a is executable in a state σ if there is no proposition *impossible_if*$(a_e, [l_1, \ldots l_n])$ s.t. $a_e \in a$ and $\{l_1, \ldots, l_n\} \subseteq \sigma$. A set S of fluent literals is *closed under* a set Z of static causal laws if S includes the head l_0 of every static causal law s.t. $\{l_1, \ldots, l_n\} \subseteq S$. The set $Cn_Z(S)$ of *consequences* of S under Z is the smallest set of fluent literals that contains S and is closed under Z. The notation $E(a_e, \sigma)$ is used to denote the set of all literals l_0 s.t. there is a dynamic causal law *causes*$(a_e, l_0, [l_1, \ldots, l_n])$ and $\{l_1, \ldots, l_n\} \subseteq \sigma$. Moreover, $E(a, \sigma) = \bigcup_{a_e \in a} E(a_e, \sigma)$.

An action description specifies a transition diagram that satisfies certain properties. One is that all the states of the transition diagram must satisfy the static causal laws. Second, if there is a transition from σ to σ' labeled by a, then a must be executable in σ. Furthermore, σ' must include the direct effects $E(a, \sigma)$ of a, the indirect effects that follow from the static causal laws and it must contain literals that are otherwise not affected by a but are preserved by the common sense law of inertia.

Formally, the transition system specified by an action description AD is defined as follows.

[1]This name is normally used for propositions declaring when an action is executable, not impossible. But since they are used with a similar purpose, we will call these the same.

Definition 1 An action description AD with signature Σ describes the transition system $T = (S, R)$ where:

1. S is the set of all the states of Σ that are closed under the static laws of AD;
2. R is the set of all triples $\langle \sigma, a, \sigma' \rangle$ s.t. a is executable in σ and σ' is the fixpoint of the equation:

$$\sigma' = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma')) \quad (1)$$

where Z is the set of all the static causal laws in AD.

For activity recognition, we are interested in determining what an agent is doing given an action description and a history. The propositions used in $\mathcal{ALI}$to describe the history have the following form (α denotes an action sequence, i a time point, and l a fluent literal):

1. *intended*(α, i): meaning that the agent intends to execute α at time point i;
2. *happened*(α, i): meaning that α happened starting at time point i;
3. *observed*(l, i): meaning that l was observed to hold at time point i.

A *history* is then a set of propositions of the above forms. The semantics of *happened* and *observed* is defined in the usual way for $\mathcal{A}$-like languages (see Definition 2 below). The semantics of *intended* as defined in (Baral and Gelfond 2005) is based on the following assumptions:

1. once an agent establishes the intention to execute an action, it does so as soon as the action is executable;
2. for an intended sequence $\langle a_1, \ldots, a_n \rangle$, a_1 is intended first and each a_{i+1} is subsequently intended only after a_i executes;
3. if an intended action is not executable, the intention to execute it persists until it becomes possible to execute it.

A history is interpreted by trajectories of the background transition system. The following definition describes when a trajectory is a model of a history.

Definition 2 Let AD be an action description, $P = \langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ be a trajectory, H be a history and $1 \leq i \leq n$.

1. P satisfies *observed*(l, i) if l is true in σ_i ($l \in \sigma_i$). Similarly for initial state statements, P satisfies *observed*$(l, 0)$ if l is true in σ_0.
2. P satisfies *happened*$(\langle a'_1, \ldots, a'_n \rangle, i)$ if for all $1 \leq j \leq n$, $a'_j \subseteq a_{i+j-1}$. In this case, each element of a'_j is said to be *supported* at $i + j - 1$.
3. P satisfies *intended*(a, i) if
   (a) $a \subseteq a_i$; or
   (b) a is not executable in σ_{i-1} (i.e. there is a proposition *impossible_if*$(a_e, \vec{l}) \in AD$ s.t. $a_e \in a$ and $\vec{l} \subseteq \sigma_{i-1}$), $i < n$ and P satisfies *intended*$(a, i + 1)$.

   If $a \subseteq a_i$, we say that a *ends at* i+1 and that each element of a is *supported at* i.

4. P satisfies *intended*(α, i) where $\alpha = \langle a'_1, \ldots, a'_m \rangle$ and $m > 1$, if
   (a) P satisfies *intended*(a'_1, i) and
   (b) P satisfies *intended*$(\langle a'_2, \ldots, a'_m \rangle, j)$ where a'_1 ends at j in P.
   We say that α *ends at* k in P, if a'_m ends at k in P.
5. P is a model of a history H if it satisfies all the statements of H and, for all $1 \leq i \leq n$, all the elements of a_i are supported at i in P.

Notice that the definitions do not rule out empty actions (i.e. nothing occurs) from appearing in a history.

Activities

Technically, the problem of activity recognition as described above can be cast as the problem of finding models of the recorded history, after taking care of a few issues. Activity recognition differs from reasoning about histories in $\mathcal{ALI}$ in several respects. First, activity recognition is done from the perspective of an external observer of the agent that is executing the actions. Second, in addition to the background theory of actions and the statements in the history, the reasoner has additional knowledge in the form of a set of activities that the agent being observed may do and information about which actions are "purposeful."

Named Activities

Our approach to activity recognition, like other approaches, is based on the availability of a set background activities that the observed agent may do. These then serve as a hypothesis space to the recognition system. In our case these activities will be represented as pairs (s, α) where s is the name of the activity and α a sequence of actions (including other activities). Thus we extend the signature of the language with an additional set C of activity names. An action description will contain a set of pairs (s, α) with at most one pair for each $s \in C$. We will often use s to refer to an activity (s, α). Sequences α in named activities are assumed not to repeat actions.

In the cooking domain example, we have activities

$$(\textit{ccm}, \langle \textit{mk\_marinara}, \textit{mix\_chicken\_marinara} \rangle)$$

$$(\textit{cfm}, \langle \textit{mk\_fettuccini,mk\_marinara,mix\_fettuccini\_marinara} \rangle)$$

$$(\textit{cfa}, \langle \textit{mk\_fettuccini,mk\_alfredo,mix\_fettuccini\_alfredo} \rangle)$$

and so on.

We assume that only actions can be observed to occur and thus will not use activity names in *happened* statements. On the other hand, we do allow activity names in *intended* statements as part of the history, as we consider the possibility that the observed agent declares its intentions or that the activity recognizer is otherwise informed of those intentions.

Purposeful Actions

By purposeful actions we mean actions whose execution in isolation, as opposed to as part of a more complex activity, is considered reasonable. For example, one may consider the action of taking a bus as not purposeful since normally a person does not take a bus for the sake of taking a bus, but does so as part of a more complex activity such as commuting to work. Here we treat purposefulness as a fixed (non fluent) property of activities. A more elaborate treatment of purposefulness intuitively seems to require this notion to be context dependent and to be captured as a default. We will consider this more general notion of purposefulness in future work.

In our example, we declare the action of *mk_marinara* sauce as not purposeful. The activity of cooking fettuccini marinara, *cfm* above, on the other hand, is considered purposeful since it is a complete meal cooking activity. Purposefulness is subjective, one can easily elaborate the cooking scenario by adding an activity for "having a meal" in the context of which cooking a meal may no longer be considered purposeful.

Purposeful actions are simply declared to be so by means of statements of the form *purposeful*(c), where c is an action or an activity. Actions not declared to be purposeful are assumed not to be purposeful. In the next section we formally describe how knowledge about the purposefulness of actions and activities influences reasoning about intended actions.

Activity Recognition

In our formalization of activity recognition we do not allow a history to state that a named activity happened, only actions can be observed. We moreover assume that all the observed actions were intentional.

The definition of satisfaction of history statements *observed*(l, i), *happened*(a, i) and *intended*(a, i) by a trajectory is as before. While activities cannot be observed to occur, we allow that the reasoner may be informed that the observed agent intends to execute a named activity. This means that we must extend histories by allowing statements *intended*(s, i) where s is an activity name, and extend the definition of satisfaction of such statements by a trajectory P. The definition is as follows.

Definition 3 A trajectory P satisfies a statement *intended*(s, i), where (s, α) is a named activity, if P satisfies *intended*(α, i).

We will assume that any intended sequence in the history is named so that *intended*(s, i) is used instead of *intended*(α, i) for an activity (s, α). From now on we will use *actions* to refer to both non-elementary actions (sets of elementary actions) and named activities

Before we introduce models of a history, we need to introduce some terminology. We say that an action (including named activities) c *starts at* i in a trajectory P if P satisfies *intended*(c, i) but does not satisfy *intended*$(c, i-1)$, i.e. it is said to start when it becomes intended. A named activity $(s, \langle a_1, \ldots, a_n \rangle)$ *ends at* i in P if a_n ends at i in P (as in

Definition 2).[2] Furthermore, an action c (including named activities) is said to be *in progress* at k in P if c starts at i and ends at j in P and $i \leq k < j$. Henceforth we will omit a reference to P when clear from context and say c starts at i, c ends at j, etc.

Next we define the key notion of *justified actions*. This notion captures the intuition that if an action that is not purposeful is stated to have occurred or to be intended at time point i, then it must be the case that a purposeful activity is in progress at the same time i and the action is part of it. The following definition formally describes when an action is justified.

Definition 4 Let AD be an action description, P be a trajectory and c be an action.

1. c is *justified by* c (self-justified) at i if *purposeful*$(c) \in AD$;
2. c is *justified by* s at i if the following conditions are satisfied:
   (a) (s, α) is a named activity in AD,
   (b) c appears in α,
   (c) s is in progress at i,
   (d) s does not justify c at an earlier time point in its current execution, that is, if l is the latest start time of s such that $l < i$, then s does not justify c at k such that $l \leq k < i$.

We say that c is *justified* at i if c is justified by b at i for some action b.

Note that an action can be justified by more than one action at the same time instant.

We are now ready to define models of a history. This definition must take into account whether actions are justified or not for the purpose of reasoning about activity recognition.

In addition to satisfying history statements as defined above, a trajectory must satisfy a number of additional conditions to be a model. Condition (2) below precludes vacuous actions from models. Condition (3) intuitively says that for every action in progress there must be at least one action that justifies it from start to end. Condition (4) says that an activity cannot end if an action that appears in its sequence is still intended, unless that action is justified by some other activity. Finally, Condition (5) says that at the end of the trajectory, no intended actions remain.

Definition 5 A trajectory $P = \langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ is a model of a history H of an action description AD if the following conditions hold:

1. P satisfies all the statements of H;
2. for each $1 \leq i \leq n$, all the elements of a_i are supported at i;
3. for every action c such that P satisfies *intended*(c, i) and c starts at i and ends at j, there is an action c' such that c is justified by c' at k for every $i \leq k < j$;
4. for every activity $(s, \langle c_1, \ldots, c_m \rangle)$ such that s ends at $i+1$, there is no action c_k, $1 \leq k < m$, in the sequence of s such that
   (a) P satisfies *intended*(c_k, i),
   (b) c_k is justified by s at i,
   (c) there is no $s' \neq s$ such that c_k is justified by s' at i;
5. for every action c, if c is in progress at n, then c ends at $n+1$.

Example 2 Consider again the Cooking domain of Example 1. Suppose that we have a history containing the following statements:

intended(*mk_fettuccini*, 1),
intended(*mix_chicken_marinara*, 3).

If we do not allow concurrent actions, this history has no models of length less than 4. It has one model of length 4 with actions:

mk_fettuccini,
mk_marinara,
mix_chicken_marinara,
mix_fettuccini_marinara

occurring in that order. (With concurrency it can be shortened to a length of 3.) Intuitively, this means that two activities are occurring: cooking chicken marinara (*ccm*), which is in progress from time 2 to 3, and cooking fettuccini marinara (*cfm*), which is in progress from 1 to 4. The activities share the action *mk_marinara* which is justified by both *ccm* and *cfm* at time 2.

The same history has 4 models of length 5. One of them contains the actions

mk_marinara,
mk_fettuccini,
mix_chicken_marinara,
mk_marinara,
mix_fettuccini_marinara

In this model *ccm* and *cfm* are again the purposeful activities that occur. *ccm* is in progress from 1 to 3 and *cfm* from 1 to 5. This time *mk_marinara* is not shared between the activities because it occurs for *ccm* before it becomes intended for *cfm*.

In the other two models, *ccm* and cooking fettuccini alfredo (*cfa*) occur, in one *mk_marinara* occurs at 1 and *mk_fettuccini* at 2. In the other they are in the opposite order.

Formalization in ASP

Our formalization of activity recognition in Answer Set Programming captures the domain transition system of Definition 1 by means of a set of rules using the predicate *holds*(F, T) (see e.g. (Baral 2003)). We have also adapted rules defining intention for elementary actions (Baral and Gelfond 2005) and activities (Gelfond 2006). The notion of *justified* activities requires a substantial elaboration with rules to recognize intention of actions at earlier times with respect to a given history statement. In the ASP formalization below, we assume that actions contain at most one primitive action. In other words, there is no concurrency and the

[2] An action "starts" when it becomes intended even if it has not yet started to *execute*. So "starts" and "ends" mark the period between an action first becoming intended and the termination of its execution.

empty action, i.e., nothing occurs, is allowed. This permits us to treat actions as primitive objects rather than sets and makes the presentation simpler. It is not difficult, however, to extend the formalization with general concurrent actions.

Let us describe the main components of the ASP formalization. Although many rules below seem to be unsafe, we assume implicit domain predicates are used to make them safe. For computing answer sets, we use the Smodels construct `#domain` to specify a domain for all the variables that appear in the rules below. We omit domain predicates for readability but it should be kept in mind that they are needed and included in order to make all the rules safe. We start with the component that, given a statement *intended*(c, i) in the history, conjectures that some activity s that has c in its sequence is in progress.

```
inprogress(S,I) :-
        component(C,K,S),
        intended(C,I),
        K <= I,
        not other_justified(C,I,S).

other_justified(C,I,S) :-
        component(C,K,S),
        justified(C,I,S1),
        neq(S,S1).
```

Conjecturing that activities are in progress possibly leads to concluding that the conjectured activity was intended at some point:

```
intended(S,I) :-
       inprogress(S,I),
       not inprogress(S,I-1).

intended(S,I) :-
       inprogress(S,I),
       ends(S,I).
```

Next we describe the component that captures the notion of justified actions, starting with self-justified actions:

```
justified(C,I,C) :-
       inprogress(C,I),
       purposeful(C).
```

The following two rules capture item 2 of Definition 4:

```
justified(C,I,C) :-
       inprogress(C,I),
       component(C,K,S),
       inprogress(S,I),
       not justified_before(C,S,I).

justified_before(C,S,I2) :-
       start_to_now(S,I1,I2),
       justified(C,I,S),
       not other_justified(C,I,S),
       ends(C,I3),
       I1 <= I, I < I3, I3 <= I2.
```

Another part of our formalization captures reasoning about which actions are in progress and which are intended. This part can be divided into two components: given a history statement about time point i, one component does inference about what holds at time points preceding i and the other inference about time points later than i. We start with the rules for reasoning about later time points.

The following two rules directly encode the definition of *in progress*:

```
inprogress(S,I) :- starts(S,I).

inprogress(S,I) :-
       inprogress(S,I-1),
       not ends(S,I).
```

In (Gelfond 2006), the formalization of intended actions includes rules for drawing conclusions about later time points with respect to history statements. In addition to the use of *inprogress* and some other differences, we must generalize those rules to take into account reasoning about *justified* actions.

```
intended(C,I) :-
       starts(S,I),
       component(C,1,S).

intended(C2,I) :-
       inprogress(S,I),
       component(C2,K,S),
       component(C1,K-1,S),
       ends(C1,I),
       justified(C1,I-1,S).
```

The component for inference at earlier time points includes a rule saying that if an activity is in progress and it justifies one of its components that is not the first, then it is in progress in the previous time point:

```
inprogress(S,I) :-
      inprogress(S,I+1),
      intended(A,I+1),
      component(A,K,S),
      K > 1,
      justified(A,I+1,S).
```

Given history statements about the occurrence of an action, it is possible to draw inferences about the intentions of the agent before the action was executed, especially if the action is not self-justified. The following rules say, roughly, that if the action is a component of an activity and the activity has an earlier component, then either the earlier component is intended in the preceding time point or the action that occurred was intended in the preceding time point.

```
intended(A1,I) :-
      inprogress(S,I+1),
      occurs(A2,I+1),
      component(A2,K,S),
      component(A1,K-1,S),
      not -occurs(A1,I),
      not intended(A2,I).

intended(A2,I) :-
      inprogress(S,I+1),
      occurs(A2,I+1),
      component(A2,K,S),
      component(A1,K-1,S),
      not intended(A1,I).
```

Finally, the following rules capture conditions (3–5) in Definition 5:

Condition (3):

```
:- inter(C,I,I1),
   not full_just(C,I,I1).

full_just(C,I,I1) :-
     full_justified(C,C1,I,I1).

full_justified(C,C1,I,I1) :-
     justified(C,I,C1).

full_justified(C,C1,I,I1) :-
     justified(C,I,C1),
     I < I1,
     full_justified(C,C1,I+1,I1).
```

Condition (4):

```
:- ends(S,I+1),
   intended(C,I),
   justified(C,I,S),
   not other_justified(C,I,S),
   length(S,K),
   not component(C,K,S).
```

Condition (5):

```
:- intended(C,n).
```

In the above last rule, n is a constant defined to be the maximum length of the trajectories to be considered when solving an activity recognition problem.

The above set of rules will be denoted by Π_{ar}.

The encoding of a history H and an action description AD is as follows. Statements *intended*(c, i), *intended*(s, i), *happened*(a, i) and *observed*(l, i) are encoded directly as facts `intended(a,i)`, `intended(s,i)`, `happened(a,i)` and `observed(l,i)`, respectively. The set of such facts obtained from a history H is denoted by $\pi(H)$.

The translation of a domain description AD is denoted by $\pi(AD)$ and contains the following rules and facts. For each named activity $(s, \langle c_1, \ldots, c_m\rangle)$ we include the facts:

```
activity(s).
length(s,m).
component(c_1,1,s).
...
component(c_m,m,s).
```

As mentioned earlier, there is a fact `purposeful(c)` for each purposeful action or named activity c.

Dynamic causal laws, static causal laws and executability propositions are translated as follows:

For each dynamic causal law *causes*$(a, l_0, [l_1, \ldots, l_n])$ in AD:

```
holds(l_0,I+1) :-
      occurs(a,I),
      holds(l_1,I),
      ...
      holds(l_n,I).
```

For each static causal law *caused*$(l_0, [l_1, \ldots, l_n])$ in AD:

```
holds(l_0,I) :-
      holds(l_1,I),
      ...
      holds(l_n,I).
```

Finally, for each executability proposition *impossible_if*$(a, [l_1, \ldots l_n])$ in AD:

```
-occurs(a,I) :-
      holds(l_1,I),
      ...
      holds(l_n,I).
```

We also include in $\pi(AD)$ a rule for the common sense law of inertia:

```
holds(L,I+1) :-
      holds(L,I),
      not holds(-L,I+1).
```

For reasoning about intended atomic actions, we use the following set of rules, denoted by Π_I:

```
%% Observed actions are intended
intended(A,I) :- happened(A,I).
occurs(A,I) :- happened(A,I).

occurs(A,I) :-
      intended(A,I),
      not -occurs(A,I).

intended(A,I+1) :-
      intended(A,I),
      -occurs(A,I),
      not -inteded(A,I+1).

starts(A,I) :-
      intended(A,I),
      not inteded(A,I-1).

starts(A,I) :-
      intended(A,I),
      ends(A,I).

ends(A,I+1) :- occurs(A,I).

inprogress(A,I) :- starts(A,I).

inprogress(A,I) :-
      inprogress(A,I-1),
      not ends(A,I).
```

The rest of the rules in the formalization are minor auxiliary rules.

Throughout the paper we have been focusing on intention and occurrence of actions and have sidelined reasoning about the value of fluents in the states of the trajectory. Incompleteness of information about fluents is another interesting aspect of the activity recognition problem we are considering. We leave that side of the problem for the full version of this paper. The following result assumes a history with complete initial state, i.e. for every fluent f there is a statement *observed*$(f, 0)$ or *observed*$(\neg f, 0)$.

The complete formalization of an activity recognition problem in answer set programming is the program $\pi(AD, H) = \Pi_I \cup \Pi_{ar} \cup \pi(AD) \cup \pi(H)$. The models of a program $\pi(AD, H)$ induce trajectories as follows.

Definition 6 Let A be a subset of the literals of a given program $\pi(AD, H)$. A is said to define the trajectory

$\langle \sigma_0, a_1, \sigma_1, \ldots, \sigma_n, a_n \rangle$ if $\sigma_i = \{l \mid \mathit{holds}(l, i) \in A\}$ and $\mathit{occurs}(a_j, j) \in A$ for all $0 \leq i \leq n$ and $1 \leq j \leq n$

Theorem 1 For an action description AD and history H with complete initial state, a trajectory P without concurrency (i.e. all actions are singletons or empty sets) is a model of H iff P is defined by an answer set of the program $\pi(AD, H)$.

Example 3 The following are the results of some sample runs with the Cooking domain. Using the following facts:

```
intended(make_fettuccini,1).
happened(mix_chicken_marinara,3).
```

and setting the max (plus 1) trajectory length constant `const n=5` we obtain an answer set that contains:

```
inprogress(cfm,1)
inprogress(cfm,2)  inprogress(ccm,2)
inprogress(ccm,3)  inprogress(cfm,3)
inprogress(cfm,4)
justified(make_fettuccini,1,cfm)
justified(make_marinara,2,ccm)
justified(make_marinara,2,cfm)
justified(mix_chicken_marinara,3,ccm)
justified(mix_fettuccini_marinara,3,cfm)
justified(mix_fettuccini_marinara,4,cfm)
occurs(make_fettuccini,1)
occurs(make_marinara,2)
occurs(mix_chicken_marinara,3)
occurs(mix_fettuccini_marinara,4)
```

Now let us try the following history:

```
happened(make_marinara,2).
happened(make_marinara,4).
```

With `const n=5` we get no answer sets, as expected. If we increase the constant to 6 we get two answer sets. In one, *ccm* happens twice and with no action occurring at 1:

```
inprogress(ccm,2)
inprogress(ccm,3)
inprogress(ccm,4)
inprogress(ccm,5)
occurs(make_marinara,2)
occurs(mix_chicken_marinara,3)
occurs(make_marinara,4)
occurs(mix_chicken_marinara,5)
```

In the second answer set, *cfm* is in progress from 1 to 3 and *ccm* from 4 to 5.

```
inprogress(cfm,1)
inprogress(cfm,2)
inprogress(cfm,3)
inprogress(ccm,4)
inprogress(ccm,5)
occurs(make_fettuccini,1)
occurs(make_marinara,2)
occurs(mix_fettuccini_marinara,3)
occurs(make_marinara,4)
occurs(mix_chicken_marinara,5)
```

Setting `const n=7` gives us two additional answer sets:

```
inprogress(ccm,2)
inprogress(ccm,3)  inprogress(cfm,3)
inprogress(ccm,4)  inprogress(cfm,4)
inprogress(ccm,5)  inprogress(cfm,5)
inprogress(ccm,6)
occurs(make_marinara,2)
occurs(make_fettuccini,3)
occurs(make_marinara,4)
occurs(mix_fettuccini_marinara,5)
occurs(mix_chicken_marinara,6)
```

and

```
inprogress(ccm,2)
inprogress(ccm,3)  inprogress(cfm,3)
inprogress(ccm,4)  inprogress(cfm,4)
inprogress(ccm,5)  inprogress(cfm,5)
                   inprogress(cfm,6)
occurs(make_marinara,2)
occurs(make_fettuccini,3)
occurs(make_marinara,4)
occurs(mix_chicken_marinara,5)
occurs(mix_fettuccini_marinara,6)
```

Related Work

There is a substantial body of work on activity or plan recognition. The work of (Kautz and Allen 1996), from which we adapted the cooking example, was among the first to use some kind of action formalism, albeit a simple one and without considering an explicit notion of intended actions. There are some recent approaches based on Hidden Markov Models (Bui, Venkatesh, and West 2002; Blaylock and Allen 2003). These approaches cannot handle multiple activities occurring simultaneously as in our framework. Perhaps closer to our action based approach is the recent work of (Demolombe and Hamon 2002; Demolombe and Fernandez 2005). This work is based on a different action formalism, the situation calculus, and instead of notions of purposeful and justified actions, that framework relies on an explicit specification of which actions must *not occur* at particular points of an activity if is to be recognized from the history. This makes the definitions of activities more complicated and less elaboration tolerant. The approach of (Goultiaeva and Lespeance 2007) extends that of (Demolombe and Hamon 2002) by reformalizing the problem in a way that allows incremental recognition of plans. Both of these approaches require the history to be a complete prefix of an activity in order to recognize it. Thus they are not able to solve the problems in Example 3 because the history is missing some of the earlier occurring actions. These frameworks also require observations of actions to be made in the order the actions occur and cannot handle shared actions between activities. On the other hand, they can define more complex activities than sequences. A more general notion of activity is in our plans for future work.

It is worth pointing out as well that none of the above frameworks allow in the history statements of intention to execute an action or activity, since they do not employ a formal notion of intended actions. Finally, another unique feature of our approach is the direct implementation of the answer set programming formalization through execution by answer set finders like Smodels (Niemelä and Simons 1997). Nevertheless, several of the above frameworks have features we would like to integrate into our framework, e.g. activities

of a more general form than sequences, and probabilities or qualitative information useful for choosing among alternative solutions to an activity recognition problem.

Conclusion

We have introduced a new approach to activity recognition that is based on a formal theory of actions and a notion of intended actions. Our approach is based on using knowledge about the intention and the occurrences of non-purposeful actions to conjecture that more complex purposeful activities may be occurring. In addition to $\mathcal{A}$-type action language domain descriptions with a transition system-based semantics, we provide a formalization in answer set programming that can be readily used to solve problems by submitting the formalization to an answer set finder such as Smodels (Niemelä and Simons 1997). We have illustrated our approach with problems in a meal cooking domain and included some sample results obtained with Smodels.

This framework can be extended in many possible ways. For instance, if one considers a case where the observed agent's actions can fail, then we would need to recognize or predict that the agent repeated the execution of an action because the first try failed, and that because it failed, its intention to execute that action persisted after the first try. Another extension to consider is where the observed agent may abandon an intended activity in the middle of the execution and start executing some other activity. We plan to look at these and many other interesting possibilities in future work.

References

Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer.

Baral, C., and Gelfond, M. 2005. Reasoning about intended actions. In *Procs. of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 689–694.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.

Blaylock, N., and Allen, J. 2003. Corpus-based statistical goal recognition. In *Proceedings Procs. of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*.

Bui, H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*.

Demolombe, R., and Fernandez, A. 2005. Intention recognition in the situation calculus and probability theory frameworks. *Computational Logic in Multi Agent Systems* 358–372.

Demolombe, R., and Hamon, E. 2002. What does it mean that an agent is performing a typical procedure? a formal definition in the situation calculus. In *Procs. of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 905–911.

Geib, C. W., and Steedman, M. 2007. On natural language processing and plan recognition. In *Procs. of the International Joint Conference on Artificial Intelligence (IJCAI'07)*.

Gelfond, M. 2006. Going places - notes on a modular development of knowledge about travel. In *Procs. of the AAAI Spring Symposium "Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering"*.

Goultiaeva, A., and Lespeance, Y. 2007. Incremental plan recognition in an agent programming framework. In *Procs. of AAAI 2007 Workshop on Plan, Activity, and Intent Recognition (PAIR'07)*.

Kautz, H., and Allen, J. F. 1996. Generalized plan recognition. In *Procs. of 13th National Conference on Artificial Intelligence (AAAI'96)*, 32–37.

Niemelä, I., and Simons, P. 1997. Smodels–an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th Int'l Conf. on Logic Programming and Non-Monotonic Reasoning*, 420–429.

Model-Based Contractions for Description Logics

Martín O. Moguillansky and **Marcelo A. Falappa** and **Guillermo R. Simari**
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Department of Computer Science and Engineering
Universidad Nacional del Sur (UNS), Bahía Blanca, ARGENTINA
e-mail: {mom, maf, grs}@cs.uns.edu.ar

Abstract

Model-Based Contractions are a formalism –based on model-theoretic semantics– which characterizes an operation that modifies a knowledge base to avoid the satisfiability of a given expression. In the context of ontology revision, model-based contractions are a functional component that yields an ontology ready to evolve consistently. In this work we formalize the theory for contractions providing a model, variations, and their axiomatic characterization. Afterwards an algorithm towards its realization is proposed. Such algorithm has no further impact in computability, since it works on top of the satisfiability checking of the incoming information.

Introduction

We propose an ontology change operator which models the dynamics of the knowledge represented by ontologies. Such operator allow the addition of axioms (terminological descriptions) and assertions to the respective ontology in a consistent manner. In this sense, we follow consistency by assuming the change operator to be applied to ontologies for which consistency is a critical matter due to either, the domain they model, or the systems referring to them. Because of its highly reusable distributed nature, this kind of ontologies should pass only through consistent intermediate states of an evolutional process.

For an ontology change operation, it is important to follow the minimal change principle. This is related to the avoidance of instance data loss, *i.e.*, assertions, whenever it is possible. When some change is stated, as a result of that, some axioms may end up unsatisfiable, turning the ontology to incoherency. Here another important issue is unveiled regarding how the old and the new information is considered.

For the sake of ontology evolution, the new information is some new appreciation of the world produced by a change in the shared domain, or in its conceptualization. This means that the new knowledge should be prioritized over the older. Therefore, when some axiom experiences satisfiability loss it should be considered outdated wrt. the current state of the shared domain. Hence, its condition is analyzed in order to automatically restore its satisfiability, unless the ontology engineer interprets it needs to be treated separately.

Therefore, realization of the ontology change operation will rely on two sub-operations, the first one, namely *model-based contractions*, will modify the ontology accordingly to the incoming information such that it could be consistently and coherently incorporated later, whereas the latter sub-operation[1] will restore the satisfiability of the outdated axioms in order to coherently reincorporate them along with the new information in a consistent manner.

In this work, we formally define the model-based contraction operator, providing its axiomatic characterization, and an algorithm for its realization. Consequently, as description logic reasoners usually deal with huge ontologies, it is of utmost importance to provide an algorithm capable of reusing previous computations. This means that such algorithm should work on top of the satisfiability checking of the incoming information.

Description Logics Brief Overview

The following constitutes a very brief overview of the description logics (DLs) used in this paper, for more detailed information refer to (Baader *et al.* 2003). An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty domain $\Delta^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}}$ that maps every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual to an element of $\Delta^{\mathcal{I}}$.

The basic description language $\mathcal{AL}$ is formed by concept descriptions according to the syntax $C, D \longrightarrow A|\bot|\top|\neg A|C \sqcap D|\forall R.C|\exists R.\top$ where A is an atomic concept, R is an atomic role; and the interpretation function $\cdot^{\mathcal{I}}$ is extended to the universal concept as $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$; the bottom concept as $\bot^{\mathcal{I}} = \emptyset$; the atomic negation as $(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$; the intersection as $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$; the universal quantification as $(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$; and the limited existential quantification as $(\exists R.\top)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \exists b.(a, b) \in R^{\mathcal{I}}\}$.

More expressive languages are possible by adding different constructors to $\mathcal{AL}$ like union of concepts (identified as $\mathcal{U}$), interpreted as $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$; full existential quantification ($\mathcal{E}$), interpreted as $(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \exists b.(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$; full negation or complement ($\mathcal{C}$), interpreted as $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$; and more. Extend-

[1] The formalizations for the second sub-operation and the ontology change operation are out of the scope of this work.

ing $\mathcal{AL}$ by any of the above yields a particular language respectively named by a string of the form $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}]$. DLs considered in this work follow such kind of specifications.

A knowledge base (KB) is a pair $\Sigma = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \rangle$, where $\mathcal{T}_\Sigma$ represents the TBox, containing the terminologies (or axioms) of the application domain, and $\mathcal{A}_\Sigma$, the ABox, which contains assertions about named individuals in terms of these terminologies. Regarding the TBox $\mathcal{T}_\Sigma$, axioms are sketched as $C \sqsubseteq D$ and $C \equiv D$, therefore, an interpretation $\mathcal{I}$ satisfies them whenever $C^\mathcal{I} \subseteq D^\mathcal{I}$ and $C^\mathcal{I} = D^\mathcal{I}$ respectively. An interpretation $\mathcal{I}$ is a model for the TBox $\mathcal{T}_\Sigma$ if $\mathcal{I}$ satisfies all the axioms in $\mathcal{T}_\Sigma$. Thus, the TBox $\mathcal{T}_\Sigma$ is said to be satisfiable if it admits a model. Besides, in the ABox $\mathcal{A}_\Sigma$, $\mathcal{I}$ satisfies $C(a)$ if $a \in C^\mathcal{I}$, and $R(a, b)$ if $(a, b) \in R^\mathcal{I}$. An interpretation $\mathcal{I}$ is said to be a model of the ABox $\mathcal{A}_\Sigma$ if every assertion of $\mathcal{A}_\Sigma$ is satisfied by $\mathcal{I}$. Hence, the ABox $\mathcal{A}_\Sigma$ is said to be satisfiable if it admits a model. Finally, regarding the entire KB, an interpretation $\mathcal{I}$ is said to be a model of Σ if every statement in Σ is satisfied by $\mathcal{I}$, and Σ is said to be satisfiable if it admits a model.

In the rest of this article, we write $\mathcal{L}$ to identify some specific $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}]$ DL. When necessary, we will make difference in a logic $\mathcal{L}$ between the representation for axioms, writing $\mathcal{L}_\mathcal{T}$, and the $\mathcal{L}$ logic for assertions, noted as $\mathcal{L}_\mathcal{A}$. A KB contains implicit knowledge that is made explicit through inferences. The notion of semantic entailment is given by $\Sigma \models \alpha$, meaning that every model of the KB $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$ is also a model of the sentence $\alpha \in \mathcal{L}$.

(Semantic Entailment) $\Sigma \models \alpha$ *iff* $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\{\alpha\})$

The semantics of an ABox in DLs (as every ontological language), are characterized by the open world assumption (OWA), this means that absence of information in an ABox means nothing but lack of knowledge, in contrast to instances in databases where absence of information is interpreted as negative information.

Remark 1 (Restriction Towards a Practical Approach) *For some description languages (like $\mathcal{ALC}$), every satisfiable KB is known to have infinitely many models most of which are infinite. In order to provide a practical approach, we will restrict this work to finite sets of finite models. Besides, unique name assumption (UNA) is also assumed in order to assure that each individual (in the world) will map to a unique individual name. Finally, we will assume that the representation of every KB taken into consideration is made with an acyclic TBox.*

Given the assumptions above, a query α to the KB Σ, noted as $\Sigma \stackrel{?}{\models} \alpha$, is solved by checking if every element $\mathcal{I} \in \mathcal{M}(\Sigma)$ (*i.e.*, every model of Σ), is also a model of α. If this is true, the query is said to be satisfied, namely $\Sigma \models \alpha$, and α inferred, being `YES` the answer. To the contrary, if $\Sigma \models \neg\alpha$, the query is not satisfied, being `NO` its answer. Finally, if none of the previous are verified, *i.e.*, $\Sigma \not\models \alpha$ and $\Sigma \not\models \neg\alpha$, then the query is answered as `UNKNOWN`.

Intuitions for Ontology Change Operations

Given a DL $\mathcal{L}$, a consistent KB $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$ and a satisfiable sentence $\varphi \in \mathcal{L}$, we want Σ to evolve towards a new KB $\Sigma^R \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$ such that $\Sigma^R \models \varphi$, or equivalently, by reduction to unsatisfiability, $\Sigma^R \cup \{\neg\varphi\}$ has no models. An intuitive solution is to avoid the inference of $\neg\varphi$ such that the further inclusion of φ would end up in a consistent KB. Therefore, this change operation would be composed by two main sub-operations, passing from satisfaction of $\neg\varphi$, to uncertainty, and afterwards to satisfaction of φ. From a model-theoretic viewpoint (see Fig. 1), this is analogous to pass from a state in which every model satisfies $\neg\varphi$, through an intermediate state in which some models does not satisfy $\neg\varphi$, to a final state in which no model satisfies $\neg\varphi$, exactly as reduction to satisfiability requires.

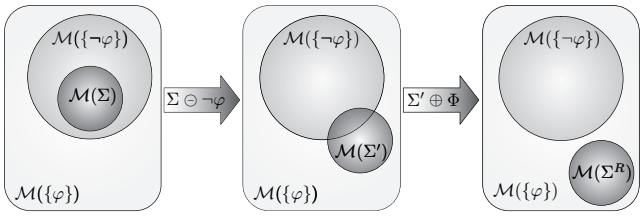


Figure 1: The operation $\mathfrak{C}^\Sigma(\varphi)$ and its two sub-operations.

As stated in the introductory section, we want to generate consistent outcomes from each sub-operation, preserving coherency and consistency of the final evolved KB. In this sense, as an effect of the first sub-operation, if an axiom is predicted to become unsatisfiable –considered along with the incoming sentence φ– it would be eliminated by the first sub-operation. Afterwards, the second sub-operation would repair such axioms and incorporate them along with φ to the resultant KB.

Satisfiability restoration of a given axiom may be achieved by considering its maximal consistent fragment. For this matter, we can take advantage of the advances achieved in the last years in the area of ontology debugging (Schlobach & Cornet 2003; Kalyanpur *et al.* 2006).

After the first sub-operation is applied, a second sub-operation consistently adds a set $\Phi = \{\varphi\} \cup \mathcal{T}_\mathcal{Z}$, where φ stands for the original expression, and $\mathcal{T}_\mathcal{Z}$, for the set of repaired axioms eliminated from Σ. Finally, the *ontology change operation* would be: $\mathfrak{C}^\Sigma(\varphi) = (\Sigma \ominus \neg\varphi) \oplus \Phi$, where "$\ominus$" refers to the first sub-operation, namely *model-based contraction*. The second sub-operation "$\oplus$" is out of the scope of this paper, and is part of our future work to formalize the specification of the *ontology change operator* $\mathfrak{C}$.

The following example shows, in an intuitive manner, how the ontology change operation would operate.

Example 1 *Let $\Delta^\mathcal{I} = \{a\}$ be a domain for the KB $\Sigma = \langle \{C \sqsubseteq D_1,\ D_3 \sqsubseteq C \sqcap \neg D_2\},\ \{C(a), D_1(a),\ \neg D_2(a),\ D_3(a)\}\rangle$. We want to consistently integrate the axiom $C \sqsubseteq D_1 \sqcap D_2$ to the KB. Since $C \sqsubseteq D_1$ is part of the TBox, the sentence φ to incorporate will be just $C \sqsubseteq D_2$. Thus, by effect of an ontology change operation $\mathfrak{C}^\Sigma(C \sqsubseteq D_2)$, we want to achieve a KB Σ^R such that $\Sigma^R \models \varphi$.*

By reduction to unsatisfiability, $\Sigma^R \cup \{(C \sqcap \neg D_2)(x)\}$ for a free variable x, should have no models. Consequently,

we will first generate a contracted KB $\Sigma' = \Sigma \ominus (C \sqcap \neg D_2)(x)$ such that $\Sigma' \models \neg\varphi$ is not verified. Two minimal proofs[2] *arise for $\neg\varphi$: $\{C(a), \neg D_2(a)\}$ and $\{D_3 \sqsubseteq C \sqcap \neg D_2, D_3(a)\}$. Suppose now $\neg D_2(a)$ is retired from the former set, and the axiom $D_3 \sqsubseteq C \sqcap \neg D_2$ from the latter – given that it becomes unsatisfiable if considered along with $C \sqsubseteq D_2$. Hence, after the contraction "$\ominus$", the resultant KB would be $\Sigma' = \langle\{C \sqsubseteq D_1\}, \{D_1(a), C(a), D_3(a)\}\rangle$.*

From the second stage of the ontology change operation "$\mathfrak{C}$", the sentence $D_3 \sqsubseteq C \sqcap \neg D_2$ needs to be repaired. A possible solution may be to change it to $D_3 \sqsubseteq C$.[3] *Now, by effect of the final sub-operation "$\oplus$", we can consistently add the sentence φ along with the repaired axiom. Hence, the final KB would be $\Sigma^R = \langle\{C \sqsubseteq D_1 \sqcap D_2,\ D_3 \sqsubseteq C\}, \{D_1(a), C(a), D_3(a)\}\rangle$.*

Model-Based Contractions

Belief bases are sets of formulae not closed under logical consequence, containing implicit beliefs that are made explicit through inferences. Conforming the AGM model (Alchourrón, Gärdenfors, & Makinson 1985) of theory change, Kernel Contractions (Hansson 1994) are a construction for contracting belief bases based on the following intuition: in order to avoid the KB to infer a given sentence α, at least one element of each α-kernel (minimal α proof) is removed. Afterwards, no proof for α will appear in the resultant KB.

Applying directly kernel contractions to ontology languages –considering *open-world semantics*– important concerns appear. For instance, kernel contractions are not designed to reason following case analysis as done by semantic entailment. This problem is made clear below.

Example 2 *Let $\Sigma = \langle\emptyset, \{SU(a,b),\ SU(b,c),\ SU(c,d),\ Re(a), \neg Re(c), \neg Re(d)\}\rangle$ be a KB where SU and Re stand for "supervised-by" and "researcher", respectively. Now suppose a new regulation poses that no academic researcher might be supervised by a non-researcher. Therefore, we would need to provoke the ontology to evolve by an operation $\mathfrak{C}^{\Sigma}(\varphi)$, where $\varphi = Re \sqsubseteq \forall SU.Re$. In consequence, we should first apply a model-based contraction $\Sigma \ominus \alpha$, where $\alpha = (Re \sqcap \exists SU.\neg Re)(x)$, avoiding any certainty about the existence of some researcher who is supervised by a non researcher.*

Assuming a domain $\Delta^{\mathcal{I}} = \{a, b, c, d\}$, the only possibility to answer α is by case analysis on the interpretation $\mathcal{I}$ of the individual name b regarding the concept Re. That is, in case $b \in Re^{\mathcal{I}}$, the related minimal proof would be $\{Re(b), SU(b,c), \neg Re(c)\}$, whereas if $b \in \neg Re^{\mathcal{I}}$, the related minimal proof would be $\{Re(a),\ SU(a,b), \neg Re(b)\}$. Finally, the query $\Sigma \models \alpha$ is satisfied.

Queries in such situations cannot be correctly answered if case analysis is not performed. In such a case, kernel contractions would not recognize any α-kernel and therefore, the inference of α could not be avoided. An alternative for it could be its redefinition by use of the semantic entailment "$\models$", but again some new problems would appear since it would no longer find minimal proofs but minimal sets. Each of those sets will contain an incomplete proof considering an individual $a \in C^{\mathcal{I}}$ and the opposite proof for the case in which $a \in \neg C^{\mathcal{I}}$.

Example 3 (Ex. 2 cont.) *A set $\mathcal{K} = \{SU(a,b),\ SU(b,c),\ Re(a),\ \neg Re(c)\}$ is a minimal subset $\mathcal{K} \subseteq \Sigma$ entailing α, i.e., $\Sigma \models \alpha$.*

Example 4 *Given $\Sigma = \langle\emptyset, \mathcal{A}_{\Sigma}\rangle$, where $\mathcal{A}_{\Sigma} = \{SU(a,b),\ SU(b,c),\ SU(c,d),\ SU(e,b),\ SU(b,f),\ Re(a),\ Re(e),\ \neg Re(c),\ \neg Re(d),\ \neg Re(f)\}$, and the sentence $\alpha = (Re \sqcap \exists SU.\neg Re)(x)$. Considering a domain $\Delta^{\mathcal{I}} = \{a, b, c, d, e, f\}$, four different minimal sets for α appear:*

$\mathcal{K}_1 = \{SU(a,b), SU(b,c), Re(a), \neg Re(c)\}$
$\mathcal{K}_2 = \{SU(a,b), SU(b,f), Re(a), \neg Re(f)\}$
$\mathcal{K}_3 = \{SU(e,b), SU(b,c), Re(e), \neg Re(c)\}$
$\mathcal{K}_4 = \{SU(e,b), SU(b,f), Re(e), \neg Re(f)\}$

From the previous examples, in order to avoid $\Sigma \models \alpha$ we may eliminate at least one belief from each set $\mathcal{K}$ such that no minimal set for α would exist. Although this method "seems to successfully[4]" achieve a contraction operation, it is not so reasonable since this would require exponential space to compute. That is, $O(m^n)$, where m is an average of minimal α-proofs valid in each model $\mathcal{I} \in \mathcal{M}(\Sigma)$, and n is the cardinality of $\mathcal{M}(\Sigma)$. Hence, adapting the classical theory of kernel contractions to ontology languages, by simply changing its classical consequence operator "$\vdash$" to semantic entailment, seems to be quite unnatural and very inefficient.

Contractions Machinery

Our interest relies on a construction requiring polynomial space to compute. That is, a different construction is needed over which some new methodology is to be applied. This construction should rely on the KB and some sole model. In consequence, the methodology would run in polynomial space wrt. to the number of minimal α-proofs appearing in the construction determined by an appropriate model. Such an approach would allow to operate the change in a more efficient, intuitive, and natural manner; irrespective of considering a possibly infinite set $\mathcal{M}(\Sigma)$.

In general, our proposal is based on a similar intuition to that of Kernels, but since semantic entailment considers an inference by verifying its satisfiability in every model, minimal proofs for α should be found in every "extension" of the KB Σ. Such KB extension, namely $\Sigma_{\mathcal{I}}$, will contain the KB Σ extended by the assumptions made in the related model $\mathcal{I} \in \mathcal{M}(\Sigma)$.

Definition 1 (KB Extension "$\Sigma_{\mathcal{I}}$") *Let $\mathcal{L}$ be a DL, $\Sigma \subseteq \mathcal{L}_{\mathcal{T}} \times \mathcal{L}_{\mathcal{A}}$, a knowledge base such that $\Sigma = \langle\mathcal{T}_{\Sigma}, \mathcal{A}_{\Sigma}\rangle$, and $\mathcal{M}(\Sigma)$, its set of models. The finite Σ extension by a finite model $\mathcal{I} \in \mathcal{M}(\Sigma)$ is a KB $\Sigma_{\mathcal{I}} = \langle\mathcal{T}_{\Sigma}, \mathcal{A}_{\Sigma} \cup \mathrm{S}_{\Sigma_{\mathcal{I}}}\rangle$ where:*

$$\mathrm{S}_{\Sigma_{\mathcal{I}}} = \{C(a) | \forall a \in \Delta^{\mathcal{I}}, \text{where } a \in C^{\mathcal{I}} \text{ and } C(a) \notin \mathcal{A}_{\Sigma}\} \cup \{R(a,b) | \forall a, b \in \Delta^{\mathcal{I}}, \text{where } (a,b) \in R^{\mathcal{I}} \text{ and } R(a,b) \notin \mathcal{A}_{\Sigma}\}.$$

[2] Minimal proofs will be formally defined later.

[3] A brief discussion on this matter is given by the end of this article.

[4] It will be clear later that such methodology does not guarantee success.

The set $\mathrm{S}_{\Sigma_\mathcal{I}}$ is referred as "set of assumptions" determined by the finite model $\mathcal{I}$.

From the definition above, the extended ABox $\mathcal{A}_{\Sigma_\mathcal{I}}$ of $\Sigma_\mathcal{I}$ is composed by two different kinds of assertions: factual assertions, *i.e.*, those contained in $\mathcal{A}_\Sigma$; and non-factual assertions, or so called set of assumptions $\mathrm{S}_{\Sigma_\mathcal{I}}$.

Whenever $\Sigma \models \alpha$, for a KB Σ and a sentence α, we may identify different explanations for α conformed as minimal sets, inside some KB extension from a model $\mathcal{I}$.

Definition 2 (Extended α-Kernel "EαK") *Given a DL $\mathcal{L}$, a KB $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$, its extension $\Sigma_\mathcal{I} = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \cup \mathrm{S}_{\Sigma_\mathcal{I}} \rangle$ and a sentence $\alpha \in \mathcal{L}$. An extended α-Kernel (for short EαK), is a KB $\mathcal{K} = \langle \mathcal{T}_\mathcal{K}, \mathrm{F}_\mathcal{K} \cup \mathrm{S}_\mathcal{K} \rangle$ verifying the following conditions:*

(1) $\mathcal{K} \subseteq \Sigma_\mathcal{I}$, *i.e.,* $\mathcal{T}_\mathcal{K} \subseteq \mathcal{T}_\Sigma$, $\mathrm{F}_\mathcal{K} \subseteq \mathcal{A}_\Sigma$, *and* $\mathrm{S}_\mathcal{K} \subseteq \mathrm{S}_{\Sigma_\mathcal{I}}$.

(2) $\mathcal{K} \models \alpha$.

(3) *There is no $\mathcal{K}' \subset \mathcal{K}$ such that $\mathcal{K}' \models \alpha$.*

From Def. 2, given an EαK $\mathcal{K} = \langle \mathcal{T}_\mathcal{K}, \mathcal{A}_\mathcal{K} \rangle$, the factual assertions $\mathrm{F}_\mathcal{K} \subseteq \mathcal{A}_\mathcal{K}$ may be identified as $\mathcal{A}_\Sigma \cap \mathcal{A}_\mathcal{K}$. The set containing every EαK in a KB extension is defined as follows.

Definition 3 (Set of EαKs "${\Sigma_\mathcal{I}}^{\Perp}\alpha$") *Given a DL $\mathcal{L}$, a KB $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$, and a sentence $\alpha \in \mathcal{L}$. The set ${\Sigma_\mathcal{I}}^{\Perp}\alpha \subseteq 2^{\mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}}$ is the set of every EαK in the KB extension $\Sigma_\mathcal{I}$.*

Different models may determine different sets of EαKs for each associated KB extension, as is shown below.

Example 5 (Ex. 4 cont.) *Consider $\mathcal{I} \in \mathcal{M}(\Sigma)$ the model generating the KB extension $\Sigma_\mathcal{I} = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \cup \mathrm{S}_{\Sigma_\mathcal{I}} \rangle$, where $\mathrm{S}_{\Sigma_\mathcal{I}} = \{\}$. Therefore, the related EαKs are the four $\mathcal{K}_i \subseteq \Sigma_\mathcal{I}$ detailed in Ex. 4. Later on, ${\Sigma_\mathcal{I}}^{\Perp}\alpha = \{\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4\}$.*

Consider now, a different model $\mathcal{I}' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$, where $\Delta^{\mathcal{I}'} = \Delta^\mathcal{I}$, and $b \in \neg Re^\mathcal{I}$. Then, for the related KB extension $\Sigma_{\mathcal{I}'} = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \cup \mathrm{S}_{\Sigma_{\mathcal{I}'}} \rangle$, it follows $\mathrm{S}_{\Sigma_{\mathcal{I}'}} = \{\neg Re(b)\}$. Finally, ${\Sigma_{\mathcal{I}'}}^{\Perp}\alpha = \{\{Re(a), SU(a,b), \neg Re(b)\}, \{Re(e), SU(e,b), \neg Re(b)\}\}$.

We identify as *complete EαK* to an EαK containing all the necessary knowledge to infer α with no need to make case analysis, *i.e.*, no extra assumptions are required to be done. From the example above, each EαKin ${\Sigma_{\mathcal{I}'}}^{\Perp}\alpha$ is complete.

We look for a KB extension in which every complete EαK is contained. In this sense, we can apply a methodology running in polynomial space wrt. to the number of EαKs in the KB extension. It is clear that not every model allows to determine such a KB extension. Thus, it is needed to extend the KB regarding some appropriate model $\mathcal{I}$.

In this sense, the appropriate KB extension may be determined by upper and lower bounds. That is, the KB extension should contain only the necessary non-factual assertions to (completely) explain α (*i.e.*, the upper bound); but in the other hand, every EαK should be completed using only non-factual assertions from the KB extension (*i.e.*, the lower bound). In the same manner, the model $\mathcal{I}$ determining such a KB extension should also consider an appropriate minimal domain. That is, the smallest domain which suffices to identify every complete EαK.

Definition 4 (α-Minimal Extension & α-Minimal Model) *Given a sentence α, a KB Σ, and a model $\mathcal{I} \in \mathcal{M}(\Sigma)$. The KB extension $\Sigma_\mathcal{I} = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \cup \mathrm{S}_{\Sigma_\mathcal{I}} \rangle$ is α-minimal iff it follows:*

(1) $\mathrm{S}_{\Sigma_\mathcal{I}} = \bigcup_{\mathcal{K} \in ({\Sigma_\mathcal{I}}^{\Perp}\alpha)} \mathrm{S}_\mathcal{K}$, *and*

(2) *there is no $\mathcal{I}' \in \mathcal{M}(\Sigma)$, where $\Delta^\mathcal{I} \subseteq \Delta^{\mathcal{I}'}$, such that for the related KB extension $\Sigma_{\mathcal{I}'}$ it holds (1) and $\Sigma_\mathcal{I} \subset \Sigma_{\mathcal{I}'}$.*

Consequently, the α-minimal model $\mathcal{I}$ is noted as $\mathcal{I}^\alpha$, and the associated α-minimal extension $\Sigma_\mathcal{I}$ is referred as $\Sigma_{\mathcal{I}^\alpha}$.

Proposition 1 *If $(\Sigma_{\mathcal{I}^\alpha} \subseteq \Sigma_\mathcal{I})$ then $({\Sigma_{\mathcal{I}^\alpha}}^{\Perp}\alpha = {\Sigma_\mathcal{I}}^{\Perp}\alpha)$.*

In particular, we are interested in the set of EαKs "${\Sigma_{\mathcal{I}^\alpha}}^{\Perp}\alpha$" determined by an α-minimal model $\mathcal{I}^\alpha$. In the following example it is shown how the structure proposed so far is built. Note that, although more than one model is considered in the examples (and thus, more than one KB extension), the theory requires just one single model.

Example 6 *Consider $\Sigma = \langle \{C \sqsubseteq D\}, \{C(a), D(b)\} \rangle$, $\alpha = D(a)$, and models of domain $\{a, b\}$, determining: $\mathrm{S}_{\mathcal{I}_1} = \{C(b), D(a)\}$, and $\mathrm{S}_{\mathcal{I}_2} = \{\neg C(b), D(a)\}$. Thus, for the KB extensions $\Sigma_1 = \langle \{C \sqsubseteq D\}, \{C(a), D(b), C(b), D(a)\} \rangle$, and $\Sigma_2 = \langle \{C \sqsubseteq D\}, \{C(a), D(b), \neg C(b), D(a)\} \rangle$, their respective sets of EαKs will be ${\Sigma_1}^{\Perp}D(a) = \{\langle \{C \sqsubseteq D\}, \{C(a)\} \rangle, \langle \{\}, \{D(a)\} \rangle\} = {\Sigma_2}^{\Perp}D(a)$.*

Note that there is just one α-minimal extension $\Sigma_{\mathcal{I}^\alpha} = \langle \{C \sqsubseteq D\}, \{C(a), D(b), D(a)\} \rangle$. Hence, its related set of EαKs ${\Sigma_{\mathcal{I}^\alpha}}^{\Perp}D(a)$ coincides with ${\Sigma_1}^{\Perp}D(a)$ and ${\Sigma_2}^{\Perp}D(a)$.

Example 7 (Ex. 2 cont.) *Let $\mathcal{I}_1^\alpha, \mathcal{I}_2^\alpha \in \mathcal{M}(\Sigma)$ be the two α-minimal models with domain $\Delta^\mathcal{I}$, such that $b \in Re^{\mathcal{I}_1^\alpha}$ and $b \in \neg Re^{\mathcal{I}_2^\alpha}$. One E$\alpha$K appears in each α-minimal extension: $\mathcal{K}_1 = \langle \{\}, \{SU(b,c), \neg Re(c), Re(b)\} \rangle$, $\mathcal{K}_1 \subseteq \Sigma_{\mathcal{I}_1^\alpha}$.*

$\mathcal{K}_2 = \langle \{\}, \{Re(a), SU(a,b)\} \cup \{\neg Re(b)\} \rangle$, $\mathcal{K}_2 \subseteq \Sigma_{\mathcal{I}_2^\alpha}$. Finally, the related sets of EαKs are: ${\Sigma_{\mathcal{I}_1^\alpha}}^{\Perp}\alpha = \{\mathcal{K}_1\}$, and ${\Sigma_{\mathcal{I}_2^\alpha}}^{\Perp}\alpha = \{\mathcal{K}_2\}$.

In order to avoid α inferences, we need to analyze every EαK in some set ${\Sigma_\mathcal{I}}^{\Perp}\alpha$. A function "$\gamma$", namely *selection function*, determines the appropriate model from where the KB extension and the set of EαKs are built. Such function should apply the restrictions in Def. 4 guided by some *model preference criterion*, namely "$\prec$", used to univocally determine the "most profitable" α-minimal model.

Definition 5 (Model Selection Function "γ") *Let Σ be a KB, α, a sentence, and "$\prec$", a model preference criterion. A function "γ" is a "model selection function" determined by "$\prec$" iff $\gamma^\alpha(\Sigma) = \mathcal{I}^\alpha \in \mathcal{M}(\Sigma)$, where $\mathcal{I}^\alpha$ is an α-minimal model and for no other model $\mathcal{I} \in \mathcal{M}(\Sigma)$, it follows $\mathcal{I} \prec \mathcal{I}^\alpha$. The selected model $\mathcal{I}^\alpha$ will be noted as $\mathcal{I}^\gamma$.*

As stated before, the intuition behind a model-based contraction is to impact the cardinality of the set of models satisfying the KB such that some new admitted model will fail to satisfy α in the resultant KB. For this matter, we will generate the set of EαKs ${\Sigma_{\mathcal{I}^\gamma}}^{\Perp}\alpha$ of the KB extension obtained from the model $\mathcal{I}^\gamma$ selected by the function "γ". Afterwards, we will define a mapping "σ" from the set ${\Sigma_{\mathcal{I}^\gamma}}^{\Perp}\alpha$ to a sub-KB to be further eliminated from the original KB.

In this sense, two levels of deletions are to be considered: (1) axioms, or (2) assertions supporting the inference of α from every EαK in the selected KB extension. The foundations of such kind of contractions have been explained before: axioms are considered outdated and assertions are chosen when no axiom is considered in the EαK.

Definition 6 (Model Incision Function "σ") *Let $\mathcal{L}$ be a DL, $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$, a KB, $\alpha \in \mathcal{L}$, a sentence, $\mathcal{I}^\gamma$, the model selected by the model selection function "γ", and $\sigma : 2^{\mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}} \longrightarrow \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$, a function mapping from $(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$ to a KB $\langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$. Then, "$\sigma$" is a "model incision function" iff it verifies:*

(1) $\sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha) \subseteq (\bigcup_{\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)} \mathcal{K}) \cap \Sigma$.

(2) *For all EαK $\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$ it follows:*

a) $\mathcal{T}_\mathcal{K} \subseteq \mathcal{T}_\sigma$, *and*

b) if $\mathcal{T}_\mathcal{K} = \emptyset$ *and* $\mathrm{F}_\mathcal{K} \neq \emptyset$ *then* $\mathrm{F}_\mathcal{K} \cap \mathcal{A}_\sigma \neq \emptyset$.

Note that, from Def. 6, the fact of considering assertions only when no axiom exists, may be used to specify the model preference criterion "$\prec$" from Def. 5. Therefore, an option to identify the "most profitable" α-minimal model may be to analyze which $\mathcal{I}^\alpha$ leads to the KB extension with less EαKs $\mathcal{K}$ verifying $\mathcal{T}_\mathcal{K} = \emptyset$. This would avoid instance data loss, passing more axioms to be debugged by the second change sub-operation.

Definition 7 (Model-Based Contraction) *Let $\mathcal{L}$ be a DL, $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$, a KB, $\alpha \in \mathcal{L}$, a sentence, $\Sigma_{\mathcal{I}^\gamma}$, the KB extended through the selected model $\mathcal{I}^\gamma$; and "σ", a model incision function. The operator "$\ominus_\sigma$", referred as model-based contraction determined by "σ", is defined as $\Sigma \ominus_\sigma \alpha = \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$.*

Finally, "$\ominus$" is a "model-based contraction operator" for Σ iff there exists a model incision function "σ" such that $\Sigma \ominus \alpha = \Sigma \ominus_\sigma \alpha$ for all sentence α.

Example 8 (Ex. 6 cont.) *From Def. 5 we have only one α-minimal model $\mathcal{I}^\alpha = \mathcal{I}^\gamma$ determining $\Sigma_{\mathcal{I}^\gamma}$. Later on, there are two EαKs in $\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha$, from which one is not considered by the incision function given that its related sets $\mathcal{T}_\mathcal{K}$ and $\mathrm{F}_\mathcal{K}$ are empty. Finally, since the other EαK considers terminologies (axioms), from Def. 6 we have that $\sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha) = \langle \{C \sqsubseteq D\}, \{\} \rangle$. Finally, the resultant KB would be $\Sigma' = \Sigma \ominus D(a) = \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp D(a)) = \langle \{\}, \{C(a), D(b)\} \rangle$.*

Axiomatic Characterization

As the basis for the axiomatization, we extend the basic postulates for bases given in (Hansson 1999).

(Inclusion) $\Sigma \ominus \alpha \subseteq \Sigma$.

(Success) [5] If $\not\models \alpha$ then $\Sigma \ominus \alpha \not\models \alpha$.

(Core Retainment) [6] If $\beta \in \Sigma$ and $\beta \notin \Sigma \ominus \alpha$ then there is some $H \subseteq \Sigma_{\mathcal{I}^\gamma}$ such that $H \not\models \alpha$ but $(H \cup \{\beta\}) \models \alpha$.

(Uniformity) For every $H \subseteq \Sigma_{\mathcal{I}^\gamma}$ it is verified that if $H \models \alpha$ *iff* $H \models \beta$ then $\Sigma \ominus \alpha = \Sigma \ominus \beta$.

[5] We use $\models \alpha$ to denote α as tautological.

[6] When β is an axiom, $(H \cup \langle \{\beta\}, \emptyset \rangle) \models \alpha$ follows; whereas when β is an assertion, it follows $(H \cup \langle \emptyset, \{\beta\} \rangle) \models \alpha$.

As is shown below, a model-based contraction operator defined so far does not guarantee success.

Example 9 (Ex. 7 cont.) *Suppose $\mathcal{I}^\gamma = \mathcal{I}_1^\alpha$, thus $\neg Re(c)$ could be chosen by the incision "σ". In such a case, the resultant KB Σ' would admit new models $\mathcal{I}'$ where $c \in Re^{\mathcal{I}'}$. Hence a new EαK will appear: $H \cup \langle \emptyset, \{Re(c)\} \rangle$, where $H = \langle \emptyset, \{SU(c,d), \neg Re(d)\} \rangle$, $H \subseteq \Sigma'$, but also $H \subseteq \Sigma$.*

A contraction operation not guaranteing success was proposed in (Fermé & Hansson 2001), but in the context of ontology evolution, we believe that success should be a must.

After a model-based contraction deletes some beliefs, it may fail to guarantee success if some subset $H \subseteq \Sigma$ along with the negation of some beliefs chosen by "σ", turns out being a new α-proof in the resultant KB. These kind of subsets are referred as *shielding sets* of information. Note that a shielding set is also a KB.

For a shielding set H, it follows $H \cup \langle \emptyset, \widehat{\mathcal{A}_\sigma} \rangle \models \alpha$, if $\widehat{\mathcal{A}_\sigma}$ is a *disagreement set* of the assertions chosen by "σ". Intuitively, a disagreement set negates some of those assertions and keeps the rest of them as they are.

Definition 8 (Disagreement set) *Given an ABox $\mathcal{A}$, the set $\delta(\mathcal{A})$ of disagreement sets $\widehat{\mathcal{A}}$ is:*

$$\delta(\mathcal{A}) = \{\widehat{\mathcal{A}} \neq \mathcal{A} | \forall \beta, \beta \in (\mathcal{A} \backslash \widehat{\mathcal{A}}) \text{ iff } \neg\beta \in (\widehat{\mathcal{A}} \backslash \mathcal{A})\}$$

Note that a disagreement set $\widehat{\mathcal{A}}$ may be also referred as a disagreement ABox. Now we can formally define the *shielding sets* by means of some disagreement set of $\mathcal{A}_\Sigma$.

Definition 9 (Shielding Set) *A set $H \subseteq \Sigma$ is a "shielding set" iff $H \cup \langle \emptyset, \widehat{\mathcal{A}} \rangle \models \alpha$, for some $\widehat{\mathcal{A}} \in \delta(\mathcal{A})$, $\mathcal{A} \subseteq \mathcal{A}_\Sigma$.*

From an incision, some of its disagreements may "activate" a shielding set from Σ. In such a situation, the resultant KB will keep the previous EαKs, and for the new triggered models, a new EαK will appear. Thus, by restricting incisions, we avoid any shielding set to be activated.

(Anti-Shielding) There is no $H \subseteq \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$ such that $H \cup \langle \emptyset, \widehat{\mathcal{A}_\sigma} \rangle \models \alpha$, for some $\widehat{\mathcal{A}_\sigma} \in \delta(\mathcal{A}_\sigma)$.[7]

This property anticipates the generation of a new model satisfying α in the resultant KB. If this happens, *anti-shielding* restricts the incision function in order to avoid the validity of that model. Hence, by considering a model-based contraction determined by a model incision function which satisfies *anti-shielding*, we guarantee success.

Definition 10 (Anti-Shielding Model-Based Contraction) *Let "$\ominus$" be a model-based contraction operator determined by a model incision function "σ". The function "σ" guaranties anti-shielding iff "$\ominus$" is an "anti-shielding model-based contraction operator".*

In the following example it is shown how a model incision function is restricted by anti-shielding.

Example 10 (Ex. 9 cont.) *Assume $\mathcal{A} = \{\neg Re(c)\}$, hence $\delta(\mathcal{A}) = \{\{Re(c)\}\}$. Thus, given $H \cup \langle \emptyset, \widehat{\mathcal{A}} \rangle \models \alpha$, where*

[7] Note that $\sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$.

$\widehat{\mathcal{A}} \in \delta(\mathcal{A})$, *if* $\mathcal{A}_\sigma = \mathcal{A}$, *where* $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$, *it is clear that "*$\sigma$*" does not guarantee anti-shielding. Finally,* $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \{\}, \{SU(b,c)\} \rangle$. *Hence, for the resultant KB* $\Sigma \ominus \alpha = \langle \emptyset, \{SU(a,b),\ SU(c,d),\ Re(a),\ \neg Re(c),\ \neg Re(d)\} \rangle$ *success is assured, i.e.,* $\Sigma \ominus \alpha \not\models \alpha$.

Proposition 2 *An operator "*$\ominus$*" is an anti-shielding model-based contraction iff it guaranties success.*

Proof: Given a KB $\Sigma = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \rangle$, and a sentence α, we will assume $\Sigma \models \alpha$, and $\not\models \alpha$. Moreover, a new contracted KB $\Sigma' = \Sigma \ominus \alpha$ is such that $\Sigma' = \langle \mathcal{T}_{\Sigma'}, \mathcal{A}_{\Sigma'} \rangle$, where $\mathcal{T}_{\Sigma'}$ and $\mathcal{A}_{\Sigma'}$ are the contracted TBox and ABox, respectively.

($\Rightarrow$) For the first part, if "$\ominus$" is an anti-shielding model-based contraction then it guaranties success, let us assume to the contrary that "$\ominus$" does not guarantee success, *i.e.,* $\Sigma' \models \alpha$. It follows that every model $\mathcal{I}' \in \mathcal{M}(\Sigma')$ satisfies α. Thus, in every KB extension $\Sigma'_{\mathcal{I}'^\alpha}$ there exists at least one EαK $\mathcal{K} = \langle \mathcal{T}_\mathcal{K}, \mathrm{F}_\mathcal{K} \cup \mathrm{S}_\mathcal{K} \rangle$ such that $H = \langle \mathcal{T}_\mathcal{K}, \mathrm{F}_\mathcal{K} \rangle$, where $\mathcal{T}_\mathcal{K} \subseteq \mathcal{T}_{\Sigma'}$ and $\mathrm{F}_\mathcal{K} \subseteq \mathcal{A}_{\Sigma'}$, and a subset of assumed beliefs $\mathrm{S}_\mathcal{K} \subseteq \mathrm{S}_{\mathcal{I}'^\alpha}$, where $\mathcal{I}'^\alpha \in \mathcal{M}(\Sigma')$. Let $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$ be the KB determined by model incision function, such that $\mathcal{T}_\sigma \subseteq \mathcal{T}_\Sigma$ and $\mathcal{A}_\sigma \subseteq \mathcal{A}_\Sigma$.

By Def. 7, $\mathcal{A}_\sigma \not\subseteq \mathcal{T}_{\Sigma'}$, then some $\Sigma'_{\mathcal{I}'^\alpha}$ may consider any disagreement $\widehat{\mathcal{A}_\sigma} \in \delta(\mathcal{A}_\sigma)$, as well as $\mathcal{A}_\sigma$ itself. Therefore, $\mathrm{S}_\mathcal{K} \subseteq \widehat{\mathcal{A}_\sigma}$ holds for any $\widehat{\mathcal{A}_\sigma} \in \delta(\mathcal{A}_\sigma)$. Hence, $\mathcal{K} \models \alpha$, and in particular $H \cup \langle \emptyset, \widehat{\mathcal{A}_\sigma} \rangle \models \alpha$ hold, contradicting the anti-shielding postulate.

($\Leftarrow$) For the opposite way, if "$\ominus$" does guarantee success then it is an anti-shielding model-based contraction. We know that every $\mathcal{I} \in \mathcal{M}(\Sigma)$ satisfies α, but since $\Sigma' \not\models \alpha$, there is some $\mathcal{I}' \in \mathcal{M}(\Sigma')$ which does not satisfy α. Thus, it is plausible to assume that the interpretation $\mathcal{I}'$ ends up being a model for Σ' as a result of the contraction operation, *i.e.,* $\Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$ where $\mathcal{I}^\gamma \in \mathcal{M}(\Sigma)$, and $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$. Thereafter, for every EαK $\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$ we have two standpoints from Def. 6: a)$\mathcal{T}_\mathcal{K} \subseteq \mathcal{T}_\sigma$, and b) if $\mathcal{T}_\mathcal{K} = \emptyset$ and $\mathrm{F}_\mathcal{K} \neq \emptyset$ then $\mathrm{F}_\mathcal{K} \cap \mathcal{A}_\sigma \neq \emptyset$.

Since anti-shielding affects only assertional knowledge in $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$, the former case is verified trivially. From case b), we have $\mathcal{A}_\sigma \subseteq \Sigma$ and $\mathcal{A}_\sigma \not\subseteq \Sigma'$. This means that $\mathcal{A}_\sigma$ is satisfied by every model $\mathcal{I} \in \mathcal{M}(\Sigma)$, but there are some $\mathcal{I}' \in \mathcal{M}(\Sigma')$ that do not satisfy it. Therefore, for each $\beta \in \mathcal{A}_\sigma$ there is some $\mathcal{I}'$ satisfying $\neg\beta$. Moreover, there is at least one model $\mathcal{I}'$ for each disagreement $\widehat{\mathcal{A}_\sigma} \in \delta(\mathcal{A}_\sigma)$. Finally, each $\mathcal{I}'$ determines a KB extension $\Sigma'_{\mathcal{I}'}$ that do not contain any EαK for α. This means that $\Sigma'_{\mathcal{I}'} \not\models \alpha$ and since $\widehat{\mathcal{A}_\sigma} \subseteq \mathrm{S}_{\mathcal{I}'} \subseteq \Sigma'_{\mathcal{I}'}$, it is clear that there is no $H \subseteq \Sigma'$, *i.e.,* $H \subseteq \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$ such that $H \cup \langle \emptyset, \widehat{\mathcal{A}_\sigma} \rangle \models \alpha$, as stated by anti-shielding. □

Theorem 1 (Anti-Shielding Model-Based Contraction) *An operator "*$\ominus$*" is an anti-shielding model-based contraction operator iff it satisfies success, inclusion, core retainment, and uniformity.*

Proof: **Construction-to-postulates:** Let "$\ominus$" be an anti-shielding model-based contraction for $\Sigma = \langle \mathcal{T}_\Sigma, \mathcal{A}_\Sigma \rangle$. We will show that it satisfies the four conditions of the theorem.

Success follows from the first part of Prop. 2, and inclusion, trivially from Def. 7. For core-retainment, suppose that $\beta \in \Sigma$ and $\beta \notin \Sigma \ominus \alpha$. Then it holds that $\beta \in \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$, given that $\Sigma \ominus \alpha = \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$ by Def. 7. Therefore, by Def. 6, $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) \subseteq (\bigcup_{\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)} \mathcal{K}) \cap \Sigma$, where $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$. Thus, there is some E$\alpha$K $\mathcal{K} \subseteq \Sigma_{\mathcal{I}^\gamma}$ such that $\beta \in \mathcal{K}$. It follows that $\beta \in \mathcal{T}_\Sigma$ or $\beta \in \mathcal{A}_\Sigma$. For the former, when $\beta \in \mathcal{T}_\Sigma$, we know $\beta \in \mathcal{T}_\mathcal{K}$ and following condition $(2a)$ in Def. 6, $\mathcal{T}_\mathcal{K} \subseteq \mathcal{T}_\sigma$ holds for every $\mathcal{K}$. For the latter, when $\beta \in \mathcal{A}_\Sigma$, we know $\beta \in \mathrm{F}_\mathcal{K}$ and following condition $(2b)$ in Def. 6, we know that $\mathcal{T}_\mathcal{K} = \emptyset$, $\mathrm{F}_\mathcal{K} \neq \emptyset$, and $\beta \in (\mathrm{F}_\mathcal{K} \cap \mathcal{A}_\sigma)$. Thus, in any case, it is proved the existence of some EαK $\mathcal{K} \subseteq \Sigma_{\mathcal{I}^\gamma}$ such that $\beta \in \mathcal{K}$. Let now consider $H = \mathcal{K} \backslash \{\beta\}$. It is clear that $H \subseteq \Sigma_{\mathcal{I}^\gamma}$, then $H \not\models \alpha$ but $H \cup \{\beta\} \models \alpha$ (from condition (3) in Def. 2) shows core-retainment is satisfied.

For uniformity, suppose that it holds for all subsets $B \subseteq \Sigma_{\mathcal{I}^\gamma}$ that $B \models \alpha$ if and only if $B \models \beta$. By Prop. 3, $\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha$=$\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta$. It follows from this that $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta)$, and by the definition of "$\ominus$" that $\Sigma \ominus \alpha = \Sigma \ominus \beta$, so that uniformity is satisfied.

Postulates-to-construction: Let "$\ominus$" and Σ be such that the four conditions of the theorem are satisfied. We are going to show that "$\ominus$" is an anti-shielding model-based contraction. For that purpose, let σ be such that for all α:

$$\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \Sigma \backslash (\Sigma \ominus \alpha).$$

This follows from inclusion ($\Sigma \ominus \alpha \subseteq \Sigma$) and $\Sigma \ominus \alpha = \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$ as posed by Def. 7. Thereafter, we need to verify that σ is an anti-shielding model incision function for Σ. To be that, both "γ" and "σ" must be functions, and "σ" also satisfy (1), and (2) from Def. 6, and anti-shielding.

Proof that "γ" and "σ" are functions: Let α and β be two sentences such that $\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha = \Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta$. We need to show that $\gamma^\alpha = \gamma^\beta = \mathcal{I}^\gamma$ and $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta)$. It follows from $\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha = \Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta$, by Prop. 3, that every subset $B \subseteq \Sigma_{\mathcal{I}^\gamma}$ implies α if and only if it implies β. Thus, by uniformity, $\Sigma \ominus \alpha = \Sigma \ominus \beta$. Hence, from the definition of "γ", "σ", and "$\ominus$" it follows that $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\beta)$.

Proof that (1) is satisfied: We will show that $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) \subseteq (\bigcup_{\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)} \mathcal{K}) \cap \Sigma$. Let $\beta \in \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$, it follows from core-retainment (given its preconditions $\beta \in \Sigma$ and $\beta \notin \Sigma \ominus \alpha$) that there is some $H \subseteq \Sigma_{\mathcal{I}^\gamma}$ such that $H \not\models \alpha$ but $H \cup \{\beta\} \models \alpha$. Hence, it follows that there is some EαK $\mathcal{K}$ such that $\beta \in \mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$. Thus, $\beta \in \bigcup_{\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)} \mathcal{K}$ and since $\beta \in \Sigma$, we conclude $\beta \in ((\bigcup_{\mathcal{K} \in (\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)} \mathcal{K}) \cap \Sigma)$.

Proof that (2) is satisfied: Suppose that $\langle \emptyset, \emptyset \rangle \neq \mathcal{K} \in \Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha$. It follows from this that $\not\models \alpha$. By success, $\Sigma \ominus \alpha \not\models \alpha$. Since $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) \subseteq \Sigma$ (where $\sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha) = \langle \mathcal{T}_\sigma, \mathcal{A}_\sigma \rangle$) and $\mathcal{K} \models \alpha$ we may conclude that $(\mathcal{T}_\mathcal{K} \cup \mathrm{F}_\mathcal{K}) \not\subseteq \Sigma \ominus \alpha$, *i.e.*, that there is some $\beta \in (\mathcal{T}_\mathcal{K} \cup \mathrm{F}_\mathcal{K})$ such that $\beta \notin \Sigma \ominus \alpha$. This leave us two options: $\beta \in \mathcal{T}_\sigma$ or $\beta \in \mathcal{A}_\sigma$. Besides, since $(\mathcal{T}_\mathcal{K} \cup \mathrm{F}_\mathcal{K}) \subseteq \Sigma$ it follows $\beta \in (\Sigma \backslash \Sigma \ominus \alpha)$, *i.e.,* $\beta \in \sigma(\Sigma_{\mathcal{I}^\gamma}{}^{\perp\!\!\!\perp}\alpha)$. Therefore, we have also that $\beta \in \mathcal{T}_\mathcal{K}$ or $\beta \in \mathrm{F}_\mathcal{K}$. Hence, if $\beta \in \mathcal{T}_\mathcal{K}$, this is enough to prove a). To the contrary, if $\beta \in \mathrm{F}_\mathcal{K}$, we have that $\mathcal{T}_\mathcal{K} = \emptyset$, $\mathrm{F}_\mathcal{K} \neq \emptyset$, and $\beta \in (\mathrm{F}_\mathcal{K} \cap \mathcal{A}_\sigma)$. Finally $(\mathrm{F}_\mathcal{K} \cap \mathcal{A}_\sigma) \neq \emptyset$.

Proof that "σ" guaranties anti-shielding: Given success, it follows directly from the second part of Prop. 2. □

The following equivalence is similar to that introduced in (Hansson 1999). Its proof is absence due to space reasons.

Proposition 3 *The following conditions are equivalent:*

(1) $\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha = \Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \beta$

(2) *For all subsets* $H \subseteq \Sigma_{\mathcal{I}^\gamma}$*:* $H \models \alpha$ *iff* $H \models \beta$.

The following theorems are related to the principles proposed in (Dalal 1988). Theorem 2 assures that any resultant KB from a model-based contraction operation will conform the DL used for the previous KB, whereas Theorem 3 abstract away from the DL used, stating that any pair of logically equivalent knowledge will be equally treated.

Theorem 2 (Adequacy of Representation) *Let $\mathcal{L}$ be an $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}]$ DL. For any KB $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$ and any sentence $\alpha \in \mathcal{L}$, it follows $(\Sigma \ominus \alpha) \subseteq \mathcal{L} \times \mathcal{L}$.*

Theorem 3 (Irrelevance of Syntax) *Let $\mathcal{L}, \mathcal{L}'$ be two $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}]$ DLs, $\Sigma \subseteq \mathcal{L}_\mathcal{T} \times \mathcal{L}_\mathcal{A}$ and $\Sigma' \subseteq {\mathcal{L}_\mathcal{T}}' \times {\mathcal{L}_\mathcal{A}}'$, two KBs, and $\alpha \in \mathcal{L}$ and $\alpha' \in \mathcal{L}'$, two such DL sentences. If $\alpha \approx \alpha'$ (where $\approx$ means logically equivalent to) and for every $\beta \in \Sigma$ and every $\beta' \in \Sigma'$, $\beta \approx \beta'$, then for every $\beta_R \in (\Sigma \ominus \alpha)$ and every $\beta'_R \in (\Sigma' \ominus \alpha')$, $\beta_R \approx \beta'_R$.*

Algorithm Specification

Since our theory relies on the proper selection of an α-minimal model $\mathcal{I}^\gamma \in \mathcal{M}(\Sigma)$, it is important to propose an algorithm implementing the selection function. An interesting approach would be to take advantage of the tableau algorithm used by the DL's subjacent inference engine, in such a way that $\mathcal{I}^\gamma$ turns out being its outcome at the time the satisfiability of α is being checked. This would provide an important shortcut in favor of the computability, such that finding the proper model $\mathcal{I}^\gamma$ would be attached to the time of computing the satisfiability checking of the sentence α.

Notice that the (canonical) model identified by a classical tableau procedure has the form of a model $\mathcal{I}^\alpha$, thus we will trivially assume the devised model as the one selected by the model selection function "γ", such that $\mathcal{I}^\alpha = \mathcal{I}^\gamma$. Hence, its related α-minimal KB extension $\Sigma_{\mathcal{I}^\gamma}$ may be generated and therefore it could be checked which of the EαKs obtained by the tableau process is included in $\Sigma_{\mathcal{I}^\gamma}$, thus generating the set of EαKs $\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha$. Hence, a model incision function "σ" may be applied obtaining $\sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$ and finally the contraction $\Sigma \ominus \alpha$ is resolved.

A special mention should be done regarding the anti-shielding property. As seen before, the anti-shielding validation is directly related to the assertional knowledge the incision function chooses to eliminate, and the sentence α to be contracted . The main problem appears when the incised assertions $\mathcal{A}_\sigma$, and its disagreement sets $\widehat{\mathcal{A}_\sigma}$, may conform new models satisfying α. That is, if the sentence α to be contracted considers a concept C and also some role R whose range is in its complement $\neg C$ (see Ex. 2), this may provoke the disagreement sets to be part of the new α-minimal extensions of the resultant KB.

To solve this situation, a simple heuristic may consider to avoid any incision of a concept if its complement is also considered in the satisfiability checking of the tableau machinery, disregarding both are relating different individual names. This is specified below.

Anti-Shielding Rule

Condition: $\mathcal{A}$ contains $A(x)$, and $\forall R.\neg A(y)$ or $\exists R.\neg A(y)$, for any individual names x, y such that $x \neq y$.

Action: $\mathcal{A}' \leftarrow \mathcal{A} \backslash \{A(x), \neg A(y)\}$.

Notice that the anti-shielding rule will be used after obtaining a closed constraint system, and it will always leave a non-empty ABox $\mathcal{A}'$ since such a rule could only be applied in the presence of some role $R(x, y)$. Therefore, the model incision function would have at least $R(x, y)$ to choose.

Algorithm 1 Calculate $\Sigma' = \Sigma \ominus \alpha$.

Input: Σ, α.
Output: Σ'.
  $\overrightarrow{S} \leftarrow tableauProc(\Sigma, \alpha)$.
  if $\overrightarrow{S}$ is closed **then**
    $\mathcal{I}^\alpha \leftarrow canonicalModel(\Sigma, \overrightarrow{S})$.
    $\sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha) \leftarrow antiShIncision(\Sigma, \mathcal{I}^\alpha, \overrightarrow{S})$.
    $\Sigma' \leftarrow \Sigma \backslash \sigma(\Sigma_{\mathcal{I}^\gamma} \perp\!\!\!\perp \alpha)$.
  else
    $\Sigma' \leftarrow \Sigma$.
  end if

The $antiShIncision$ procedure identifies each EαK from the closed constraint system conforming the model $\mathcal{I}^\alpha$ and the KB Σ. This is done by recognizing each EαK at a time, in order to maintain the same space requirement of the related tableau procedure. After one EαK is identified, it is viewed as an instantiation from the closed constraint system, then the Anti-Shielding Rule above is applied to the EαK restricting the domain of the incision wrt. assertions. Consequently, the model incision is applied to the remainder of the EαK at issue, in accordance to Def. 6. Afterwards, the closed constraint system (which remains intact) will determine the next recognition of a new EαK.

Theorem 4 (Model-Based Contractions Complexity) *Let $\mathcal{L}$ be an $\mathcal{ALC}$ DL, $\Sigma \subseteq \mathcal{L}$, a KB, $\alpha \in \mathcal{L}$, a sentence, and "$\ominus$", an anti-shielding model-based contraction operator. The complexity of $\Sigma \ominus \alpha$ is PSPACE-complete.*

Proof sketch: $\mathcal{ALC}$ DLs have been proved to be PSPACE-complete for satisfiability of concepts descriptions, following the related tableau algorithm. Thus, since Algorithm 1 calls once to the tableau procedure, we should analyze that the rest of the algorithm could be computed in polynomial space as being required by the procedure $tableauProc$.

The most problematic procedure in our algorithm may be $antiShIncision$. Conforming the given canonical model and the resultant constraint system, $antiShIncision$ chooses the related knowledge to each EαK by calculating, and considering them, each at a time (this is done with no need to calculate the related KB extension). Afterwards, since each EαK fits the space of the constraint system considered, it follows

that $antiShIncision$ is executed in the polynomial space required before. Finally, Alg.1 for $\mathcal{ALC}$ DLs is proved to be PSPACE-complete. □

Related Work

Ontology revision is currently an interesting topic in which belief revision meets description logics. In the last few years, several articles in this area has been published. For instance, in (Flouris *et al.* 2006), incoherence and inconsistency of ontologies are formally presented. Based on the distinction between coherent and consistent negation, a set of postulates for revising DLs is proposed, although no operator is specified. Recently in (Qi *et al.* 2008), a kernel revision operator for terminologies was presented, there an incision function is specified to delete axioms avoiding a terminology to evolve incoherently. (Ribeiro & Wassermann 2007) presents a similar approach in which an incision is performed over terminologies dealing with inconsistency. Similarly, in (Haase *et al.* 2005), based on a selection function different sub-ontologies are identified to consistently incorporate a given new axiom.

In general, most of the ontology change operators proposed so far are based on the notions of MIPS and MUPS. Such constructions were originally presented in (Schlobach & Cornet 2003) as a debug tool for pinpointing terminological errors to correct inconsistencies. In contrast, in (Meyer, Lee, & Booth 2005) the KB is weakened and inconsistencies are also tolerated.

In our approach, we manage the evolution in ontologies by considering not only axioms but also assertional knowledge. Moreover, a model-based contraction addresses in advance inconsistencies and incoherencies, that is, statements eliminated from the ontology avoid as much as possible instance data loss (following minimal change). Afterwards, the intention of the complete change operator is to accommodate the deleted axioms to avoid incoherency, and further reincorporate them. This latter sub-task of the change operator is part of our ongoing work in the matter.

Similar to the construction of α-kernels, MUPS are defined as minimal inconsistent sets for a given atomic concept. This structure is constructed with the aim of restoring satisfiability to an unsatisfiable concept definition. In our approach, we anticipate the change and prepare the ontology to accept the new information consistently, leaving the accommodation of outdated axioms to the completion of the ontology change operation. This is the purpose of model-based contractions, defined as a functional part of the general change operation that makes effective an ontological change.

As stated before, model-based contractions are motivated by kernel contractions in their intuitions of breaking minimal proofs for a given sentence, but despite both constructions delete beliefs from minimal proofs, they are semantically different. In kernel contractions, deletions break or cut proofs. However, in model-based contractions deletions imply a variation of the set of models and therefore the generation of new associated KB extensions, while the original proofs remain.

In (Dalal 1988) a KB revision operation was semantically specified at the knowledge level by considering model-theoretic semantics. Although, as stated by the author, no consideration about differential treatment of certain atoms, and even formulae, is taken into account, so that some beliefs could be more easily given up.

The six AGM basic postulates for contractions (Alchourrón, Gärdenfors, & Makinson 1985) were supposed to capture the intuition behind any contraction operation on a belief set (closed under logical consequence). But recently in (Flouris February 2006), it was shown that some Tarskian logics –non-AGM Compliant logics– do not admit a contraction operation satisfying the six AGM postulates. However, they admit contractions without recovery.

Regarding DLs, we are particularly interested in very expressive logics like $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$, which are shown to be equivalent to OWL-Lite and OWL-DL (Horrocks & Patel-Schneider 2003), the two OWL sub-languages for which complete reasoners are known. Such DLs are known to be non-AGM Compliant (Flouris February 2006), but since this requires to guarantee recovery, the alternative to find some replacement postulate appears sensible.

In particular, recovery has been the most problematic postulate, standing unnaturally for the principle of minimal change. For instance, in (Hansson 1999) regarding bases, core-retainment have been proposed as a substitute for recovery, while uniformity stands for extensionality.

Discussion

In contrast to item $2a)$ in Def. 6, it seems enough for it to be defined as: "If $\mathcal{T}_{\mathcal{K}} \neq \emptyset$ then $\mathcal{T}_{\mathcal{K}} \cap \mathcal{T}_{\sigma} \neq \emptyset$, and... ".

Although that would (apparently) follow minimal change, such decision would be a detriment to minimal change wrt. the complete ontology change operator, as will be seen below. In consequence, we decided to take complete terminologies as stated by the following intuition: when an axiom ends up being unsatisfiable, a contradiction appears while checking its unfolded version. Since that terminology is part of a minimal α-proof, each of the axioms considered –along with the sentence to be incorporated– are interrelated, as inferential steps to infer a minimal incoherence.

Axiom unfolding may provoke an explosion in the size of the search space leading to a notable degradation in performance. For that reason, a careful analysis is required to provide an efficient methodology. In (Tsarkov, Horrocks, & Patel-Schneider 2007), a complete overview of lazy unfolding and other reasoning optimizations are described.

By analyzing a unique unfolded axiom it let us to identify the exact point in which the axiom turns to unsatisfiability. That is, the terminology from each EαK along with the sentence φ is a minimal incoherence preserving subterminology (MIPS). After that, technics from ontology debugging could be applied to restore coherence to the subterminology. Finally, it is incorporated to the KB.

For instance, in Ex. 1, we have axiom $D_3 \sqsubseteq C \sqcap \neg D_2$ in an EαK. Considering the sentence $\varphi = C \sqsubseteq D_2$, note that $\{D_3 \sqsubseteq C \sqcap \neg D_2, C \sqsubseteq D_2\}$ is a MIPS. It is clear that the unfolded axiom $D_3 \sqsubseteq D_2 \sqcap \neg D_2$ is unsatisfiable. Later

on, coherency may be restored by assuming the unfolded axiom as $D_3 \sqsubseteq D_2$, which means that the repaired sub-terminology ends up being $\{D_3 \sqsubseteq C, C \sqsubseteq D_2\}$. Finally, the repaired terminology may be reincorporated to the evolved KB, which would end up consistent and coherent.

Note that, this (apparently) drawback on minimal change wrt. the contraction does not contradict *core-retainment*. The matter discussed above is part of the ongoing work, and is proposed as future work for the completion of the ontology change operation.

Conclusions and Future Work

In this paper we have proposed a new contraction operator of model-theoretic semantics, dedicated to avoid the inference of a sentence α in a specific ontology, both expressed in some description language as $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}]$.

The process modeled by model-based contractions may be summed up as follows: from an ontology $\mathcal{O}$ and some finite model $\mathcal{I}$, we extend the ontology to $\mathcal{O}_{\mathcal{I}}$. In it, different sub-ontologies, namely EαKs $\mathcal{K}_1, \mathcal{K}_2, \ldots$ are identified. Analyzing the information in such sub-ontologies, an incision function "σ" determines a sub-ontology $\mathcal{O}_\sigma \subseteq \mathcal{O}$, such that $\mathcal{O} \models \alpha$, but $(\mathcal{O} \backslash \mathcal{O}_\sigma) \not\models \alpha$.

Our intention regarding the formal theory here provided, is to apply it on other more expressive DLs like $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$. This work is introductory in that sense. As part of its preliminary results, an algorithm was provided as a possible option towards its further realization. Such algorithm has been defined on top of the tableau procedure used by the related DL reasoner to find the appropriate model for the theory to be applied. Moreover, since the theory mostly relies on such a selection, the complexity results in terms of the space required to compute, depends on the tableaux algorithm used to reason.

In this sense, the model selection (and its related preference criterion "$\prec$") has been abstracted away from the algorithm, and assumed to be the canonical model recognized from the tableau procedure. For this matter, it becomes interesting to investigate the model preference criterion "$\prec$" to be specified. By determining such a criterion, the tableau procedure might be directed by specifying an order of the transformation rules to apply, and different properties to determine which constraint systems should be attended first.

Since ontologies are highly reusable distributed, generation of intermediate inconsistent states may be critical for any change operation. This is a considerable advantage that model-based contractions provide, since they "foretell" any undesired effect from the change operation, repairing it in advance. In this sense, adjustment of outdated axioms is part of our future work in the field of ontology debugging.

This proposal would be incomplete, without considering model-based contractions as a sub-operation of a broader ontology change operator. Indeed, some assumptions made relying on that consideration, are sensible only for that matter. Future work also involves the formalization of the ontology change operation, along with a set of general principles for the evolution, to state a set of postulates by which the new change operator may be axiomatically characterized.

Acknowledgments

We would like to thank to Nicolás D. Rotstein, Juergen Dix, Gabriele Kern-Isberner, and Pablo R. Fillottrani for the invaluable contributions provided. This work is partially financed by CONICET (PIP 5050), Universidad Nacional del Sur (PGI 24/ZN11) and Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro 13096).

References

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. *On the Logic of Theory Change: Partial Meet Contraction and Revision Functions. The Journal of Symbolic Logic* 50:510–530.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P., eds. 2003. *Description Logic Handbook: Theory, Implementation and Application*. Cambridge: Cambridge University Press.

Dalal, M. 1988. Investigations into a Theory of Knowledge Base Revision. In *AAAI*, 475–479.

Fermé, E., and Hansson, S. O. 2001. Shielded Contraction. *In M-A Williams and H. Rott eds. Frontiers in Belief Revision. Applied Logic Series 22* 85–107.

Flouris, G.; Huang, Z.; Pan, J. Z.; Plexousakis, D.; and Wache, H. 2006. Inconsistencies, Negations and Changes in Ontologies. In *AAAI*, 1295–1300.

Flouris, G. February 2006. On Belief Change and Ontology Evolution. *Doctoral Dissertation, Department of Computer Science, University of Crete*.

Haase, P.; van Harmelen, F.; Huang, Z.; Stuckenschmidt, H.; and Sure, Y. 2005. A Framework for Handling Inconsistency in Changing Ontologies. In *ISWC*, 353–367.

Hansson, S. O. 1994. Kernel Contraction. *Journal of Symbolic Logic* 59:845–859.

Hansson, S. O. 1999. *A Textbook of Belief Dynamics: Theory Change and Database Updating. Springer*.

Horrocks, I., and Patel-Schneider, P. 2003. Reducing OWL Entailment to Description Logic Satisfiability. *ISWC, 2003* 2870:17–29.

Kalyanpur, A.; Parsia, B.; Sirin, E.; and Grau, B. C. 2006. Repairing Unsatisfiable Concepts in OWL Ontologies. In *ESWC*, 170–184.

Meyer, T.; Lee, K.; and Booth, R. 2005. Knowledge Integration for Description Logics. In *AAAI*, 645–650.

Qi, G.; Haase, P.; Huang, Z.; and Pan, J. Z. 2008. A Kernel Revision Operator for Terminologies. In *DL*.

Ribeiro, M. M., and Wassermann, R. 2007. Base Revision in Description Logics - prelimimnary results. In *IWOD*.

Schlobach, S., and Cornet, R. 2003. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *IJCAI*, 355–362.

Tsarkov, D.; Horrocks, I.; and Patel-Schneider, P. F. 2007. Optimizing Terminological Reasoning for Expressive Description Logics. *J. Autom. Reason.* 39(3):277–316.

Degrees of Recovery and Inclusion in Belief Base Dynamics

Márcio Moretto Ribeiro and **Renata Wassermann**
Department of Computer Science
University of São Paulo, Brasil
{marciomr,renata}@ime.usp.br

Abstract

When contracting a formula from a belief base, two desiderata compete: one wants to avoid including any new belief in the process (inclusion) but may want to be able to recover information that was in the base before the contraction took place (recovery).

The AGM paradigm imposes both constraints on contraction operations. However, for finite belief bases inclusion and recovery cannot be simultaneously satisfied.

In this paper, we examine constructions that weaken the inclusion constraint and retain some form of recovery. We show that depending on what is allowed to be added, we obtain a counterpart to the principle of minimal change, where we add just enough information to allow recovery.

Introduction

Belief Revision (Gärdenfors 1988; Gärdenfors and Rott 1995; Hansson 1999) deals with the problem of accommodating new information into a body of existing beliefs. The new piece of information may be inconsistent with the previous information held by the agent. In this case, he may have to give up some previous beliefs.

The problem has been extensively studied in the literature and most formal proposals derive from what is known as the *AGM paradigm*, due to the initials of the authors of (Alchourrón, Gärdenfors, and Makinson 1985). In the AGM paradigm, three operations of belief change are distinguished: expansion, which is the simple addition of a new belief; contraction, which consists in removing the desired belief; and revision, which consists in adding a new belief in such a way that the resulting set is consistent.

In the AGM paradigm, the beliefs of an agent are represented by a set of formulas closed under logical consequence, a *belief set*. The operation of expansion is obtained by adding the new belief and closing the resulting set under logical consequence. The operations of contraction and revision are not uniquely defined, but restricted by a set of desired axioms (the *rationality postulates*). Several mathematical constructions were proposed that have the property of being equivalent to the set of postulates, in the sense that not only an operation following these constructions satisfies the postulates, but also any operation satisfying the postulates can be obtained from these constructions (Gärdenfors 1988). All of these constructions satisfy the Levi identity, that shows how revision can be obtained from contraction and expansion. Therefore, in this paper we will concentrate on the contraction operation.

The six basic AGM postulates for contraction are listed below:

(K-1) $K - \varphi$ is a belief set (*closure*)
(K-2) $K - \varphi \subseteq K$ (*inclusion*)
(K-3) If $\varphi \notin K$, then $K - \varphi = K$ (*vacuity*)
(K-4) If not $\vdash \varphi$, then $\varphi \notin K - \varphi$ (*success*)
(K-5) $K \subseteq (K - \varphi) + \varphi$ (*recovery*)
(K-6) If $\vdash \varphi \leftrightarrow \psi$, then $K - \varphi = K - \psi$ (*extensionality*)

These postulates are supposed to capture the intuition behind the operation of giving up a belief in a rational way. Postulate **(K-1)** says that the result of contracting a belief set by a formula should again be a belief set. The next postulate assures that in an operation of contraction no new formulas are added to the initial belief set. If the formula to be contracted is not an element of the initial belief set, then by **(K-3)** nothing changes. Postulate **(K-4)** says that unless the sentence to be contracted is logically valid (and hence, an element of every theory), it is not an element of the resulting belief set. The recovery postulate **(K-5)** is the most controversial one (Makinson 1987). It says that a contraction should be recoverable, that is, that the original belief set should be recovered by expanding by the formula that was contracted. The last postulate assures that contraction by logically equivalent sentences produces the same output.

The postulate of *recovery* has been debated since the very beginning of the AGM paradigm. We want to avoid contraction operations that simply discard all the beliefs. Intuitively, when we contract by a formula α, we want to discard a minimal subset of the belief set such that the resulting set does not contain α. This is known as the Principle of Minimal Change. Recovery is one way to try to capture this minimality, but some examples show that the postulate may be too strong:

Example 1 (Hansson 1999): "I have read in a book about Cleopatra that she had both a son and a daughter. My

set of beliefs therefore contains both p and q, where p denotes that Cleopatra had a son and q that she had a daughter. I then learn from a knowledgeable friend that the book is in fact a historical novel. After that I contract $p \vee q$ from my set of beliefs, i.e., I do not any longer believe that Cleopatra had a child. Soon after that, however, I learn from a reliable source that Cleopatra had a child. It seems perfectly reasonable for me to then add $p \vee q$ to my set of beliefs without also reintroducing either p or q. This contradicts Recovery."

Operations that satisfy **(K-1)**-**(K-4)** and **(K-6)**, i.e., that do not satisfy recovery, were called *withdrawals* in (Makinson 1987). There are several constructions in the literature that do not satisfy recovery (Makinson 1987; Fermé 2001; Rott and Pagnucco 1999), but for belief sets they all present some other undesirable property.

The postulates of *success* and *inclusion* are sometimes seen as the minimal requirements for a contraction operation. Together they state that the desired belief is removed and nothing else new is included in the belief set. (Booth et al. 2005) calls *retraction* an operation that satisfies **(K-1)**, **(K-3)**, **(K-4)** and **(K-6)** plus $K - \bot = K$ (*failure*). A retraction does not satisfy recovery or inclusion. The idea is that the contraction of a formula can "liberate" other beliefs that were abandoned because of that formula.

An alternative representation to belief sets is the use of belief bases, i.e., sets of formulas that are not necessarily closed under logical consequence. Besides being more expressive (as we can always derive the corresponding belief set taking the closure of a belief base), belief bases have clear advantages from the computational point of view. For belief bases, there are constructions that seem reasonable and do not satisfy recovery, as we will show later. The use of belief bases instead of logically closed sets brings with it the possibility of distinguishing between different syntactical representations of the same information. The belief bases $B_1 = \{p, q\}$, $B_2 = \{p \wedge q\}$, and $B_3 = \{p, p \rightarrow q\}$ are all different, although they imply exactly the same formulas according to classical logic. We can look at this possibility as an advantage in terms of expressivity. If we do not want to distinguish between the three cases, we only have to look to the formulas in the logical closure of the bases.

In this paper we will discuss the role and adequacy of the inclusion postulate and its relation to recovery. If we move from closed belief sets to belief bases, the postulate may be too restrictive, as the following examples show:

Example 2: Suppose we have a belief base containing $p \wedge q$, which stands for the fact that Cleopatra had a son (p) and a that she had a daughter (q). If we want to contract the base by p, i.e., we want to remove the belief that she had a son, we have to give up the whole conjunction, and since the formula q is not included in the base, we give up the belief that Cleopatra had a daughter too.

Example 3: Suppose I believe that penguins are birds ($p \rightarrow b$) and that birds fly ($b \rightarrow f$). Now I want to contract my belief that penguins fly ($p \rightarrow f$). We may want to end with the belief that all birds except for penguins fly ($(b \wedge \neg p) \rightarrow f$). This is also not allowed if the inclusion postulate holds.

The third example is a typical case of what (Maranhao 2001) calls Refinement, where there is a rule that needs to be weakened in order to accommodate exceptions.

We would like to have a construction that allows us to keep parts of formulas being removed. In this paper, we will present some ideas on such constructions. We will present two constructions for contraction without inclusion: one which is based on the idea of first expanding the base and then applying partial meet contraction (Alchourrón, Gärdenfors, and Makinson 1985) to it and one where we first apply partial meet contraction to the base and then expand the result.

It is important to note that even if we are questioning the adequacy of inclusion, we are concerned about it with respect to belief bases. We are not willing to allow completely new information to be added during contraction (as is the case of retraction), but only to allow the addition of information that was previously derivable. If we look at the closure of the belief base, we are not adding, but just retaining information.

The rest of the paper is organized as follows: in the next section, we introduce partial meet contraction of belief bases and its properties. Then we give an example of contraction without inclusion and propose a more general construction. We show that this construction can be instantiated and give rise to operations with different properties. We also propose a second construction together with a new rationality postulate and show how to introduce the idea of degrees of recovery.

Throughout the paper we use lower case letters to denote atoms, Greek lower case for formulas and upper case letters to denote set of formulas. We consider Cn to be the classical consequence operator and $A \vdash \alpha$ if and only if $\alpha \in Cn(A)$.

Contraction of Belief Bases

In this section, we will present postulates and a construction for contraction of belief bases. We will then discuss the properties of the operation.

The first construction for contraction that was proposed and proved to be equivalent to the six AGM postulates was *Partial Meet Contraction* (Alchourrón, Gärdenfors, and Makinson 1985). The operation is based on the idea of selecting maximal subsets of a belief set that do not imply the formula to be contracted.

Given a belief set K and a formula α, the *remainder* of K and α, denoted by $K \bot \alpha$ is the set of maximal subsets of K that do not imply α.

Definition 1 *(Alchourrón and Makinson 1982) Let X be a set of formulas and α a formula. For any set Y, $Y \in X \bot \alpha$ if and only if:*

- $Y \subseteq X$
- $Y \nvdash \alpha$
- *For all Y' such that $Y \subset Y' \subseteq X$, $Y' \vdash \alpha$.*

The operation of contraction is based on selecting the best sets in $K \perp \alpha$ and taking their intersection. The choice is encoded into a selection function:

Definition 2 *(Alchourrón, Gärdenfors, and Makinson 1985) A* **selection function** *for K is a function γ such that:*

- *If $K \perp \alpha \neq \emptyset$, then $\emptyset \neq \gamma(K \perp \alpha) \subseteq K \perp \alpha$.*
- *Otherwise, $\gamma(K \perp \alpha) = \{K\}$.*

Definition 3 *(Alchourrón, Gärdenfors, and Makinson 1985) The* **partial meet contraction operator** *on K based on a selection function γ is the operator $-_\gamma$ such that for all sentences α:*
$K -_\gamma \alpha = \bigcap \gamma(K \perp \alpha)$.

The operation of partial meet contraction was proven to be completely axiomatized by the set of postulates **(K-1)**-**(K-6)**:

Theorem 4 *(Alchourrón, Gärdenfors, and Makinson 1985) An operation $-$ is a partial meet contraction if and only if it satisfies postulates* **(K-1)**-**(K-6)**.

The same construction can be used for belief bases. It is easy to see that for belief bases, $-_\gamma$ satisfies **(K-2)**, **(K-4)**, **(K-6)** and a stronger version of **(K-3)** that we call *logical vacuity*: If $\alpha \notin Cn(B)$, then $B - \alpha = B$. To see that it does not satisfy **(K-5)**, we can look at the Cleopatra example:

Example 1 Revisited: Let $B = \{p, q\}$. Then the remainder of B and $p \vee q$ is $B \perp (p \vee q) = \{\emptyset\}$. Hence, $\gamma(B \perp (p \vee q)) = \{\emptyset\}$ and the contraction is given by $B -_\gamma (p \vee q) = \emptyset$.

But adding $p \vee q$ again does not necessarily bring all the information back: $B \not\subseteq B -_\gamma (p \vee q) + (p \vee q) = \{p \vee q\}$.

Given a contraction operator on belief bases, we can define a contraction operator on belief sets generated from it by taking the closure of the result. If $-$ is a contraction on belief bases, then it generates an operator $-'$ such that $Cn(B) -' \alpha = Cn(B - \alpha)$. If $-_\gamma$ is a partial meet base contraction, the example shows that even the operation on belief sets generated from it does not satisfy recovery, i.e.,
$Cn(B) \not\subseteq Cn(B -_\gamma (p \vee q)) + (p \vee q) = Cn(\{p \vee q\})$.

Actually, Hansson (Hansson 1999) has shown that under very general conditions, any base-generated contraction operation fails to satisfy recovery.

Hansson has proposed an alternative axiomatization for partial meet base contraction and proven that it is equivalent to the construction:

Theorem 5 *(Hansson 1992) An operator $-$ is an operator of partial meet base contraction on B if and only if:*

- *If $\alpha \notin Cn(\emptyset)$, then $\alpha \notin Cn(B - \alpha)$ (success)*
- *$B - \alpha \subseteq B$ (inclusion)*
- *If $\beta \in B \setminus (B - \alpha)$, then there is some B' such that $B - \alpha \subseteq B' \subseteq B$, $\alpha \notin Cn(B')$ and $\alpha \in Cn(B' \cup \{\beta\})$ (relevance)*
- *If for all subsets B' of B, $\alpha \in Cn(B')$ if and only if $\beta \in Cn(B')$, then $B - \alpha = B - \beta$ (uniformity)*

The inclusion and success postulates are the same as **(K-2)** and **(K-4)**. The closure postulate (**(K-1)**) does not apply to belief bases. The last postulate (uniformity) is a stronger version of extensionality. Instead of recovery, Hansson suggested the relevance postulate in order to capture the idea of minimal change. In a contraction by α, the only formulas given up are those that somehow contribute to the derivation of α.

As we have seen, adapting the traditional AGM construction of partial meet contraction to belief bases results in an operation that satisfies inclusion and success, but not recovery. In the next section, we will see how weakening inclusion can bring back some sort of recovery.

Contraction without Inclusion

We have seen in the Introduction that even the inclusion postulate may be too strong when talking about belief bases. We would like sometimes to retain parts of the beliefs that are being given up, which means replacing them by some of their consequences. The second Cleopatra example (Example 2) shows that we may want to have $\{p \wedge q\} - p = \{q\}$, i.e., instead of giving up the conjunction, replace it by one of its conjuncts.

Giving up inclusion all together may bring too much freedom for the allowed constructions: we could end up adding just any formula that did not threaten success, such as having $\{p\} - p = \{q\}$. The idea is to weaken the postulate so as to allow the addition of some kinds of formulas. In this section and the next one, we will explore some possibilities. The constructions that we propose satisfy *logical inclusion*, a weaker version of inclusion proposed in (Hansson 1989):

(logical inclusion) $Cn(B - \alpha) \subseteq Cn(B)$

Note that if Cn is Tarskian, logical inclusion is equivalent to $B - \alpha \subseteq Cn(B)$. We will use any of the two forms in this paper. Hansson called an operation satisfying success and logical inclusion a *pseudo-contraction*.

Following the same line, we may think of a weaker version of recovery. Recall that recovery for belief bases would state that $B \subseteq (B - \alpha) \cup \{\alpha\}$. We can relax the postulate requiring only that the original base is contained in the closure of the result of first contracting and then expanding by the same formula:

(logical recovery) $B \subseteq Cn(B - \alpha + \alpha)$

It is easy to see that logical recovery is indeed weaker than recovery. Recovery implies logical recovery, since Cn is Tarskian. If we look again at Example 2, we can think of an operation that satisfies logical recovery but does not satisfy recovery.

Example 2 Revisited: Let $B = \{p \wedge q\}$, and suppose we want to give up the belief in p and retain the belief in q, i.e., we want that the result of removing p from B is $B - p = \{q\}$. If expansion is defined in the usual way, as the simple union of a base and a formula, than $B \not\subseteq (B - p) + p = \{p, q\}$, but $B \subseteq Cn(\{p, q\})$.

Nebel has proposed a construction for pseudo-contraction that satisfies logical recovery. The idea is to add to partial meet contraction some consequences of the formulas of the original belief base so that they allow for the recovery of the contracted sentences:

Definition 6 *(Nebel 1989) Let B be a finite belief base, α a formula and γ a selection function.*

$$B-\alpha = \begin{cases} B & \text{if } \alpha \in Cn(\emptyset) \\ \bigcap \gamma(B\perp\alpha) \cup \{\alpha \rightarrow \beta | \beta \in B\} & \text{otherwise} \end{cases}$$

It is easy to see that not all formulas of the form $\alpha \rightarrow \beta$ have to be added, we can restrict ourselves to those β such that $\beta \in B \setminus \bigcap \gamma(B\perp\alpha)$ and still retain logical recovery:

Proposition 7 *Let B be a finite belief base, α a formula and γ a selection function and define the contraction $B-\alpha$ as:*

$$B-\alpha = \begin{cases} B & \text{if } \alpha \in Cn(\emptyset) \\ \bigcap \gamma(B\perp\alpha) \cup \{\alpha \rightarrow \beta | \beta \in B \setminus \bigcap \gamma(B\perp\alpha)\} & \text{otherwise} \end{cases}$$

Then $-$ satisfies logical inclusion, logical vacuity, success, logical recovery and extensionality.

Proof: Logical inclusion, logical vacuity, logical recovery and extensionality follow directly from the construction. To see that success is satisfied, suppose that $\alpha \notin Cn(\emptyset)$ and $\alpha \in Cn(B-\alpha)$. Then there are $\beta_1, \beta_2, ..., \beta_n$ in $B \setminus \bigcap \gamma(B\perp\alpha)$ such that $\bigcap \gamma(B\perp\alpha) \cup \{\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, ..., \alpha \rightarrow \beta_n\} \vdash \alpha$. Using the deduction theorem and the fact that $(\alpha \rightarrow \beta) \rightarrow \alpha$ is equivalent to α, we have that $\bigcap \gamma(B\perp\alpha) \vdash \alpha$, which we know cannot be the case (since this is the traditional partial meet construction). □

In a sense, Nebel's construction seems to have been coined to satisfy recovery. There is no other intuition about why exactly the implications of the form $\{\alpha \rightarrow \beta | \beta \in B\}$ should be added.

If we go back to our second Cleopatra example (Example 2), Nebel's pseudo-contraction (or the more economic form of it) would make $\{p \wedge q\} - p = \{p \rightarrow p \wedge q\}$, which means that if we do not believe anymore that Cleopatra had a son, we do not know anything about her having a daughter.

We will propose here a slightly different construction. Suppose that we have a partial meet contraction for a belief base B, with an associated selection function γ. Instead of taking the result of the contraction and then adding some formulas, we will first expand the belief base and then perform a partial meet contraction.

Let Cn^* denote an operation that generates some consequences of a set of formulas, i.e., $Cn^*(A) \subseteq Cn(A)$. As an example, we could have $Cn^*(A) = \{\alpha \vee \beta | \alpha \in A\}$. Given a selection function γ for a set B, we can extend γ to a selection function for a superset of B. Such extension is usually not unique.

Definition 8 *Let γ be a selection function for B and let B^* contain B. An extension of γ to B^* is a selection function γ^* such that for every $Y \in \gamma^*(B^*\perp\alpha)$ there is an $X \in \gamma(B\perp\alpha)$ such that $X \subseteq Y$.*

Observation 9 *Let γ be a selection function for B and let B^* contain B. If γ^* is an extension of γ to B^*, then for any $X \in \gamma(B\perp\alpha)$ there is a $Y \in \gamma^*(B^*\perp\alpha)$ such that $X \subseteq Y$.*

We define the general partial meet pseudo-contraction by first expanding B by a set containing some of the consequences of the formulas that would be given up in a partial meet contraction, and then applying partial meet contraction on the expanded base. The set of formulas that would be given up is given by $\{\beta | \beta \in B \setminus \bigcap \gamma(B\perp\alpha)\}$ and the consequences that will be used are selected by Cn^*. We call B^* the result of expanding B with the selected consequences.

Definition 10 *Let B be a finite belief base, α a formula and γ a selection function for B. The general partial meet pseudo-contraction $B-\alpha$ is given by:*

$$B-\alpha = \begin{cases} B & \text{if } \alpha \in Cn(\emptyset) \\ \bigcap \gamma^*(B^*\perp\alpha) & \text{otherwise} \end{cases}$$

where $B^* = B \cup Cn^*(\{\beta | \beta \in B \setminus \bigcap \gamma(B\perp\alpha)\})$ and γ^* is an extension of γ.

Proposition 11 *The pseudo-contraction operation defined as above satisfies logical inclusion, logical vacuity, success, and extensionality.*

Proof: Directly from the definition and the observation that $\bigcap \gamma(B\perp\alpha)$ satisfies extensionality, thus equivalent formulas generate the same B^*. □

The proposition shows that general pseudo-contraction satisfies some form of inclusion, vacuity, success and extensionality, four of the five relevant AGM postulates (closure is not applicable to belief bases). Moreover, if we consider the operation on belief sets generated from general partial meet pseudo-contraction, it satisfies closure, inclusion, vacuity, success, and extensionality. Whether this construction satisfies logical recovery or not depends on the Cn^* used. It is easy to see that the restricted form of Nebel's pseudo-contraction can be obtained if we take $Cn^*(A) = \{\alpha \rightarrow \beta | \beta \in A\}$. In this particular case, we do have logical recovery. And the operation on belief sets generated from it satisfies recovery.

We can think of different definitions for Cn^* depending on the intuitions. For example, if we look once more to the Cleopatra example, we may want to allow the addition of only those consequences which are subformulas of the formulas removed. The pseudo-contraction operation using this definition of Cn^* does not satisfy logical recovery, as can be seen from the following example:

Example 4: Let $B = \{p, p \rightarrow q\}$. Then $B\perp q = \{\{p\}, \{p \rightarrow q\}\}$. Suppose that $\gamma(B\perp q) = \{\{p \rightarrow q\}\}$. Since the only consequence of $p \rightarrow q$ that is a subformula of it is the whole formula, we have $B^* = B$. And since γ^* extends γ, we must have $\gamma^*(B\perp q) = \{\{p \rightarrow q\}\}$. Hence $B - q = \{p \rightarrow q\}$ and $B \not\subseteq Cn(B - q + q)$.

The operation does not even satisfy the weaker postulate of relevance, as inclusion may not hold and thus we may not have a set B' such that $B-\alpha \subseteq B' \subseteq B$. It does however satisfy a still weaker version of relevance, introduced by Hansson in (Hannson 1991):

(core-retainment) If $\beta \in B \setminus (B-\alpha)$, then there is some B' such that $B' \subseteq B$, $\alpha \notin Cn(B')$ and $\alpha \in Cn(B' \cup \{\beta\})$

Core-retainment, as relevance, states that if a belief is given up in a contraction, then it was relevant for the derivation of the formula contracted. But unlike relevance, it does not require inclusion. In fact, core-retainment is satisfied for any general partial meet pseudo-contraction, regardless of the particular Cn^* used (we do not even need to have $Cn^*(A) \subseteq Cn(A)$):

Proposition 12 *For any operator Cn^*, general partial meet pseudo-contraction satisfies core-retainment.*

Proof: Let $\beta \in B \setminus (B-\alpha)$. Then, there is $X \in \gamma^*(B^*\perp\alpha)$ such that $\beta \notin X$. As γ^* extends γ, there must be $Y \in \gamma(B\perp\alpha)$ such that $Y \subseteq X$. Then take $B' = Y$. □

Another way to relax relevance is to follow what was done with inclusion and recovery. Instead of requiring inclusion, we can require logical inclusion:

(logical relevance) If $\beta \in B \setminus (B-\alpha)$, then there is some B' such that $B-\alpha \subseteq B' \subseteq Cn(B)$, $\alpha \notin Cn(B')$ and $\alpha \in Cn(B' \cup \{\beta\})$.

Logical relevance is satisfied by any general partial meet pseudo-contraction whenever Cn^* selects only classical consequences of the set, i.e., $Cn^*(A) \subseteq Cn(A)$:

Proposition 13 *If for every set A, $Cn^*(A) \subseteq Cn(A)$, then general partial meet pseudo-contraction satisfies logical relevance.*

Proof: Let $\beta \in B \setminus (B-\alpha)$. Then, there is $X \in \gamma^*(B^*\perp\alpha)$ such that $\beta \notin X$. Since $B^* \subseteq Cn(B)$, we can make $B' = X$. □

Propositions 13 and 11 show that if Cn^* is such that for every set A, $Cn^*(A) \subseteq Cn(A)$, then general partial meet pseudo-contraction satisfies logical inclusion, logical vacuity, success, logical relevance and extensionality. Moreover, the operation on belief sets generated from it satisfies closure, inclusion, vacuity, success, relevance and extensionality. (Fuhrmann and Hansson 1994) has shown that for belief sets, if an operation satisfies relevance, then it satisfies recovery. We have then the following corollary:

Corollary 14 *A belief set contraction generated from general partial meet pseudo-contraction where for every set A, $Cn^*(A) \subseteq Cn(A)$, satisfies the six AGM postulates for contraction.*

Minimal Additions

In the previous section, we have presented a general construction for contraction without inclusion which was based on the idea of first expanding the base and then applying partial meet to it. In this section, we follow a different approach, closer to Nebel's proposal: we first apply partial meet contraction to the base and then expand the result. In the end of this section we show the relation between both approaches.

Recall that Nebel's construction (Definition 6) adds the set $\{\alpha \rightarrow \beta | \beta \in B\}$ to the result of the partial meet contraction in order to obtain logical recovery. We have shown in Proposition 7 that it suffices to add $\{\alpha \rightarrow \beta | \beta \in B \setminus \bigcap \gamma(B\perp\alpha)\}$. This means adding an implication for each formula that was given up in the contraction. Can we do with less than that? That is, are there cases in which we can add less than that and still retain logical recovery? The answer is clearly yes: consider a base containing $\{p \wedge \neg r, p \wedge \neg r \wedge q\}$ and a contraction by p. Both formulas of the base are given up, but in order to be logically recoverable, we only need to add $p \rightarrow p \wedge \neg r \wedge q$.

We would like to ensure that only formulas really needed for logical recovery are added in the contraction operation. The following postulate is one option:

(core-addition) If $\beta \in (B-\alpha) \setminus B$, then there exist $\beta' \in B \setminus (B-\alpha)$ and $B' \subseteq B-\alpha$ such that $\alpha \rightarrow \beta' \notin Cn(B')$ but $\alpha \rightarrow \beta' \in Cn(B' \cup \{\beta\})$.

This postulate assures that if a new formula is added when performing a contraction, it is needed in order for some belief that was given up to be recoverable. Core-addition provides a counterpart to core-retainment with respect to minimal change: while core-retainment prevents unnecessary loss of beliefs, core-addition avoids unnecessary addition of new beliefs. In particular, the examples 2 and 3 of the introduction do not satisfy inclusion, but they satisfy logical inclusion and core-addition. In the example 2 we can add q to the base because together with p it implies $p \wedge q$. Likewise, in the example 3 we can add $b \wedge \neg p \rightarrow f$ because it helps to recover $b \rightarrow f$.

We are now left with the issue of finding constructions that satisfy core-addition (together with logical inclusion, logical recovery and success). The following definition provides such a construction, using the idea of a minimal set that together with the partial meet contraction recovers the base but does not imply the sentence being contracted:

Definition 15 *Let B be a belief base, α a formula and γ a selection function for B. Let $\Delta(B, \alpha, \gamma)$ be a minimal subset of $Cn(B)$ such that:*

- $\alpha \rightarrow \beta \in Cn(\bigcap \gamma(B\perp\alpha) \cup \Delta(B, \alpha, \gamma)$ *for all* $\beta \in B \setminus (\bigcap \gamma(B\perp\alpha))$
- *For all $X \in B\perp\alpha$, we have $\alpha \notin Cn(X \cup \Delta(B, \alpha, \gamma))$*

We define the Δ-partial-meet pseudo-contraction of B by α as $B - \alpha = \bigcap \gamma(B\perp\alpha) \cup \Delta(B, \alpha, \gamma)$

Given B, α and γ, $\Delta(B, \alpha, \gamma)$ chooses one of the minimal sets that satisfy the two properties, as there may be more than one. We know that at least one such set exists, since the set used in Nebel's construction $\{\alpha \rightarrow \beta | \beta \in B\}$ satisfies the two properties and hence there must be a minimal subset of it that satisfies the properties.

Proposition 16 *Δ-Partial-meet pseudo-contraction satisfies success, logical inclusion, logical recovery, logical vacuity, and core-addition.*

Note that the construction does not always satisfy extensionality, since Δ depends on the particular formula being contracted. If Δ is such that for logically equivalent formulas α and β we always have $\Delta(B, \alpha, \gamma) = \Delta(B, \beta, \gamma)$ then Δ-partial-meet pseudo-contraction satisfies extensionality.

In this case, the operation on belief sets generated from Δ-partial-meet pseudo-contraction satisfies the six AGM postulates for contraction.

We will now show that this construction is a special case of general partial meet pseudo-contraction (Definition 10). Recall that general partial meet pseudo-contraction was defined as $\bigcap\gamma^*(B^*\perp\alpha)$, where $B^* = B \cup Cn^*(B \setminus \bigcap\gamma(B\perp\alpha))$ and γ^* is an extension of γ. The definition of Cn^* was allowed to vary, and only for some definitions we had logical recovery. Let us now consider the case where $Cn^*(B \setminus \bigcap\gamma(B\perp\alpha)) = \Delta$. We will show that $\bigcap\gamma^*(B^*\perp\alpha) = \bigcap\gamma(B\perp\alpha) \cup \Delta$, i.e., that the two operations coincide.

Proposition 17 *Let Δ be a minimal subset of $Cn(B)$ such that:*

- $\alpha \rightarrow \beta \in Cn(\bigcap\gamma(B\perp\alpha) \cup \Delta)$ *for all* $\beta \in B \setminus (\bigcap\gamma(B\perp\alpha))$
- *For all $X \in B\perp\alpha$, we have $\alpha \notin Cn(X \cup \Delta)$*

and let $Cn^*(B \setminus \bigcap\gamma(B\perp\alpha)) = \Delta$. Then $\bigcap\gamma^*(B^*\perp\alpha) = \bigcap\gamma(B\perp\alpha) \cup \Delta$.

Proof: With $Cn^*(B \setminus \bigcap\gamma(B\perp\alpha)) = \Delta$ we have that $B^* = B \cup \Delta$. From the definition of the extension γ^* we know that for every element Y of $\gamma^*((B \cup \Delta)\perp\alpha)$ there is an $X \in \gamma(B\perp\alpha)$ such that $X \subseteq Y$. From the definition of remainders, since X is a maximal subset of B that does not imply α, and since by the definition of Δ we know that $X \cup \Delta$ does not imply α, Y must be $X \cup \Delta$. This means that $\Delta \subseteq Y$ for all $Y \in \gamma^*((B \cup \Delta)\perp\alpha)$ and hence, $\Delta \subseteq \bigcap\gamma^*((B \cup \Delta)\perp\alpha)$. And again by the definition of γ^*, $\bigcap\gamma^*((B \cup \Delta)\perp\alpha) = \bigcap\gamma(B\perp\alpha) \cup \Delta$. □

This proposition shows that Δ-partial-meet pseudo-contraction is a special case of general partial-meet pseudo-contraction that satisfies logical recovery.

Degrees of recovery

In the construction defined above, we have added enough formulas to the belief base in order to be able to recover all the information that was given up during contraction. Δ-Partial-meet pseudo contraction satisfies logical recovery and is minimal in the sense captured by the core-addition postulate.

As exemplified in the introduction, recovery, as well as the logical recovery postulate can not always be accepted. In some cases, however, we may be interested in partial recovery, i.e., in making sure that a subset of the previous beliefs can be recovered. The core-addition postulate tries to avoid unnecessary addition of formulas to the belief base. The additions are minimal in the sense that if some formula is added to the base it is because it helps to recover some previously removed formula.

Δ-Partial-meet pseudo contraction satisfies core-addition as well as logical recovery. However it is possible to define a construction for contraction that satisfies core-addition, but does not necessarily satisfy logical recovery. For this purpose we need to define a function that chooses the formulas that we want to keep recoverable. In this sense, we can talk about different degrees of recovery. This function must return some elements of $B \setminus \bigcap\gamma(B\perp\alpha)$. Formally, given a set of formulas, let f choose a subset of it (for any set X, $f(X) \subseteq X$). We can easily adapt Definition 15 in order to have partial recovery controlled by f, where Δ_f then is the minimal set that allows recovery of the formulas chosen by f:

Definition 18 *Let f be a function as defined above, B a belief base, α a formula and γ a selection function for B. Let $\Delta_f(B, \alpha, \gamma)$ be a minimal subset of $Cn(B)$ such that:*

- $\alpha \rightarrow \beta \in Cn(\bigcap\gamma(B\perp\alpha) \cup \Delta_f(B, \alpha, \gamma))$ *for all* $\beta \in f(B \setminus (\bigcap\gamma(B\perp\alpha)))$
- *For all $X \in B\perp\alpha$, we have $\alpha \notin Cn(X \cup \Delta_f(B, \alpha, \gamma))$.*

We define the Δ_f-partial-meet pseudo-contraction of B by α as $B - \alpha = \bigcap\gamma(B\perp\alpha) \cup \Delta_f(B, \alpha, \gamma)$.

This construction makes clear the relation between the different degrees of recovery and inclusion. At one extreme, if f selects the whole set, we have logical recovery at the price of adding new formulas to the contracted base. At the other extreme, if f selects the empty set, the contraction operation satisfies inclusion, i.e., no new formulas are added. Between these two extremes there is a whole universe of contractions that recover parts of the base, each of which satisfies success, logical inclusion and core-addition.

The adapted construction does not satisfy logical recovery in general, but inherits relevance (and thus, also core-retainment) from the usual partial-meet contraction included in it.

And similarly to what we did to Δ-partial-meet pseudo contraction, we can show that:

Proposition 19 *If $Cn^*(B \setminus \bigcap\gamma(B\perp\alpha)) = \Delta_f(B, \alpha, \gamma)$, then $\bigcap\gamma^*(B^*\perp\alpha) = \bigcap\gamma(B\perp\alpha) \cup \Delta_f(B, \alpha, \gamma)$*

This proposition shows that if Cn^* is such that it chooses the same elements as Δ_f, then Δ_f-partial-meet pseudo-contraction is a special case of general partial meet pseudo-contraction.

Conclusions and Future Work

In this paper, we have discussed how weakening the inclusion requirement can provide some sort of recovery for operations on belief bases.

We have proposed two general constructions based on partial meet that allow some consequences of the original belief base to be added during contraction. We have shown that Nebel's construction (Nebel 1989) is a special case of our proposals. We have then shown that our proposal allows for operations that satisfy recovery (like Nebel's), but also operations that only satisfy weaker versions of it.

We are left with a whole spectrum of operations varying from those where any formula can be added (i.e., no inclusion postulate) to those where nothing can be added. In between, when we use a weaker notion of inclusion, there are constructions satisfying recovery in different degrees.

Future work includes studying other possible construction and proving representation results for them. We also plan

to explore the impact of different choices of Cn^*, γ^* and $\Delta(B, \alpha, \gamma)$.

Definition 15 has connections to abduction that we plan to investigate. An abduction problem consists in, given a set A and a formula α, finding a set X such that $A \cup X \vdash \alpha$ and $A \cup X \nvdash \bot$. In our case, we have the set $(B - \alpha) + \alpha$ and we are looking for a set Δ so that $((B-\alpha)+\alpha) \cup \Delta$ implies all formulas of B.

Acknowledgments: The first author is supported by FAPESP (grant 2006/53028-6) and the second author is partially supported by CNPq (grant 302514/2007-4). This work was developed as part of FAPESP project 2004/14107-2.

References

Alchourrón, C., and Makinson, D. 1982. On the logic of theory change: contraction functions and their associated revision functions. *Theoria* 48:14–37.

Alchourrón, C.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change. *Journal of Symbolic Logic* 50(2):510–530.

Booth, R.; Chopra, S.; Ghose, A.; and Meyer, T. 2005. Belief liberation (and retraction). *Studia Logica* 79:47–72.

Fermé, E. 2001. Five faces of recovery. In Rott, H., and Williams, M.-A., eds., *Frontiers in Belief Revision*. Kluwer.

Fuhrmann, A., and Hansson, S. O. 1994. A survey of multiple contraction. *Journal of Logic, Language and Information* 3(1):39–74.

Gärdenfors, P., and Rott, H. 1995. Belief revision. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume IV. Oxford University Press. chapter 4.2.

Gärdenfors, P. 1988. *Knowledge in Flux - Modeling the Dynamics of Epistemic States*. MIT Press.

Hannson, S. O. 1991. Belief contraction without recovery. *Studia Logica* 50(2):251–260.

Hansson, S. O. 1989. New operators for theory change. *Theoria* 55:114–132.

Hansson, S. O. 1992. Reversing the Levi identity. *Journal of Philosophical Logic* 22:637–639.

Hansson, S. O. 1999. *A Textbook of Belief Dynamics*. Kluwer Academic Press.

Makinson, D. 1987. On the status of the postulate of recovery in the logic of theory change. *Journal of Philosophical Logic* 16:383–394.

Maranhao, J. 2001. Refinement: a tool to deal with inconsistencies. In *ICAIL '01: Proceedings of the 8th international conference on Artificial intelligence and law*, 52–60. ACM.

Nebel, B. 1989. A knowledge level analysis of belief revision. In Brachman, R.; Levesque, H.; and Reiter, R., eds., *First International Conference on Principles of Knowledge Representation and Reasoning - KR'89*, 301–311. Toronto, ON: Morgan Kaufmann.

Rott, H., and Pagnucco, M. 1999. Severe withdrawal (and recovery). *Journal of Philosophical Logic* 28(5):501–547.

Consistency Maintenance of Plausible Belief Bases Based on Agents Credibility

Luciano H. Tamargo Alejandro J. García Marcelo A. Falappa Guillermo R. Simari
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Artificial Intelligence Research and Development Laboratory,
Department of Computer Science and Engineering - Universidad Nacional del Sur (UNS),
Bahía Blanca, ARGENTINA,
e-mail: {lt, ajg, maf, grs}@cs.uns.edu.ar

Abstract

In this work we address the problem of knowledge representation in a collaborative multi-agent system where agents can obtain new information from others through communication. Informant agents will be ranked by their credibility. This credibility order will be used when new incoming information is contradictory with the current agent's belief base. We propose a method for analyzing the received information, if inconsistency arises, the credibility order will be used to decide which information prevails. The proposed operator satisfies the minimal change principle and incoming information can be rejected when the agent has more credible beliefs that contradict the new information.

Introduction

In this paper we propose a formalism for knowledge representation and consistency maintenance in a multi-agent system where deliberative agents can receive new information from others through communication.

A variety of notations have been adopted by researchers investigating *Belief Revision* (BR) in *Multi-Agent Systems* (MAS). A good understanding of the relationships between these approaches is essential before carrying out any further research. In (Liu and Williams 1999) an exhaustive analysis of these approaches is presented and a very interesting hierarchy is introduced (See Figure 1). Observe that in the hierarchy, *Multi-Agent Belief Revision* (MABR) and *Belief Revision using information from Multiple Sources* (MSBR) are distinguished.

As stated in (Liu and Williams 1999), BR could be considered as part of the agent's skills to maintain the consistency of its own epistemic state. In this case, an individual BR process is carried out in a multi-agent environment, where the new information may come from multiple sources and maybe conflict. BR in this sense is called MSBR by (Dragoni, Giorgini, and Baffetti 1997). Cantwell (Cantwell 1998) tries to resolve conflicting information by ordering the information sources on the basis of their trustworthiness. This could be served as a rational way of generating the new information credibility based on the source reliability using the terms of MSBR.

However, as discussed in (Liu and Williams 1999), BR could also be used to achieve a society's or team's mutual belief goals (*e.g.* reaching consensus before carrying out plans). In this setting, more than one agent takes part in the process. In order to pursue the mutual goal, agents involved need to communicate, cooperate, coordinate and negotiate with one another. A MABR system is a MAS whose mutual goal involves BR.

MSBR studies individual agent revision behaviors, *i.e.*, when an agent receives information from multiple agents towards whom it has social opinions. MABR investigates the overall BR behavior of agent teams or a society. MSBR is one of the essential components of MABR.

The AGM paradigm (Alchourrón, Gärdenfors, and Makinson 1985) has been widely accepted as a standard framework for BR. But it is only capable of prescribing revision behaviors of a single agent. The BR process is more complex in multiple agent case. Besides the Principle of Minimal Change, there exist other requisites due to the sophisticated agent interactions.

An agent is capable of carrying out Individual Belief Revision (IBR), while an agent society or team is capable of MABR. IBR in a single agent environment (Single Belief Revision, SBR) could be achieved using classical BR satisfying or adapting AGM postulates. IBR in a multiple agent environment is MSBR, *i.e.*, a single agent will have to process information coming from more than one source.

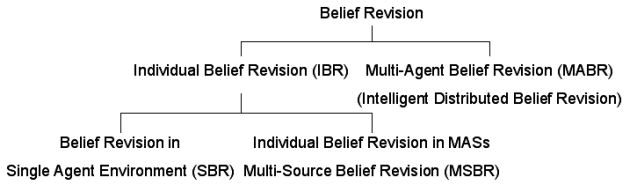


Figure 1: Belief Revision Hierarchy

Different formalisms have been presented to deal with MABR (Liu and Williams 1999; 2001; Kfir-Dahav and Tennenholz 1996). In (Liu and Williams 1999; 2001) an ontology to solve MABR is defined. That is, in these papers three major categories of heterogeneity, namely social, semantic and syntactic heterogeneity are clarified. There, is shown that several issues posed by such heterogeneities are

Partially supported by CONICET (PIP 5050), UNS and Agencia Nacional de Promoción Científica y Tecnológica.

addressed in the context of BR. They also propose the use of ontology as a tool to handle the heterogeneity issues so as to achieve the necessary reliable communication and system interoperability required by MABR.

In (Kfir-Dahav and Tennenholz 1996) research on MABR in the context of heterogeneous systems is initiated. The Private Domains (PD_i) and the Shared Domain (SD) of the agent knowledge base are defined in order to capture a general setting where each agent has private beliefs as well as beliefs shared with other agents. Under such knowledge structure, each agent may have its own perspective of the world but needs to coordinate (*i.e.* agree on) its belief on shared elements. The shared domain also defines the communication language for the agents.

In contrast to these last two proposals (Liu and Williams 1999; 2001; Kfir-Dahav and Tennenholz 1996), in our approach we focus on MSBR, where agents maintain the consistency of their belief bases. Since an agent can receive information that is contradictory with its own beliefs, in order to maintain its belief base consistent, it has to decide whether to accept or reject the new information. If an agent decides to accept a new belief that is contradictory with its belief base, then it has to select some beliefs from its belief base in order to withdraw them and avoid the contradiction. In this article, we will assume that an agent does not receive contradictions.

Similarly to Dragoni (Dragoni, Giorgini, and Puliti 1994) and Cantwel (Cantwell 1998), in our approach, informant agents can have different credibility and this credibility will be used to decide which information prevails when a contradiction arises. However, there are differences with these authors, which will be analyzed in the related work section.

In our approach, a credibility order among agents will be defined, and this order will be used by all the participants of the multi-agent system. We will assume that the credibility order among agents is fixed. To decide whether to reject or accept a new belief, a comparison criterion among beliefs will be defined. As it will be explained in detail below, this comparison criterion (that we call plausibility) will be based on the credibility order among agents.

Based on our criterion that compares agent beliefs, a revision operator will be proposed. This operator will satisfy the following principles:

- Maintenance of Consistency (Dalal 1988): If a belief base K and a belief α are both consistent, then K revised by α is consistent.
- Minimal Changes: As much old knowledge as possible should be retained in the revised knowledge.
- Non-Prioritization: If a belief base K is revised by a belief α, the new belief is not necessarily accepted in the revised belief base.

Since in our proposal plausibility of sentence is based on the credibility order, our approach differs from (Benferhat, Dubois, and Prade 1998) where they investigate revision of information from multiple sources in face of uncertainty as data fusion, using possibilistic logic.

As we will show in the next section, agent's epistemic states will be represented by belief bases (Fuhrmann 1991; Hansson 1992). The most widely studied method of changing a belief state is *partial meet contraction-revision*, also known as AGM model (Alchourrón, Gärdenfors, and Makinson 1985). The AGM model represents epistemic states by means of belief sets, that is, sets of sentences closed under logical consequence. However, our epistemic model will use belief bases, that is, sets of sentences not necessarily closed. As it will be explained below, we will adapt the notion of kernel contraction (Hansson 1994; 1999) to our epistemic model, in which the beliefs are provided by agents.

This paper is organized as follows. Next section introduces the epistemic model. The third section defines a plausibility order among beliefs. Then, in the fourth section a non-prioritized revision operator that uses the plausibility order is introduced. In the fifth section a prioritized version of the revision operator is defined. Retransmission of information is considered in the sixth section. Finally, related work and conclusions are included.

Epistemic Model

In this work, we will consider a finite set of agents identifiers that will be denoted as $\mathcal{A}gents$. Since agents can obtain information from other agents, agents' beliefs will be represented as tuples (α, A), where $A \in \mathcal{A}gents$ and α is a sentence of a propositional language $\mathcal{L}$. We will adopt a propositional language $\mathcal{L}$ with a complete set of boolean connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. Also, we assume the existence of an operator Cn that satisfies *inclusion* ($A \subseteq Cn(A)$), *iteration* ($Cn(A) = Cn(Cn(A))$), and *monotony* (if $A \subseteq B$ then $Cn(A) \subseteq Cn(B)$) and it includes the classical consequence operator. In general, we will write $\alpha \in Cn(A)$ as $A \vdash \alpha$.

Let $\mathcal{K} = 2^{\mathcal{L} \times \mathcal{A}gents}$, each agent $A \in \mathcal{A}gents$ will have a belief base $K_A \in \mathcal{K}$. As stated above, informant agents will be ranked by their credibilities. Hence, a Credibility Order over the set $\mathcal{A}gents$ will be introduced:

Definition 1 *A **credibility order among agents**, denoted by infix '$\leq_{Co}$', is a total order over $\mathcal{A}gents$, where $A_1 \leq_{Co} A_2$ means that A_2 is at least as credible as A_1. The strict relation $A_1 <_{Co} A_2$, representing A_2 is strictly more credible than A_1, is defined as $A_1 \leq_{Co} A_2$ and $A_2 \not\leq_{Co} A_1$. Moreover, $A_1 =_{Co} A_2$ means that A_1 is as credible as A_2, and it holds when $A_1 \leq_{Co} A_2$ and $A_2 \leq_{Co} A_1$.*

Since this order is total, then the following properties hold for all A_1, A_2 and $A_3 \in \mathcal{A}gents$:

- Totality or Completeness: $A_1 \leq_{Co} A_2$ or $A_2 \leq_{Co} A_1$.
- Transitivity: if $A_1 \leq_{Co} A_2$ and $A_2 \leq_{Co} A_3$ then $A_1 \leq_{Co} A_3$.
- Antisymmetry: if $A_1 \leq_{Co} A_2$ and $A_2 \leq_{Co} A_1$ then $A_1 =_{Co} A_2$.

Example 1 *Consider a set $\mathcal{A}gents = \{A_1, A_2, A_3, A_4\}$ where the credibility order is $A_1 \leq_{Co} A_2$, $A_2 \leq_{Co} A_3, A_3 \leq_{Co} A_2, A_3 \leq_{Co} A_4$. Note that A_2 is as credible as A_3. The belief base of the agent A_1 is $K_{A_1} = \{(\beta, A_1), (\alpha, A_2), (\alpha, A_3), (\alpha \rightarrow \beta, A_2), (\alpha \rightarrow \beta, A_4), (\omega, A_1), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1),$*

$(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_2), (\epsilon \rightarrow \beta, A_3), (\epsilon \rightarrow \beta, A_4)\}$. *Observe that K_{A_1} has three tuples with the sentence $\epsilon \rightarrow \beta$. Although this can be considered as redundancy, each one comes from a different informant agent. The reasons for maintaining all of them will be explained below.*

Next, two auxiliary functions are introduced in order to obtain the set of sentences (resp. set of agents) that belong to a belief base $K \in \mathcal{K}$.

Definition 2 *The **sentence function**, $Sen : \mathcal{K} \mapsto 2^{\mathcal{L}}$, is a function such that for a given belief base $K \in \mathcal{K}$, $Sen(K) = \{\alpha : (\alpha, A) \in K$ for any A in $\mathcal{A}gents\}$.*

Definition 3 *The **agent identifier function**, $Ag : \mathcal{K} \mapsto 2^{\mathcal{A}gents}$, is a function such that for a given belief base $K \in \mathcal{K}$, $Ag(K) = \{A : (\alpha, A) \in K$ for any α in $\mathcal{L}\}$.*

Example 2 *Let us consider again the belief base K_{A_1} showed in Example 1. Then*

- $Sen(K_{A_1}) = \{\beta, \alpha, \alpha \rightarrow \beta, \omega, \omega \rightarrow \beta, \alpha \rightarrow \delta, \delta \rightarrow \beta, \gamma, \gamma \rightarrow \epsilon, \epsilon \rightarrow \beta\}$, *and*
- $Ag(K_{A_1}) = \{A_1, A_2, A_3, A_4\}$.

As stated above, agents can receive new beliefs from other informants. This new information can be contradictory with their current beliefs. For instance, consider again the belief base (K_{A_1}) of Example 1, where $Sen(K_{A_1}) \vdash \beta$ (observe that there are several derivations for β). Suppose now that the agent A_1 receives the input $(\neg\beta, A_4)$. It is clear that adding $(\neg\beta, A_4)$ to K_{A_1} will produce an inconsistent belief base. Therefore, the agent has to revise its beliefs and decide whether it rejects $(\neg\beta, A_4)$ or it withdraws β. The credibility order will be used to decide which information prevails. However, since there can be several derivations of β, then we have to "cut" all of them. For doing that, all the minimal subsets of K_{A_1} that entails β will be obtained, using an extension of *Kernel contractions*.

Remark 1 *It is important to note that the agent identifier in a tuple may differ from the sender identifier (see "Retransmission of Information" Section).*

Kernel contractions introduced in (Hansson 1994) are based on a selection among the sentences that are relevant to derive the sentence to be retracted. In order to perform a contraction, kernel contractions use incision functions which cut into the minimal subsets that imply the information to be given up. We will adapt the notion of kernel contraction to our epistemic model. First, we will define the kernel set (Definition 4) and then we will present incision functions (Definition 13) that will cut beliefs according to their plausibility (Definition 10).

Definition 4 *Let $K \in \mathcal{K}$ and $\alpha \in \mathcal{L}$. Then $H \in K^{\perp\!\!\!\perp}\alpha$ if and only if*

1. $H \subseteq K$.
2. $Sen(H) \vdash \alpha$.
3. *if $H' \subset H$, then $Sen(H') \nvdash \alpha$.*

The set of minimal subsets of a belief base $K \in \mathcal{K}$ that imply α (denoted $K^{\perp\!\!\!\perp}\alpha$) is called a *kernel set*. Note that each α-kernel ($H \in K^{\perp\!\!\!\perp}\alpha$) is a set of tuples from K.

Example 3 *Consider K_{A_1} of Example 1. $K^{\perp\!\!\!\perp}_{A_1}\beta = \{H_a, H_b, H_c, H_d, H_e, H_f, H_g, H_h, H_i, H_j, H_k\}$ where*
$H_a = \{(\beta, A_1)\}$,
$H_b = \{(\alpha, A_2), (\alpha \rightarrow \beta, A_2)\}$,
$H_c = \{(\alpha, A_2), (\alpha \rightarrow \beta, A_4)\}$,
$H_d = \{(\alpha, A_3), (\alpha \rightarrow \beta, A_2)\}$,
$H_e = \{(\alpha, A_3), (\alpha \rightarrow \beta, A_4)\}$,
$H_f = \{(\omega, A_1), (\omega \rightarrow \beta, A_4)\}$,
$H_g = \{(\alpha, A_2), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1)\}$,
$H_h = \{(\alpha, A_3), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1)\}$,
$H_i = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_2)\}$,
$H_j = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_3)\}$ *and*
$H_k = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.

The information (α, A_p) that an agent A_i receives from A_p could be consistent with its current belief base K_{A_i} if $Sen(K_{A_i}) \nvdash \alpha$ or $Sen(K_{A_i}) \vdash \alpha$. If $Sen(K_{A_i}) \nvdash \alpha$, then it is clear that (α, A_p) is added to K_{A_i}. If $Sen(K_{A_i}) \vdash \alpha$ then (α, A_p) is also added to K_{A_i} because the plausibility of α may increase (see "Sentences Plausibility" Section below). Therefore, a belief base $K \in \mathcal{K}$ may contain the same belief in two tuples with different agents identifiers (for instance, $\{(\alpha, A_1), (\alpha, A_2)\} \subseteq K$). Thus, we may say that K has redundant information or we may say that K is redundant. In Example 1 the sentence $\epsilon \rightarrow \beta$ is in three tuples. From the tuples point of view there is no redundancy, due to each tuple represent to a different informant. Nevertheless, in this work a kernel set will be computed over a non-redundant belief base. Given a (possibly redundant) belief base K, the non-redundant belief base K' of K (see Definition 6) will not contain two tuples (α, A_i) and (α, A_j), where $i \neq j$, for any sentence α in $Sen(K)$. Each tuple $(\alpha, A_i) \in K'$ has the agent identifier A_i of the most credible source of α. For this purpose, a maximum belief base function will be defined next. Moreover, since a multi-agent system may contain agents with the same degree of credibility, throughout the rest of this paper a lexicographic order among agents identifiers will be assumed. For instance, assuming set of agent identifiers $\mathcal{A}gents = \{A_1, A_2, A_3\}$, identifier A_1 is considered the lowest lexicographically.

Definition 5 *The **top agent function**, $Top : \mathcal{L} \times \mathcal{K} \mapsto \mathcal{K}$, is a function such that for a given sentence $\alpha \in \mathcal{L}$ and a given belief base $K \in \mathcal{K}$, $Top(\alpha, K) = \{(\alpha, A_i) : (\alpha, A_i) \in K$ and for all $(\alpha, A_j) \in K, A_j \leq_{Co} A_i\}$.*

Definition 6 *The **maximum belief base function**, $TopBase : \mathcal{K} \mapsto \mathcal{K}$, is a function such that for a given belief base $K \in \mathcal{K}$, $TopBase(K) = \{(\alpha, A) : (\alpha, A) \in Top(\alpha, K)$ and A is lexicographically the lowest agent identifier of $Ag(Top(\alpha, K))\}$.*

Example 4 *Let us consider again the belief base K_{A_1} showed in Example 1. Then,*

- $Top(\epsilon \rightarrow \beta, K_{A_1}) = \{(\epsilon \rightarrow \beta, A_4)\}$.
- $TopBase(K_{A_1}) = \{(\beta, A_1), (\alpha, A_2), (\alpha \rightarrow \beta, A_4), (\omega, A_1), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.
- $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta = \{H_a, H_e, H_f, H_h, H_n\}$ *where*
  $H_a = \{(\beta, A_1)\}$,

$H_e = \{(\alpha, A_2), (\alpha \rightarrow \beta, A_4)\}$,
$H_f = \{(\omega, A_1), (\omega \rightarrow \beta, A_4)\}$,
$H_h = \{(\alpha, A_2), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1)\}$ *and*
$H_k = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.

Let $K \in \mathcal{K}$, note that $Sen(TopBase(K)) = Sen(K)$ and $TopBase(K) \subseteq K$. In case that $TopBase(K) \subset K$ then there is some sentence (β) in $Sen(K)$ such that it is in m tuples in K. Consider $\alpha \in \mathcal{L}$ and $X \in TopBase(K)^{\perp\!\!\!\perp}\alpha$, if $\beta \in Sen(X)$ then $K^{\perp\!\!\!\perp}\alpha$ will have at least m α-*kernels* differing only in the agent identifier of the tuple in which β is in. On the other hand, this will not happen with $TopBase(K)$ because in $TopBase(K)^{\perp\!\!\!\perp}\alpha$ there will be only one belief β in the greatest weighted tuple from those m tuples according to $\leq_{Co}$. Therefore, $TopBase(K)^{\perp\!\!\!\perp}\alpha \subseteq K^{\perp\!\!\!\perp}\alpha$. Thus, since it is more expensive to compute $K^{\perp\!\!\!\perp}\alpha$ than to compute $TopBase(K)^{\perp\!\!\!\perp}\alpha$, then as more tuples containing β are in K, the more expensive will be to compute $K^{\perp\!\!\!\perp}\alpha$. Hence, since to compute the plausibility of a sentence we will use α-kernels, we will use $TopBase(K)$ in order to do the computation. In the following section, we will prove that it is equivalent to compute the plausibility of a sentence either with $TopBase(K)$ or with K. Besides, it will be shown why keeping a non-redundant belief base is the best option.

Remark 2 *The use of $TopBase(\cdot)$ optimizes the computation of kernel sets.*

In the following section we will show how the new operator uses the additional information (agents identifiers) in order to guide the revision process. The plausibility of the sentences will be defined by using the agents identifiers stored in the belief bases of the agents and the credibility order among agents.

Sentences Plausibility

The agents identifiers (which are in the second field of the tuples) represent the information that will be used to compute the plausibility of the beliefs. That is, each agent's belief will have an associated plausibility that will depend on the agent identifier and the *credibility order among agents*. The behavior of the plausibility is similar to the epistemic entrenchment defined in (Gärdenfors and Makinson 1988). That is, if α and β are sentences in $\mathcal{L}$, the notation $\alpha \preceq_{K_A} \beta$ will be used as a shorthand for "β is at least as plausible as α relative to the belief base K of the agent A".

One belief base $K \in \mathcal{K}$ may contain either explicit sentences or entailed sentences. As stated above, the explicit sentences are those contained in $Sen(K)$. The entailed sentences are those such that they are not in $Sen(K)$ but they are entailed by sentences in $Sen(K)$. In order to obtain the entailed sentences from a belief base K we are going to use the following function:

Definition 7 *The **belief function**, $Bel : \mathcal{K} \mapsto 2^{\mathcal{L}}$, is a function such that for a given belief base $K \in \mathcal{K}$, K: $Bel(K) = \{\alpha : \alpha \in \mathcal{L}$ and $Sen(K) \vdash \alpha\}$.*

Note that also K may contain explicit sentences that are entailed by $Bel(K)$. Thus we will have several proofs for the same sentence in K. For instance, in Example 3 we showed that β has several proofs. Therefore, to calculate the plausibility of a sentence (β) we should analyze all its proofs. In order to achieve this, we are going to use the kernel sets. We consider that this calculation should be cautious. That is, from each β-kernel, we desire to obtain the lesser-plausibility tuples. This plausibility gives us the plausibility of each proof. Then, the plausibility of a derived sentence β will be the greater plausibility among the plausibilities of each β-kernel. In order to define that, two functions will be given next.

Definition 8 *The **lesser-credibility sources function**, $min : \mathcal{K} \mapsto \mathcal{K}$, is a function such that for a given belief base $K \in \mathcal{K}$, $min(K) = \{(\alpha, A_i) : (\alpha, A_i) \in K$ and for all $(\delta, A_j) \in K, A_i \leq_{Co} A_j\}$.*

Definition 9 *The **greater-credibility sources function**, $max : \mathcal{K} \mapsto \mathcal{K}$, is a function such that for a given belief base $K \in \mathcal{K}$, $max(K) = \{(\alpha, A_i) : (\alpha, A_i) \in K$ and for all $(\delta, A_j) \in K, A_j \leq_{Co} A_i\}$.*

Example 5 *Consider $\mathcal{A}gents = \{A_1, A_2, A_3\}$ where $A_1 \leq_{Co} A_2 \leq_{Co} A_3$. Let $K_{A_1} = \{(\alpha, A_1), (\alpha, A_2), (\beta, A_1), (\gamma, A_1), (\alpha \rightarrow \gamma, A_3)\}$ be the belief base of A_1. Then,*

- *$Bel(K_{A_1})$ will contain the sentences from $Sen(K_{A_1})$ plus the sentences derived by "$\vdash$". For instance, $\alpha, \beta, \gamma, \alpha \vee \beta, \alpha \vee \beta \vee \gamma, \ldots$, and so on.*
- *$min(K_{A_1}) = \{(\alpha, A_1), (\beta, A_1), (\gamma, A_1)\}$.*
- *$max(K_{A_1}) = \{(\alpha \rightarrow \gamma, A_3)\}$.*

Next, we will introduce a function that returns the plausibility of a sentence that can be explicitly in K or inferred from K.

Definition 10 *The **Plausibility function**, $Pl : \mathcal{L} \times \mathcal{K} \mapsto \mathcal{A}gents$, is a function such that for a given sentence $\alpha \in \mathcal{L}$ and a belief base $K \in \mathcal{K}$, $Pl(\alpha, K) =$ lexicographically the lowest agent identifier of $Ag(max(\bigcup_{X \in TopBase(K)^{\perp\!\!\!\perp}\alpha} min(X)))$.*

It is important to note that if $(\gamma, A_1) \in K_{A_1}$ then $Pl(\gamma, K_{A_1})$ could be different from A_1. For instance, consider the Example 5, then $Pl(\alpha, K_{A_1}) = A_2$, $Pl(\beta, K_{A_1}) = A_1$ and $Pl(\gamma, K_{A_1}) = A_2$. In Example 7 we will describe (step by step) how the *plausibility function* returns an agent identifier.

Definition 11 ***Plausibility Criterion**. Let $K_A \in \mathcal{K}$ be the belief base of the agent A and let $\{\alpha, \beta\} \subseteq Bel(K_A)$, then $\alpha \preceq_{K_A} \beta$ if and only if $Pl(\alpha, K_A) \leq_{Co} Pl(\beta, K_A)$.*

The strict relation $\alpha \prec_{K_A} \beta$, representing "β is more plausible than α", is defined as "$\alpha \preceq_{K_A} \beta$ and $\beta \not\preceq_{K_A} \alpha$". Moreover, $\alpha \simeq_{K_A} \beta$ means that α is as plausible as β, and it holds when $\alpha \preceq_{K_A} \beta$ and $\beta \preceq_{K_A} \alpha$. From the previous definition we can observe that the plausibility of the sentences inherits the properties of the *credibility order among agents* ('$\preceq_{K_A}$' is a total order on $\mathcal{L}$). Furthermore, note that the relation '$\preceq_{K_A}$' is only defined with respect to a given K_A (different belief bases may be associated with different ordering of plausibility, in Example 6 this situation is shown).

Example 6 *Consider a set* $\mathcal{A}gents = \{A_1, A_2, A_3\}$ *where the credibility order is* $A_1 \leq_{Co} A_2, A_2 \leq_{Co} A_3$. *Suppose that the agent* A_2 *has the following belief base* $K_{A_2} = \{(\alpha, A_1), (\beta, A_2), (\gamma, A_3)\}$, *and suppose that the agent* A_3 *has the following belief base* $K_{A_3} = \{(\alpha, A_1), (\beta, A_3), (\gamma, A_2)\}$. *Then, for both agents,* β *is more plausible than* α *(i.e.,* $\alpha \preceq_{K_{A_2}} \beta$ *and* $\alpha \preceq_{K_{A_3}} \beta$*). However, for* A_2, γ *is more plausible than* β *(*$\beta \preceq_{K_{A_2}} \gamma$*) whereas for* A_3, β *is more plausible than* γ *(*$\gamma \preceq_{K_{A_3}} \beta$*).*

Example 7 *Suppose that the agent* A_1 *from Example 1 needs to calculate the plausibility of* β. *In order to do so,* A_1 *will do the following steps.*

- Step 1. *Obtain the minimal subsets that derive* β *from the non-redundant belief base* K ($TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta$). *From Example 4 we can see that:* $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta = \{H_a, H_e, H_f, H_h, H_k\}$.
- Step 2. *Obtain from each* β*-kernel* $\in TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta$ *the set containing the lesser plausibility tuples determined by the lesser-credibility sources function "*min*":* $min(H_a) = \{(\beta, A_1)\}$, $min(H_e) = \{(\alpha, A_2)\}$, $min(H_f) = \{(\omega, A_1)\}$, $min(H_h) = \{(\delta \rightarrow \beta, A_1)\}$, $min(H_k) = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}$.
- Step 3. *Obtain from the tuples of the previous item, the set containing the greater plausibility tuples determined by the greater-credibility sources function "*max*":* $max(\{(\beta, A_1), (\alpha, A_2), (\omega, A_1), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}) = \{(\alpha, A_2), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}$.
- Step 4. *Obtain from the tuples of the previous item, the set containing the agents identifiers determined by the get agent identifier function "Ag":* $Ag(\{(\alpha, A_2), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}) = \{A_3, A_2\}$.
- Step 5. *The agent identifier that will be associated to the sentence is lexicographically the lowest of the agents identifiers that are in the set of the previous item. Here, the agent identifier* A_2 *will be associated to* β.

Therefore, $Pl(\beta, K_{A_1}) = A_2$. *Hence, when* β *is compared with other belief,* A_2 *will be used as the informant of* β *(the plausibility of* β *will be given by* A_2*).*

The plausibility calculation can be made with $TopBase(\cdot)$ or without it. Both ways return the same result (See Proposition 1). However, making the calculation without $TopBase(\cdot)$ requires to calculate more kernels than with $TopBase(\cdot)$.

Proposition 1 Let $K \in \mathcal{K}$ and let $\alpha \in Bel(K)$, then $Pl(\alpha, K) = Ag(max(\bigcup_{X \in TopBase(K)^{\perp\!\!\!\perp}\alpha} min(X))) = Ag(max(\bigcup_{X \in K^{\perp\!\!\!\perp}\alpha} min(X)))$.

Proof:

Let $\mathcal{A}gents = \{A_1, \ldots, A_n\}$. If $TopBase(K) = K$ then is trivially proved. If $TopBase(K) \subset K$ then there exists some sentence β in $Sen(K)$ such that β occurs in m tuples in K ($m \geq 2$). Consider $(\beta, A_i) \in X$ ($1 \leq i \leq n$) for some $X \in TopBase(K)^{\perp\!\!\!\perp}\alpha$ then $K^{\perp\!\!\!\perp}\alpha$ will have m β-$kernels$ $(X, Y_1, \ldots, Y_{m-1})$ such that they will differ only in the tuple containing β. Suppose that $(\beta, A_j^p) \in Y_p$ for all p ($1 \leq p \leq m-1$, $j \neq i$ and $1 \leq j \leq n$). Next, we will prove that X will contain the only relevant tuples to calculate the plausibility of α. There are three cases:

- If $min(X) = (\beta, A_i)$, then we have that $min(Y_p) = (\beta, A_j^p)$. This is because, by Definition 6, $A_j^p \leq_{Co} A_i$ for all p. Moreover $X, Y_1, ..., Y_{m-1}$ differ only in the tuple in that is β. Therefore, $max((\beta, A_i),(\beta, A_j^1),...,(\beta, A_j^{m-1}))$ = $(\beta, A_i) \in X$.
- If $min(X) \neq (\beta, A_i)$ and $min(Y_p) \neq (\beta, A_j^p)$, then min will return the same tuple in all the cases. This is because, $X, Y_1, ..., Y_{m-1}$ differ only in the tuple containing β.
- If $min(X) \neq (\beta, A_i)$ (suppose that $min(X) = (\omega, A_j)$) and $min(Y_p) = (\beta, A_j^p)$ for some p then since $(\omega, A_j) \in Y_p$, $A_j^p \leq_{Co} A_j$. Note that, if $min(Y_p) \neq (\beta, A_j^p)$ then by the previous case $min(Y_p) = (\omega, A_j)$. Hence, $max(min(X) \cup min(Y_1) \cup \ldots \cup min(Y_{m-1}))) = (\omega, A_j) \in X$.

Therefore, from the m β-kernels $(X, Y_1, \ldots, Y_{m-1})$ only X will contain the relevant tuples to calculate the plausibility of α. Then,
$Pl(\alpha, K) = Ag(max(\bigcup_{X \in TopBase(K)^{\perp\!\!\!\perp}\alpha} min(X))) = Ag(max(\bigcup_{X \in K^{\perp\!\!\!\perp}\alpha} min(X)))$. □

As stated above, function $TopBase(\cdot)$ is used to calculate plausibility. Recall that $TopBase(\cdot)$ optimizes the computation of kernel set. Moreover, since in this work the belief base of an agent may contain the same sentence in several different tuples, it could be natural to preserve only "the most plausible derivation" of each sentence. However, in the following example it is shown that this criterion may be problematic.

Example 8 *Consider* $\mathcal{A}gents = \{A_1, A_2, A_3\}$ *where* $A_3 \leq_{Co} A_2 \leq_{Co} A_1$. *Let* $K_{A_2} = \{(\beta \rightarrow \alpha, A_2), (\alpha, A_3)\}$ *be the belief base of* A_2. *Suppose that* A_2 *incorporates* (β, A_2) *to* K_{A_2}. *In this scenario* $K_{A_2} * (\beta, A_2) = K_{A_2} \cup \{(\beta, A_2)\}$, *there are two derivations for* α, *and* $Pl(\alpha, K) = A_2$. *Note that the plausibility of* α *was increased, and it is unnatural to withdraw sentences from* K_{A_2} *in order to preserve just one derivation of* α.

As we have shown in the previous example, it is very restrictive to have each sentence supported by only one derivation. For this reason, in this work the belief bases may be redundant. That is, a sentence α could be present in several tuples of the same belief base. Thus, the plausibility of a sentence will be determined only by the *plausibility function*. In the following section, we will define a non-prioritized revision operator that uses the sentences plausibility in order to guide the revision process.

Non Prioritized Revision Using Plausibility

In this section, the behavior of a new revision operator based on the sentences plausibility will be shown. In case that a belief base $K \in \mathcal{K}$ is revised by a tuple (α, A_i) we will have two cases:

- α is consistent with $Bel(K)$. This is the most simple case of characterizing from the logical point of view because it consists only in the addition of new tuples. In the limit case in which $\alpha \in Bel(K)$ then this operation could increase the plausibility of α.
- α is inconsistent with $Bel(K)$, that is $\neg\alpha \in Bel(K)$. This case requires a deeper analysis because: a) it is necessary to determine when the sentence will be accepted; and b) if the input is accepted then it is necessary to erase some tuples of K. For the second case we need to define an incision function on each α-kernel.

We will adapt the incision function definition proposed by (Hansson 1994) to our framework.

Definition 12 *An **incision function** σ for $K \in \mathcal{K}$ is a function such that for all α*

1. $\sigma(K^{\perp\!\!\!\perp}\alpha) \subseteq \cup(K^{\perp\!\!\!\perp}\alpha)$.
2. *if* $\emptyset \neq X \in K^{\perp\!\!\!\perp}\alpha$, *then* $X \cap \sigma(K^{\perp\!\!\!\perp}\alpha) \neq \emptyset$

The incision function selects the sentences to be discarded from K. Therefore, the subset of K that is not affected by the incision should equal the outcome of the contraction of K by α.

In the definition of *incision function* of Hansson's work is not specified how the function selects the sentences that will be discarded of each α-kernel. This can be solved with the sentences plausibility that we have defined above. The incision function σ will select the lesser plausibility sentences of each α-kernel. Hence, the new operator differs of the kernel revision operator defined by Hansson in the following issues:

1. The new operator will do an analysis to determine if the revision is necessary.
2. The sentences selection for the incision function will be defined.

According to 1, the new operator permits two options, completely accepts all the input, or completely rejects all the input. For this reason the new operator is non prioritized. Some non prioritized operators of the literature that completely accept or reject the input are Semi-Revision (Hansson 1997) and Screened Revision (Makinson 1997). Another operators may partially accept the new information, for instance Revision by a Set of Sentences (Falappa, Kern-Isberner, and Simari 2002) and Selective Revision (Fermé and Hansson 1999).

Next we will define a specific incision function, based on the beliefs plausibility, that will select the lesser plausibility sentences of each α-kernel (following the principle of minimal change).

Definition 13 *$\sigma_\downarrow$ is a **bottom incision function** for K if $\sigma_\downarrow$ is an incision function such that, $\sigma_\downarrow(K^{\perp\!\!\!\perp}\alpha) = \{(\delta, A_i) : (\delta, A_i) \in X \in K^{\perp\!\!\!\perp}\alpha$ and for all $(\beta, A_j) \in X$ it holds that $A_i \leq_{Co} A_j\}$.*

Example 9 *Consider a set $\mathcal{A}gents = \{A_1, A_2, A_3\}$ where the credibility order is $A_1 \leq_{Co} A_2$, $A_2 \leq_{Co} A_3$. Suppose that the agent A_2 has the following belief base $K_{A_2} = \{(\alpha, A_3), (\beta, A_2), (\beta \rightarrow \alpha, A_1), (\omega, A_1), (\omega \rightarrow \alpha, A_3), (\delta, A_1)\}$. Then, $K^{\perp\!\!\!\perp}_{A_2}\alpha = \{H_a, H_b, H_c\}$ where*
$H_a = \{(\alpha, A_3)\}$,
$H_b = \{(\beta, A_2), (\beta \rightarrow \alpha, A_1)\}$,
$H_c = \{(\omega, A_1), (\omega \rightarrow \alpha, A_3)\}$.
$\sigma_\downarrow(K^{\perp\!\!\!\perp}_{A_2}\alpha) = \{(\alpha, A_3), (\beta \rightarrow \alpha, A_1), (\omega, A_1)\}$.

Now that we have given the necessary background on the behavior of the new operator, the *Non-Prioritized Revision Using Plausibility* will be defined.

Definition 14 *Let $K \in \mathcal{K}$, let $\alpha \in \mathcal{L}$, let $TopBase(\cdot)$ be a maximum belief base function, and let $K^{\perp\!\!\!\perp}\alpha$ be the set of α-kernels of K. Let $\sigma_\downarrow$ a bottom incision function [1] for K. The operator "$\circ$", called **Non-Prioritized Revision Using Plausibility**, is defined as follow:*

$$K\circ(\alpha, A_i) = \begin{cases} K \cup \{(\alpha, A_i)\} & \text{if } \neg\alpha \notin Bel(K) \\ K & \text{if } \neg\alpha \in Bel(K) \\ & \text{and } A_i \leq_{Co} Pl(\neg\alpha, K) \\ (K \setminus X) \cup \{(\alpha, A_i)\} & \text{if } \neg\alpha \in Bel(K) \\ & \text{and } Pl(\neg\alpha, K) <_{Co} A_i \end{cases}$$

where: $X = \{(\omega, A_j) : \omega \in Sen(\sigma_\downarrow(TopBase(K)^{\perp\!\!\!\perp}\neg\alpha))$ and $(\omega, A_j) \in K\}$.

Example 10 *Consider a set $\mathcal{A}gents = \{A_1, A_2, A_3, A_4, A_5\}$ where the credibility order is $A_1 \leq_{Co} A_2$, $A_2 \leq_{Co} A_3, A_3 \leq_{Co} A_2, A_3 \leq_{Co} A_4, A_4 \leq_{Co} A_5$. Suppose that the agent A_1 has the following belief base $K_{A_1} = \{(\beta, A_1), (\alpha, A_2), (\alpha, A_3), (\alpha \rightarrow \beta, A_2), (\alpha \rightarrow \beta, A_4), (\omega, A_1), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_2), (\epsilon \rightarrow \beta, A_3), (\epsilon \rightarrow \beta, A_4)\}$. Furthermore, we suppose A_1 receives the tuple $(\neg\beta, A_5)$. Then, A_1 should revise K_{A_1} by $(\neg\beta, A_5)$. Next we will describe the behavior of the new operator step by step.*

- Step 1. *Obtain the minimal subsets that derive β from non-redundant belief base K_{A_1}.*
  $TopBase(K_{A_1}) = \{(\beta, A_1), (\alpha, A_2), (\alpha \rightarrow \beta, A_4), (\omega, A_1), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.
  $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta = \{H_a, H_b, H_c, H_d, H_e\}$ *where*
  $H_a = \{(\beta, A_1)\}$, $H_b = \{(\alpha, A_2), (\alpha \rightarrow \beta, A_4)\}$,
  $H_c = \{(\omega, A_1), (\omega \rightarrow \beta, A_4)\}$,
  $H_d = \{(\alpha, A_2), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1)\}$,
  $H_e = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.
- Step 2. *Apply the bottom incision function "$\sigma_\downarrow$" to $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta$ to obtain the set containing the lesser plausibility tuples from each β-kernel.*
  $\sigma_\downarrow(TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta) = \{(\beta, A_1), (\alpha, A_2), (\omega, A_1), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}$.
- Step 3. *Obtain from the tuples of the previous item, the set containing the greater plausibility tuples determined by the greater-credibility sources function "max"$(max(\sigma_\downarrow(TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta)))$.*
  $max(\{(\beta, A_1), (\alpha, A_2), (\omega, A_1), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}) = \{(\alpha, A_2), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}$.

[1] Observe that the outcome of a *bottom incision function* would be similar to that of a "*safe contraction*" (Alchourrón and Makinson 1985).

- Step 4. *Compare the agent identifier of the input tuple with the agent identifier of any tuple obtained from previous item (lets suppose (α, A_j)). If $A_5 \leq_{Co} A_j$ the operation has no effect, the input is rejected, i.e., $K \circ (\neg\beta, A_5) = K$. On the other hand, if $A_j <_{Co} A_5$ then $K \circ (\neg\beta, A_5) = K \setminus \{(\omega, A_j) : \omega \in Sen(\sigma_{\downarrow}(TopBase(K)^{\perp\!\!\!\perp}\neg\alpha))$ and $(\omega, A_j) \in K\} \cup \{(\alpha, A_i)\}$.*
  Since $A_2 \leq_{Co} A_5$, then $K_{A_1} \circ (\neg\beta, A_5) = \{(\alpha \rightarrow \beta, A_2), (\alpha \rightarrow \beta, A_4), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\epsilon \rightarrow \beta, A_2), (\epsilon \rightarrow \beta, A_3), (\epsilon \rightarrow \beta, A_4), (\neg\beta, A_5)\}$.
  If the input is (β, A_1) rather than (β, A_5), then the revision will not have effect because $A_1 \leq_{Co} A_2$.

Remark 3 *Note that, since the belief base may be redundant, in step 4 of Example 10 if the revision gives rise to a contraction then we will discard from K all those tuples whose sentences were selected by the bottom incision function without regarding the respective informants. Besides, note that our operator will never discard more plausible sentences than the input. This control can be seen in Step 4 of Example 10.*

Remark 4 *In this approach, we have assumed that the credibility order among agents is fixed. However, this order may be replaced and this will not affect the behavior of the operator. If the credibility order among agents changes, then the plausibility of all sentences may also change without changing the belief bases of the agents. This feature was one of the motivations for using agent identifiers instead of representing explicitly the plausibility of sentences as a number. For instance, consider a set $\mathcal{A}gents = \{A_1, A_2\}$ where the credibility order is $A_1 \leq_{Co} A_2$, $K_{A_1} = \{(\alpha, A_1), (\beta, A_2)\}$ and $K_{A_2} = \{(\omega, A_2), (\gamma, A_1)\}$. Hence, $\alpha \preceq_{K_{A_1}} \beta$ and $\gamma \preceq_{K_{A_2}} \omega$. If the credibility order changes to $A_2 \leq_{Co} A_1$ then $\beta \preceq_{K_{A_1}} \alpha$ and $\omega \preceq_{K_{A_2}} \gamma$. Note that the tuples in K_1 and K_2 remain unchanged.*

If the behavior of the new operator follows the Definition 14 then the operator follows the principles enunciated in the introduction of this paper. These are:

- Maintenance of Consistency (Dalal 1988): If a belief base K and a belief α are both consistent, then K revised by α is consistent.
- Minimal changes: As much old knowledge as possible should be retained in the revised knowledge. In other words, in this paper, if the revision gives rise to a contraction, then the revision should discard those less plausible beliefs.
- Non-Prioritization: If a belief base K is revised by a belief α, the new belief is not necessarily accepted in the revised belief base. In other words, in this paper the revision could have no effect if some beliefs, that will be possibly discarded, are more plausible than the input.

Lemma 1 The non-prioritized revision using plausibility follows the principle of Maintenance of Consistency.
Proof:

Let $K \in \mathcal{K}$ be a belief base and let $\alpha \in \mathcal{L}$. Suppose that K and α are consistent. By Definition 14 of *Non-Prioritized Revision Using Plausibility* we have three cases:

- If $\neg\alpha \notin Bel(K)$ then $K \circ (\alpha, A_i) = K \cup \{(\alpha, A_i)\}$.
  α and K are consistent, hence since $\neg\alpha \notin Bel(K)$ then $K \cup \{(\alpha, A_i)\}$ is consistent.
- If $\neg\alpha \in Bel(K)$ and $A_i \leq_{Co} Pl(\neg\alpha, K)$, then $K \circ (\alpha, A_i) = K$ and we are done.
- If $\neg\alpha \in Bel(K)$ and $Pl(\neg\alpha, K) <_{Co} A_i$ then $K \circ (\alpha, A_i) = (K \setminus X) \cup \{(\alpha, A_i)\}$ where $X = \{(\omega, A_j) : \omega \in Sen(\sigma_{\downarrow}(TopBase(K)^{\perp\!\!\!\perp}\neg\alpha))$ and $(\omega, A_j) \in K\}$.
  We must show that $\neg\alpha \notin Bel(K \setminus X)$.
  Suppose that $\neg\alpha \in Bel(K \setminus X)$. Then the $\neg\alpha$-kernels were not cut by the *bottom incision function*. That is, no sentences from $\neg\alpha$-kernel were removed. However, this is absurd by the Definition 13 of *bottom incision function*. The absurd comes from supposing $\neg\alpha \in Bel(K \setminus X)$. Then $\neg\alpha \notin Bel(K \setminus X)$.

Hence the non-prioritized revision using plausibility follows the principle of Maintenance of Consistency. □

Lemma 2 The non-prioritized revision using plausibility follows the principle of Minimal Change.
Proof: Straightforward by Definition 13.

Lemma 3 The non-prioritized revision using plausibility follows the principle of Non-Prioritization.
Proof: Straightforward by Definition 14.

In following section we will define a prioritized version of this operator. The prioritized revision using plausibility will not do an analysis to determine if the revision is necessary.

Prioritized Revision Using Plausibility

Note that we can easily define a prioritized version of the *Non-prioritized Revision Using Plausibility*. The prioritized version will have a similar behavior to the non-prioritized version. However, the prioritized version will never reject the input in contrast to the non-prioritized version that it completely accepts or rejects the input.

Definition 15 *Let $K \in \mathcal{K}$, let $\alpha \in \mathcal{L}$, let $TopBase(\cdot)$ be a maximum belief base function, and let $K^{\perp\!\!\!\perp}\alpha$ be the set of α-kernels of K. Let $\sigma_{\downarrow}$ a bottom incision function for K. The operator "$*$", called **Prioritized Revision Using Plausibility**, is defined as follow:*

$$K*(\alpha, A_i) = \begin{cases} K \cup \{(\alpha, A_i)\} & \text{if } \neg\alpha \notin Bel(K) \\ (K \setminus X) \cup \{(\alpha, A_i)\} & \text{if } \neg\alpha \in Bel(K) \end{cases}$$

where: $X = \{(\omega, A_j) : \omega \in Sen(\sigma_{\downarrow}(TopBase(K)^{\perp\!\!\!\perp}\neg\alpha))$ and $(\omega, A_j) \in K\}$.

Example 11 *Let us consider again Example 10. Next we will describe the behavior of the prioritized revision using plausibility operator when is applied over the belief base K_{A_1} with the tuple $(\neg\beta, A_5)$ step by step.*

- Step 1. *Obtain the minimal subsets that derive β from non-redundant belief base K.*
  $TopBase(K_{A_1}) = \{(\beta, A_1), (\alpha, A_2), (\alpha \rightarrow \beta, A_4), (\omega, A_1), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.
  $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta = \{H_a, H_b, H_c, H_d, H_e\}$ *where*

$H_a = \{(\beta, A_1)\}$, $H_b = \{(\alpha, A_2), (\alpha \rightarrow \beta, A_4)\}$,
$H_c = \{(\omega, A_1), (\omega \rightarrow \beta, A_4)\}$,
$H_d = \{(\alpha, A_2), (\alpha \rightarrow \delta, A_2), (\delta \rightarrow \beta, A_1)\}$,
$H_e = \{(\gamma, A_3), (\gamma \rightarrow \epsilon, A_2), (\epsilon \rightarrow \beta, A_4)\}$.

- Step 2. *Apply the bottom incision function "$\sigma_{\downarrow}$" to $TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta$ to obtain the set containing the lesser plausibility tuples from each β-kernel.*
  $\sigma_{\downarrow}(TopBase(K_{A_1})^{\perp\!\!\!\perp}\beta) = \{(\beta, A_1), (\alpha, A_2), (\omega, A_1), (\delta \rightarrow \beta, A_1), (\gamma, A_3), (\gamma \rightarrow \epsilon, A_2)\}$.
- Step 3. $K * (\neg\beta, A_5) = K \setminus \{(\omega, A_j) : \omega \in Sen(\sigma_{\downarrow}(TopBase(K)^{\perp\!\!\!\perp}\neg\alpha))$ *and* $(\omega, A_j) \in K\} \cup \{(\alpha, A_i)\}$.
  $K_{A_1} * (\neg\beta, A_5) = \{(\alpha \rightarrow \beta, A_2), (\alpha \rightarrow \beta, A_4), (\omega \rightarrow \beta, A_4), (\alpha \rightarrow \delta, A_2), (\epsilon \rightarrow \beta, A_2), (\epsilon \rightarrow \beta, A_3), (\epsilon \rightarrow \beta, A_4), (\neg\beta, A_5)\}$.

Retransmission of Information

As stated above, each agent $A \in \mathcal{A}gents$ will have a belief base $K_A \in \mathcal{K}$, where $\mathcal{K} = 2^{\mathcal{L}\times\mathcal{A}gents}$. Hence, the agents store each belief with an agent identifier in tuples. When an agent sends information to other agent, it sends tuples. Consider for example an agent set $\{A_1, A_2, A_3\} \subseteq \mathcal{A}gents$ where $A_1 \leq_{Co} A_2 \leq_{Co} A_3$. Suppose that $K_{A_1} = \{(\alpha, A_3)\}$ then if A_1 wants to send α to A_2, it has to send a tuple with an agent identifier. This identifier may be:

- the proper sender identifier (*e.g.*, A_1), or
- the agent identifier stored with the belief in the sender's base (*e.g.*, A_3).

Here, we adopt the latter option. That is, A_1 will send the tuple (α, A_3) to A_2. Thus, the receiver agent A_2 will know the source from where the sender A_1 has obtained the information.

From the receiver point of view, when it receives from A_1 the tuple (α, A_3) it may store:

- (α, A_3), *i.e.*, the agent identifier stored with the belief in the sender's base, or
- (α, A_i) where A_i is the agent identifier more credible (according to the credibility order among agents, Definition 1) that results of comparing the A_1 and A_3.

Here, we adopt the latter option. Thus, the agents will hold their beliefs with the most credible informant known. In this case, since $A_1 \leq_{Co} A_3$, the receiver agent A_2 will apply the new revision operator over the tuple (α, A_3).

Related Work

Two other approaches that cope with Multiple Sources Belief Revision (MSBR) are (Dragoni, Giorgini, and Puliti 1994) and (Cantwell 1998). Like us, both consider that the reliability of the source affects the credibility of incoming information, and this reliability is used to decide whether a received formula is accepted or rejected. However, these two approaches differs from ours in several issues.

In (Dragoni, Giorgini, and Puliti 1994; Dragoni, Giorgini, and Baffetti 1997) is considered that agents detect and store in tables the *nogoods*, which are the minimally inconsistent subsets of their knowledge bases. A *good* is a subset of the knowledge base such that: it is not inconsistent (it is not a superset of a *nogood*), and if augmented with whatever else assumption in knowledge base it becomes inconsistent. In contrast to our approach, they do not remove beliefs to avoid a contradiction, but, quite more generally, to choose which is the new preferred *good* among them in knowledge base. In our model, we obtain the kernel sets to cut some sentences, thus we broke the contradictions if it is necessary.

Like us, they propose to store additional information with each sentence. However, their tuples contain 5 elements: <Identifier, Sentence, OS, Source, Credibility>, where Origin Set (OS) records the assumption nodes upon which it really ultimately depends (as derived by the theorem prover). In contrast to them, in our model a tuple only store a sentence and a source, but a tuple does not store the credibility. That is, in our model the plausibility of a sentence is not explicitly stored with it, as (Dragoni, Giorgini, and Puliti 1994) does. Thus, when the plausibility of some sentence is needed the *plausibility function* should be applied. As is shown in Example 12, given a sentence α its plausibility depend on its proofs (α-$kernels$). Therefore, if one of the sentences of these proof changes, then the plausibility of α may change. Hence, if the credibility order is replaced, then the sentence plausibility may change without changing the belief base.

Example 12 *Consider a set $\mathcal{A}gents = \{A_1, A_2\}$ where the credibility order is $A_1 \leq_{Co} A_2$, $K_{A_1} = \{(\alpha, A_1), (\alpha \rightarrow \beta, A_2)\}$ and $K_{A_2} = \{(\alpha, A_2)\}$. By Definition 10, $Pl(\beta, K_{A_1}) = A_1$. Now, suppose that A_1 receives from A_2 the belief α. Now $K_{A_1} = \{(\alpha, A_1), (\alpha, A_2), (\alpha \rightarrow \beta, A_2)\}$ and A_1 has two derivations for β, hence $Pl(\beta, K_{A_1}) = A_2$. Observe that plausibility of β is increased.*

The communication policy that we have defined in "Retransmission of Information" Section differs from the one (Dragoni, Giorgini, and Puliti 1994) where the agents do not communicate the sources of the assumptions, but they present themselves as completely responsible for the knowledge they are passing on; receiving agents consider the sending ones as the sources of all the assumptions they are receiving from them.

In (Cantwell 1998), a *scenario* (set of incoming information) presented by a source is treated as a whole and not sentence by sentence, and therefore, it can be inconsistent. A relation of *trustworthiness* is introduced over sets of sources and not between single sources. Besides, if two sources give the same piece of information ϕ, and a single agent gives $\neg\phi$, then ϕ will be preferred, that is, the decision is based on majority. In his approach, the order in which the evidence is considered does not seem to be important. However, in our work, the order in which beliefs are considered is important: If an agent receives α and then receives $\neg\alpha$ and both have the same plausibility, then $\neg\alpha$ will be rejected.

Our work has some link with the idea of epistemic entrenchment (Gärdenfors and Makinson 1988; Rott 1992). Here the sentence plausibility is used in a similar way to epistemic entrenchment to update knowledge. However, there are some differences between these. For instance, in our work the order among sentences is based on the infor-

mants, whereas in (Gärdenfors and Makinson 1988) the order among sentences is implicitly defined over belief states represented by belief sets.

When we count with a multi-agent system that has only one agent, the new operator is very drastic. This is because in this scenario there is no order among agents. The same happens when all the agents of a multi-agent system have equal credibility. In these cases the bottom incision function does not have enough information to select sentences and it will erase all sentences in the α-kernels. This behavior is similar to *full meet revision* on belief bases (Hansson 1999). Nevertheless, when a multi-agent system has several agents with different credibilities and it is necessary to represent knowledge dynamics of the agents, plausibility seems to be a good criteria.

Conclusion and Future Work

In this paper we have presented a formalism for knowledge representation and consistency maintenance in a multi-agent system where deliberative agents can receive new information from others through communication. This type of Belief Revision is called *Belief Revision using information from Multiple Sources* (MSBR).

Similarly to Dragoni (Dragoni, Giorgini, and Puliti 1994) and Cantwell (Cantwell 1998), in our approach, informant agents can have different credibility and this credibility was used to decide which information prevails when a contradiction arises. However, there are differences with these authors, which were analyzed in the "Related Work" Section.

To decide whether to reject or accept a new belief, a comparison criterion among beliefs was defined. This criterion (called plausibility) is based on the credibility order among agents. Although we have first assumed that the credibility order among agents is fixed, as stated in Remark 4, this order may be replaced without affecting the behavior of the operator. As future work we propose to define an operator to revise the credibility order. This will allow us to represent changes over the credibility order.

As our model does not use (fixed) numbers or probabilities in a tuple to maintain the plausibility of a sentence, the plausibility depends on the sources of information of each one of the proofs' sentences. Thus, when the plausibility of some sentence α is needed, the *plausibility function* should be applied. Therefore, if one of the sentences of the proofs at α changes, then the plausibility of α may change.

A revision operator was proposed based on the plausibility that compares agents beliefs. This operator is based on the kernel contraction and incision function defined in (Hansson 1994), and satisfies the principles of Maintenance of Consistency (Dalal 1988), Minimal Changes and Non-Prioritization.

One of the limitations of the new operator is that a total order among agents is necessary. As future work, we want to relax this assumption and to consider a partial order among agents. We also plan to provide an axiomatic characterization for this operator.

References

Alchourrón, C., and Makinson, D. 1985. On the logic of theory change: Safe contraction. *St. Logica* 44:405–422.

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

Benferhat, S.; Dubois, D.; and Prade, H. 1998. From semantic to syntactic approaches to information combination in possibilistic logic. *Aggregation and Fusion of Imperfect Information. B. Bouchon-Meunier (Eds.), Physica-Verlag, Heidelberg* 12:141–161.

Cantwell, J. 1998. Resolving conflicting information. *Journal of Logic, Language and Information* 7(2):191–220.

Dalal, M. 1988. Investigations into a theory of knowledge base revision. *Proceedings of AAAI* 475–479.

Dragoni, A. F.; Giorgini, P.; and Baffetti, M. 1997. Distributed belief revision vs. belief revision in a multi-agent environment: First results of a simulation experiment. In *MAAMAW*, 45–62.

Dragoni, A.; Giorgini, P.; and Puliti, P. 1994. Distributed belief revision versus distributed truth maintenance.

Falappa, M. A.; Kern-Isberner, G.; and Simari, G. R. 2002. Explanations, belief revision and defeasible reasoning. *Artificial Intelligence* 141(1):1–28.

Fermé, E. L., and Hansson, S. O. 1999. Selective revision. *Studia Logica* 63(3):331–342.

Fuhrmann, A. 1991. Theory contraction through base contraction. *Journal of Philosophical Logic* 20(2):175–203.

Gärdenfors, P., and Makinson, D. 1988. Revisions of knowledge systems using epistemic entrenchment. In *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge Conference*, 83–95. Morgan Kaufmann.

Hansson, S. O. 1992. In defense of base contraction. *Synthese* 91(3):239–245.

Hansson, S. O. 1994. Kernel contraction. *Journal of Symbolic Logic* 59(3):845–859.

Hansson, S. O. 1997. Semi-revision. *Journal of Applied Non-Classical Logic* 151–175.

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

Kfir-Dahav, N. E., and Tennenholz, M. 1996. Multi-agent belief revision. In Shoham, Y., ed., *Theoretical Aspects of Rationality and Knowledge: Pro. of the Sixth Conf. (TARK 1996)*. San Francisco: Morgan Kaufmann. 175–196.

Liu, W., and Williams, M.-A. 1999. A framework for multi-agent belief revision, part i: The role of ontology. In *Australian Joint Conference on A. I.*, 168–179.

Liu, W., and Williams, M.-A. 2001. A framework for multi-agent belief revision. *Studia Logica* 67(2):291–312.

Makinson, D. 1997. Screened revision. *Theoria: Special Issue on Non-Prioritized Belief Revision*.

Rott, H. 1992. Preferential belief change using generalized epistemic entrenchment. *Journal of Logic, Language and Information* 1(1):45–78.

Action Theory Revision in Dynamic Logic

Ivan José Varzinczak

IRIT – Université de Toulouse
Toulouse, France
ivan.varzinczak@irit.fr

Meraka Institute
CSIR, Pretoria, South Africa
ivan.varzinczak@meraka.org.za

Abstract

Like any other logical theory, action theories in reasoning about actions may evolve, and thus need revision methods to adequately accommodate new information about the behavior of actions. Here we give a semantics that complies with minimal change for revising action theories stated in a version of PDL. We give algorithms that are proven correct w.r.t. the semantics for those theories that are modular.

Introduction

In logic-based approaches to reasoning about actions, theories are collections of statements of the form: "if *context*, then *effect* after *every execution* of *action*" (effect laws); and "if *precondition*, then *action executable*" (executability laws). For example, in Propositional Dynamic Logic (PDL) (Harel, Tiuryn, and Kozen 2000), one could have the law $(\neg p_1 \wedge \neg p_2) \rightarrow [a]p_1$, saying that in every context where $\neg p_1 \wedge \neg p_2$ is the case, after every execution of action a we get the effect p_1; and $(p_1 \vee \neg p_2) \rightarrow \langle a \rangle \top$, stating that $p_1 \vee \neg p_2$ is a sufficient condition for a's executability.

These are examples of what we call *action laws*, as they specify the behavior of the actions of a given domain. Besides that we can also have laws mentioning no action at all (static laws). They characterize the underlying structure of the world, i.e., its possible states. For instance, having $p_1 \rightarrow p_2$ as a static law would mean $p_1 \wedge \neg p_2$ is a forbidden state. Action theories will then be collections of laws, each of them seen as a global axiom in PDL.

Well, it may happen that such descriptions have to be revised due e.g. to new incoming information about the behavior of the world. In our example, we may learn that the only valid states are those satisfying $p_1 \wedge p_2$, or that action a has always $\neg p_2$ as outcome in $\neg p_2$-contexts, or even that p_1 is enough to guarantee a's executability. Here we are interested in this kind of theory change.

The contributions of the present work are as follows:

- What is the semantics of revising an action theory $\mathcal{T}$ by a law Φ? How to get minimal change, i.e., how to keep as much knowledge about other laws as possible?
- How to syntactically revise an action theory so that its result corresponds to the intended semantics?

Here we answer these questions.

Logical Preliminaries

Action Theories in Dynamic Logic

Our base formalism is PDL without the $*$ operator. Let $\mathfrak{Act} = \{a_1, a_2, \ldots\}$ be the set of *atomic actions* of a domain. To each a there is associated a modal operator $[a]$. We suppose our multimodal logic is independently axiomatized, i.e., the logic is a fusion and there is no interaction between the modal operators (Kracht and Wolter 1991).

$\mathfrak{Prop} = \{p_1, p_2, \ldots\}$ denotes the set of all *propositional constants* or *atoms*. The set of literals is $\mathfrak{Lit} = \{\ell_1, \ell_2, \ldots\}$, where each ℓ_i is either p or $\neg p$, for some $p \in \mathfrak{Prop}$. In case $\ell = \neg p$, we identify $\neg\ell$ with p. By $|\ell|$ we will denote the atom in literal ℓ.

By $\varphi, \psi, \ldots$ we denote *Boolean formulas*, examples of which are $p_1 \rightarrow p_2$ and $\neg p_1 \oplus p_2$. $\mathfrak{Fml}$ is the set of all Boolean formulas. A propositional valuation v is a *maximally consistent* set of literals. We denote $v \Vdash \varphi$ the fact that v satisfies φ. $val(\varphi)$ is the set of all valuations satisfying φ. $\models_{\mathsf{CPL}}$ denotes the classical consequence relation.

With $IP(\varphi)$ we denote the set of *prime implicants* (Quine 1952) of φ. By π we denote a prime implicant, and $atm(\pi)$ is the set of atoms occurring in π. For given ℓ and π, $\ell \in \pi$ abbreviates 'ℓ is a literal of π'.

We denote complex formulas (with modal operators) by $\Phi, \Psi, \ldots$ $\langle a \rangle$ is the dual operator of $[a]$, ($\langle a \rangle \Phi =_{\text{def}} \neg[a]\neg\Phi$). An example of a complex formula is $(p_1 \wedge (p_2 \vee \neg p_3)) \rightarrow [a](\neg p_1 \vee p_3)$.

A PDL-*model* is a tuple $\mathscr{M} = \langle W, R \rangle$ where W is a set of valuations, and R maps action constants a to accessibility relations $R_a \subseteq W \times W$. Given a model $\mathscr{M}$, $\models^{\mathscr{M}}_{w} p$ (p is true at world w of model $\mathscr{M}$) if $w \Vdash p$; $\models^{\mathscr{M}}_{w} [a]\Phi$ if $\models^{\mathscr{M}}_{w'} \Phi$ for every w' s.t. $(w, w') \in R_a$; truth conditions for the other connectives are as usual. By $\mathcal{M}$ we will denote a set of PDL-models. $\mathscr{M}$ is a model of Φ (noted $\models^{\mathscr{M}} \Phi$) if and only if $\models^{\mathscr{M}}_{w} \Phi$ for all $w \in W$. $\mathscr{M}$ is a model of a set of formulas Σ (noted $\models^{\mathscr{M}} \Sigma$) if and only if $\models^{\mathscr{M}} \Phi$ for every $\Phi \in \Sigma$. Φ is a *consequence of*

the global axioms Σ in all PDL-models (noted $\Sigma \models_{\mathsf{PDL}} \Phi$) if and only if for every $\mathscr{M}$, if $\models^{\mathscr{M}} \Sigma$, then $\models^{\mathscr{M}} \Phi$.

With PDL we can state laws describing the behavior of actions. Following the tradition in the reasoning about actions community, we here distinguish three types of them.

Static Laws A *static law* is a formula $\varphi \in \mathfrak{Fml}$. It characterizes the possible states of the world. The set of all static laws of a domain is denoted by $\mathcal{S}$.

Effect Laws An *effect law for a* is of the form $\varphi \rightarrow [a]\psi$, where $\varphi, \psi \in \mathfrak{Fml}$. Effect laws relate an action to its effects, which can be conditional. The consequent ψ is the effect which always obtains when *a* is executed in a state where the antecedent φ holds. If *a* is a nondeterministic action, then ψ is typically a disjunction. If ψ is inconsistent we have a special kind of effect law that we call an *inexecutability law*. For example, $(\neg p_1 \wedge p_2) \rightarrow [a]\bot$ says that *a* cannot be executed (there is no *a*-transition) in $\neg p_1 \wedge p_2$-contexts. The set of effect laws of a domain is denoted by $\mathcal{E}$.

Executability Laws An *executability law for a* has the form $\varphi \rightarrow \langle a\rangle\top$, with $\varphi \in \mathfrak{Fml}$. It stipulates the context in which *a* is guaranteed to be executable. (In PDL, the operator $\langle a\rangle$ is used to express executability, $\langle a\rangle\top$ thus reads "*a*'s execution is possible".) The set of all executability laws of a domain is denoted by $\mathcal{X}$.

Action Theories $\mathcal{T} = \mathcal{S} \cup \mathcal{E} \cup \mathcal{X}$ is an *action theory*.

Given an action *a*, $\mathcal{E}_a$ (resp. $\mathcal{X}_a$) will denote the set of only those effect (resp. executability) laws about *a*. For the sake of clarity, we here abstract from the frame and ramification problems, and assume $\mathcal{T}$ contains all frame axioms (cf. (Herzig, Perrussel, and Varzinczak 2006) for a contraction approach within a solution to the frame problem).

Elementary Atoms and Prime Valuations

Given $\varphi \in \mathfrak{Fml}$, $E(\varphi)$ denotes the elementary atoms *actually* occurring in φ. For example, $E(\neg p_1 \wedge (\neg p_1 \vee p_2)) = \{p_1, p_2\}$. An atom p is *essential* to φ if and only if $p \in E(\varphi')$ for every φ' such that $\models_{\mathsf{CPL}} \varphi \leftrightarrow \varphi'$. For instance, p_1 is essential to $\neg p_1 \wedge (\neg p_1 \vee p_2)$. $E!(\varphi)$ will denote the essential atoms of φ. (If φ is not contingent, i.e., it is a tautology or a contradiction, then $E!(\varphi) = \emptyset$.)

For $\varphi \in \mathfrak{Fml}$, $\varphi*$ is the set of all $\varphi' \in \mathfrak{Fml}$ such that $\varphi \models_{\mathsf{CPL}} \varphi'$ and $E(\varphi') \subseteq E!(\varphi)$. For instance, $p_1 \vee p_2 \notin p_1*$, as $p_1 \models_{\mathsf{CPL}} p_1 \vee p_2$ but $E(p_1 \vee p_2) \not\subseteq E!(p_1)$. Moreover $E(\varphi*) = E!(\varphi*)$, and whenever $\models_{\mathsf{CPL}} \varphi \leftrightarrow \varphi'$, $E!(\varphi) = E!(\varphi')$ and also $\varphi* = \varphi'*$.

Theorem 1 (Least atom-set theorem (Parikh 1999)) *$\models_{\mathsf{CPL}} \varphi \leftrightarrow \bigwedge \varphi*$, and $E(\varphi*) \subseteq E(\varphi')$ for every φ' s.t. $\models_{\mathsf{CPL}} \varphi \leftrightarrow \varphi'$.*

Thus for each $\varphi \in \mathfrak{Fml}$ there is a unique least set of elementary atoms such that φ may equivalently be expressed using only atoms from that set.[1]

[1] The dual notion (redundant atoms) is addressed in (Herzig and Rifi 1999), with similar purposes.

Given a valuation v, $v' \subseteq v$ is a *subvaluation*. For W a set of valuations, a subvaluation v' *satisfies* $\varphi \in \mathfrak{Fml}$ modulo W (noted $v' \Vdash_W \varphi$) if and only if $v \Vdash \varphi$ for all $v \in W$ such that $v' \subseteq v$. A subvaluation v *essentially satisfies* φ (modulo W), noted $v \Vdash^!_W \varphi$, if and only if $v \Vdash_W \varphi$ and $\{|\ell| : \ell \in v\} \subseteq E!(\varphi)$. If $v \Vdash^!_W \varphi$, we call v an *essential subvaluation* of φ (modulo W).

Definition 1 *Let $\varphi \in \mathfrak{Fml}$ and W be a set of valuations. v is a* prime subvaluation *of φ (modulo W) if and only if $v \Vdash^!_W \varphi$ and there is no $v' \subseteq v$ s.t. $v' \Vdash^!_W \varphi$.*

Prime subvaluations of a formula φ are the weakest states of truth in which φ is true. They are just another way of seeing prime implicants of φ. By $base(\varphi, W)$ we denote the set of all prime subvaluations of φ modulo W.

Theorem 2 *Let $\varphi \in \mathfrak{Fml}$ and W be a set of valuations. Then for all $w \in W$, $w \Vdash \varphi$ if and only if $w \Vdash \bigvee_{v \in base(\varphi,W)} \bigwedge_{\ell \in v} \ell$.*

Closeness Between Models

When revising a model, we will perform a change in its structure. Because there can be several different ways of modifying a model (not all of them minimal), we need a notion of distance between models to identify those that are closest to the original one.

As we are going to see in more depth in the sequel, changing a model amounts to modifying its possible worlds or its accessibility relation. Hence, the distance between two PDL-models will depend upon the distance between their sets of worlds and accessibility relations. These here will be based on the *symmetric difference* between sets, defined as $X \dot{-} Y = (X \setminus Y) \cup (Y \setminus X)$.

Definition 2 *Let $\mathscr{M} = \langle W, R\rangle$ be a model. $\mathscr{M}' = \langle W', R'\rangle$ is* as close to *$\mathscr{M}$ as $\mathscr{M}'' = \langle W'', R''\rangle$, noted $\mathscr{M}' \preceq_{\mathscr{M}} \mathscr{M}''$, if and only if*

- *either $W \dot{-} W' \subseteq W \dot{-} W''$*
- *or $W \dot{-} W' = W \dot{-} W''$ and $R \dot{-} R' \subseteq R \dot{-} R''$*

(Notice that other distance notions are also possible, like e.g. considering the *cardinality* of symmetric differences.)

Semantics of Revision

Contrary to action theory contraction (Varzinczak 2008a), where we want the negation of some law to become *satisfiable*, in revision we want to make a new law *valid*. This means that one has to eliminate all cases satisfying its negation. This depicts the duality between revision and contraction: whereas in the latter one invalidates a formula by making its negation satisfiable, in the former one makes a formula valid by forcing its negation to be unsatisfiable prior to adding the new law to the theory.

The idea behind our semantics is as follows: we initially have a set of models $\mathcal{M}$ in which a given formula Φ is (potentially) not valid, i.e., Φ is (possibly) not true in every model in $\mathcal{M}$. In the result we want to have only models of Φ. Adding Φ-models to $\mathcal{M}$ is of no help. Moreover, adding

models makes us to lose laws: the corresponding resulting theory would be more liberal.

One solution amounts to deleting from $\mathcal{M}$ those models that are not Φ-models. Of course removing only some of them does not solve the problem, we must delete every such a model. By doing that, all resulting models will be models of Φ. (This corresponds to *theory expansion*, when the resulting theory is satisfiable.) However, if $\mathcal{M}$ contains no model of Φ, we will end up with $\emptyset$. Consequence: the resulting theory is inconsistent. (This is the main revision problem.) In this case the solution is to *substitute* each model $\mathscr{M}$ in $\mathcal{M}$ by its *nearest modification* $\mathscr{M}^*_\Phi$ that makes Φ true. This lets us to keep as close as possible to the original models we had. But, what if for one model in $\mathcal{M}$ there are several minimal (incomparable) modifications of it validating Φ? In that case we shall consider all of them. The result will also be a *list of models* $\mathcal{M}^*_\Phi$, all being models of Φ.

Before defining revision of sets of models, we present what modifications of (individual) models are.

Revising a Model by a Static Law

Consider the model depicted in Figure 1, and suppose we want to revise it by the Boolean formula $p_1 \vee p_2$, i.e., we want such a formula to be a static law.

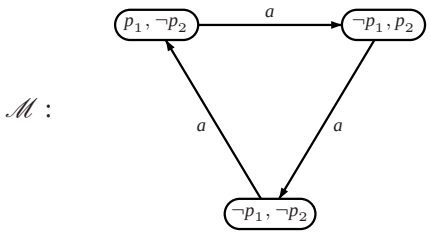


Figure 1: A model where $\neg p_1 \wedge \neg p_2$ is satisfiable.

In such a model, we do not want the formula $\neg p_1 \wedge \neg p_2$ to be satisfiable, so the first step is to remove all worlds in which it is true. The second step is to guarantee that all the remaining worlds satisfy the new law. Such an issue has been largely addressed in the literature on propositional belief base revision and update (Gärdenfors 1988; Winslett 1988; Katsuno and Mendelzon 1992; Herzig and Rifi 1999). Here we can achieve that with a semantics similar to that of classical operators: basically one shall change the set of possible valuations, by removing or adding worlds.

The delicate point in removing worlds is that we may lose some executability laws: in the example, removing $\{\neg p_1, \neg p_2\}$ also removes $p_2 \rightarrow \langle a \rangle \top$. From a semantic point of view, this is intuitive: if the state of the world to which we could move is no longer possible, then we do not have a transition to that state anymore. Hence, if that transition was the only one we had, it is natural to lose it.

Similarly, one could ask what to do with the accessibility relation if new worlds are added (when expansion is not possible): shall new arrows leave/arrive at the new world? If no arrow leaves the new added world, we may lose an executability law. If some arrow leaves it, we may lose an effect law, the same holding if we add an arrow pointing to the new world. If no arrow arrives at this new world, what about the intuition? Do we want to have an unreachable state?

All this discussion shows how drastic a change in the static laws may be: it is a change in the underlying structure (possible states) of the world! Changing it may have as consequence the loss of an effect law or an executability law.

The tradition in the reasoning about actions community says that executability laws are, in general, more difficult to state than effect laws, and hence are more likely to be incorrect. By adding no arrow to the resulting model we here comply with that and postpone correction of executability laws, if needed (cf. (Herzig, Perrussel, and Varzinczak 2006; Varzinczak 2008a)).

The semantics for revision of one model by a static law is as follows:

Definition 3 *Let* $\mathscr{M} = \langle W, R \rangle$. $\mathscr{M}' = \langle W', R' \rangle \in \mathscr{M}^*_\varphi$ *if and only if:*

- $W' = (W \setminus val(\neg\varphi)) \cup val(\varphi)$
- $R' \subseteq R$

Clearly $\models^{\mathscr{M}'} \varphi$ for each $\mathscr{M}' \in \mathscr{M}^*_\varphi$. The minimal models resulting from revising a model $\mathscr{M}$ by φ are those closest to $\mathscr{M}$ w.r.t. $\preceq_{\mathscr{M}}$:

Definition 4 $revise(\mathscr{M}, \varphi) = \bigcup \min\{\mathscr{M}^*_\varphi, \preceq_{\mathscr{M}}\}$

Revising a Model by an Effect Law

Let our language now have three atoms and consider the model $\mathscr{M}$ in Figure 2.

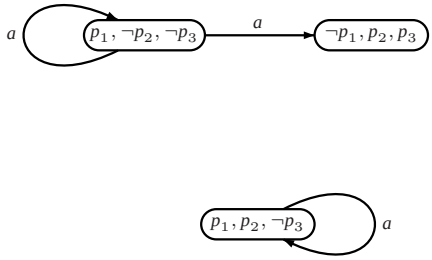


Figure 2: A model where $p_1 \wedge \langle a \rangle p_2$ is satisfiable.

(Notice that $\models^{\mathscr{M}} p_2 \rightarrow p_1 \oplus p_3$.) Suppose we want to revise $\mathscr{M}$ by $p_1 \rightarrow [a]\neg p_2$. This means that we should guarantee the formula $p_1 \wedge \langle a \rangle p_2$ is satisfiable in none of its worlds. To do that, we have to look at the worlds satisfying it (if any) and either make p_1 false, or make $\langle a \rangle p_2$ false by removing a-arrows leading to p_2-worlds.

In our example, the worlds $\{p_1, \neg p_2, \neg p_3\}$ and $\{p_1, p_2, \neg p_3\}$ satisfy $p_1 \wedge \langle a \rangle p_2$ and both have to change. Flipping p_1 would do the job but also has as consequence the loss of a static law: we would violate $p_2 \rightarrow p_1 \oplus p_3$. Here we think that changing action laws should not have as side effect a change in the static laws. Given their special status, these should change only if explicitly required (see above). In this case, each world satisfying $p_1 \wedge \langle a \rangle p_2$ has to be changed so that $\langle a \rangle p_2$ is no longer true in it. In our example, we should remove the arrows $(\{p_1, \neg p_2, \neg p_3\}, \{\neg p_1, p_2, p_3\})$ and $(\{p_1, p_2, \neg p_3\}, \{p_1, p_2, \neg p_3\})$.

The semantics of one model revision for the case of a new effect law is:

Definition 5 *Let $\mathscr{M} = \langle W, R\rangle$. $\mathscr{M}' = \langle W', R'\rangle \in \mathscr{M}^*_{\varphi\to[a]\psi}$ if and only if:*

- $W' = W$
- $R' \subseteq R$
- *If* $(w, w') \in R \setminus R'$, *then* $\models^{\mathscr{M}}_{w} \varphi$ *and* $\models^{\mathscr{M}}_{w'} \neg\psi$
- $\models^{\mathscr{M}'} \varphi \to [a]\psi$

The minimal models resulting from the revision of a model $\mathscr{M}$ by a new effect law are those that are closest to $\mathscr{M}$ w.r.t. $\preceq_{\mathscr{M}}$:

Definition 6 *Let $\mathscr{M}$ be a model and $\varphi \to [a]\psi$ an effect law. Then revise$(\mathscr{M}, \varphi \to [a]\psi) = \bigcup \min\{\mathscr{M}^*_{\varphi\to[a]\psi}, \preceq_{\mathscr{M}}\}$.*

Revising a Model by an Executability Law

Let the model depicted in Figure 3 and suppose we want to revise it by the new executability law $p_1 \to \langle a\rangle\top$.

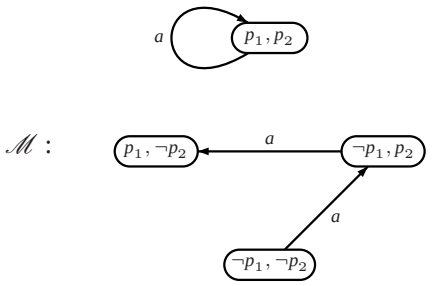


Figure 3: A model where $p_1 \wedge [a]\bot$ is satisfiable.

Observe that $\neg(p_1 \to \langle a\rangle\top)$ is satisfiable in $\mathscr{M}$, hence we must throw $p_1 \wedge [a]\bot$ away to ensure the new formula is true. To remove $p_1 \wedge [a]\bot$ we have to look at all worlds satisfying it and modify $\mathscr{M}$ so that they no longer satisfy the formula. Given world $\{p_1, \neg p_2\}$, we have two options: change the interpretation of p_1 or add a new arrow leaving this world. A question that raises is 'what choice is more drastic: change a world or an arrow'? Again, here we think that changing the world's content (the valuation) is more drastic, as the existence of such a world was foreseen by some static law and is hence assumed to be as it is, unless we have information supporting the contrary (see above). Thus we shall add a new *a*-arrow from $\{p_1, \neg p_2\}$. Having agreed on that, the issue now is: to which world should the new arrow point? Four options show up: point the arrow to $\{p_1, p_2\}$, $\{\neg p_1, p_2\}$, $\{\neg p_1, \neg p_2\}$ or $\{p_1, \neg p_2\}$ itself. The resulting model is such that the unwanted formula is unsatisfiable and $p_1 \to \langle a\rangle\top$ holds in all its worlds.

Whereas all these options make the new law true in the resulting model, not all of them comply with minimal change. To witness, putting an *a*-arrow from $\{p_1, \neg p_2\}$ to $\{\neg p_1, \neg p_2\}$ or $\{p_1, \neg p_2\}$ makes us lose the effect law $\neg p_2 \to [a]p_2$; and pointing it to $\{\neg p_1, p_2\}$ also deletes from the model $p_1 \to [a]p_1$. Note that these laws are preserved if we point the arrow to $\{p_1, p_2\}$. What would support the choice for the latter?

When pointing a new arrow leaving a world w we want to preserve as many effects as we had before doing so. To achieve this, it is enough to preserve old effects only in w (because the remaining structure of the model remains unchanged after adding *this* new arrow). The operation we must carry out is to observe what is true in w and in the candidate target world w':

- What changes from w to w' ($w' \setminus w$) must be what is obliged to do so.
- What does not change from w to w' ($w \cap w'$) must be what is either obliged or allowed to do so.

This means that every change outside what is forced to change is not an intended one. In our example, when putting the *a*-arrow from $\{p_1, \neg p_2\}$ to $\{\neg p_1, p_2\}$, $\neg p_1$ becomes a possible effect of *a*. As far as $\neg p_1$ is never caused by *a*, there is no justification for having it in a target world of $\{p_1, \neg p_2\}$. Similarly, we want the literals preserved in the target world to be *at most* those that either are consequences of some effect or are usually preserved in that context. Every preservation outside those may make us lose some law. For instance, when putting the new *a*-arrow from $\{p_1, \neg p_2\}$ to $\{\neg p_1, \neg p_2\}$, $\neg p_2$ is preserved. Because $\neg p_2$ is not a necessary effect of *a* and is moreover never preserved across *a*'s execution (in $\mathscr{M}$), there is no reason to preserve it in this new *a*-transition.

This looks like prime implicants, and that is where prime subvaluations play their role: the worlds to which the new arrow shall point are those whose difference w.r.t. the departing world are literals that are relevant, and whose similarity w.r.t. it are literals that we know do not change.

Before giving a formal definition for that, we need to consider two important issues: First, when checking satisfaction of these two conditions, looking just at what is true in the model $\mathscr{M}$ we want to modify is not enough. It can be a model in which a contingent, i.e., not true in all models formula is true. Hence we shall consider all the models in $\mathcal{M}$. Second, if *a* is never executable in w, i.e., $R_a(w) = \emptyset$ for every $\mathscr{M} = \langle W, R\rangle \in \mathcal{M}$, then lots of effects for *a* trivially hold in w, and then not all of them should be taken into account in deciding what has to be changed or preserved. In this case, one should instead look at the effects that hold for those worlds w such that $R_a(w) \neq \emptyset$ (because everything that holds in these worlds also holds trivially in those worlds with no transition by *a*).

Definition 7 *Let $\mathscr{M} = \langle W, R\rangle$ be a model, $w, w' \in W$, $\mathcal{M}$ a set of models such that $\mathscr{M} \in \mathcal{M}$, and $\varphi \to \langle a\rangle\top$ an executability law. Then w' is a* relevant target world of w *w.r.t. $\varphi \to \langle a\rangle\top$ for $\mathscr{M}$ in $\mathcal{M}$ if and only if:*

- $\models^{\mathscr{M}}_{w} \varphi$
- *If there is $\mathscr{M}' = \langle W', R'\rangle \in \mathcal{M}$ such that $R'_a(w) \neq \emptyset$:*
  - *for all $\ell \in w' \setminus w$, there is $\psi' \in \mathfrak{Fml}$ s.t. there is $v' \in$ base(ψ', W) s.t. $v' \subseteq w'$, $\ell \in v'$, and for every $\mathscr{M}_i \in \mathcal{M}$, $\models^{\mathscr{M}_i}_{w} [a]\psi'$*
  - *for all $\ell \in w \cap w'$, either there is $\psi' \in \mathfrak{Fml}$ s.t. there is $v' \in$ base(ψ', W) s.t. $v' \subseteq w'$, $\ell \in v'$, and for all $\mathscr{M}_i \in \mathcal{M}$, $\models^{\mathscr{M}_i}_{w} [a]\psi'$; or there is $\mathscr{M}_i \in \mathcal{M}$ s.t. $\not\models^{\mathscr{M}_i}_{w} [a]\neg\ell$*

- *If* $R'_a(w) = \emptyset$ *for every* $\mathscr{M}' = \langle W', R' \rangle \in \mathcal{M}$*:*
  - *for all* $\ell \in w' \setminus w$*, there is* $\mathscr{M}_i = \langle W_i, R_i \rangle \in \mathcal{M}$ *s.t. there is* $u, v \in W_i$ *s.t.* $(u, v) \in R_{ia}$ *and* $\ell \in v \setminus u$
  - *for all* $\ell \in w \cap w'$*, there is* $\mathscr{M}_i = \langle W_i, R_i \rangle \in \mathcal{M}$ *s.t. there is* $u, v \in W_i$ *s.t.* $(u, v) \in R_{ia}$ *and* $\ell \in u \cap v$*, or for all* $\mathscr{M}_i = \langle W_i, R_i \rangle \in \mathcal{M}$*, if* $(u, v) \in R_{ia}$*, then* $\neg\ell \notin v \setminus u$

By $RelTgt(w, \varphi \to \langle a \rangle \top, \mathscr{M}, \mathcal{M})$ *we denote the set of all relevant target worlds of* w *w.r.t.* $\varphi \to \langle a \rangle \top$ *for* $\mathscr{M}$ *in* $\mathcal{M}$*.*

The semantics of one model revision by a new executability law is given by:

Definition 8 *Let* $\mathscr{M} = \langle W, R \rangle$*.* $\mathscr{M}' = \langle W', R' \rangle \in \mathscr{M}^*_{\varphi \to \langle a \rangle \top}$ *if and only if:*

- $W' = W$
- $R \subseteq R'$
- *If* $(w, w') \in R' \setminus R$*, then* $w' \in RelTgt(w, \varphi \to \langle a \rangle \top, \mathscr{M}, \mathcal{M})$
- $\models^{\mathscr{M}'} \varphi \to \langle a \rangle \top$

The minimal models resulting from revising a model $\mathscr{M}$ by a new executability law are those closest to $\mathscr{M}$ w.r.t. $\preceq_{\mathscr{M}}$:

Definition 9 *Let* $\mathscr{M}$ *be a model and* $\varphi \to \langle a \rangle \top$ *be an executability law. Then* $revise(\mathscr{M}, \varphi \to \langle a \rangle \top) = \bigcup \min\{\mathscr{M}^*_{\varphi \to \langle a \rangle \top}, \preceq_{\mathscr{M}}\}$*.*

Revising Sets of Models

Now we are ready to define revision of a set of models $\mathcal{M}$ by a new law Φ:

Definition 10 *Let* $\mathcal{M}$ *be a set of models and* Φ *a law. Then*

$$\mathcal{M}^*_\Phi = \begin{cases} \mathcal{M} \setminus \{\mathscr{M} : \not\models^{\mathscr{M}} \Phi\}, & \text{if there is } \mathscr{M} \in \mathcal{M} \text{ s.t. } \models^{\mathscr{M}} \Phi \\ \bigcup_{\mathscr{M} \in \mathcal{M}} revise(\mathscr{M}, \Phi), & \text{otherwise} \end{cases}$$

Observe that Definition 10 comprises both *expansion* and *revision*: in the first one, simple addition of the new law gives a satisfiable theory; in the latter a deeper change is needed to get rid of inconsistency.

Syntactic Operators for Revision

We now turn our attention to the syntactical counterpart of revision. Suppose we have an action theory $\mathcal{T}$ and a law Φ we want to revise $\mathcal{T}$ with. If $\mathcal{T} \cup \{\Phi\}$ is satisfiable, adding Φ to $\mathcal{T}$ (expansion) will do the job. Otherwise, if $\mathcal{T} \cup \{\Phi\} \models_{\mathsf{PDL}} \bot$, then we have to modify the laws in $\mathcal{T}$ to accommodate with the new incoming law (proper revision). Our endeavor here is to perform minimal change at the syntactical level. By $\mathcal{T}^*_\Phi$ we denote the result of revising $\mathcal{T}$ with Φ.

Revision by a Static Law

Looking at the semantics of revision by Boolean formulas, we see that revising an action theory by a new static law may conflict with the executability laws: some of them may be lost and thus have to be changed as well. The approach here is to preserve as many executabilities as we can in the old possible states. To do that, we look at each possible valuation that is common to the new $\mathcal{S}$ and the old one. Every time an executability used to hold in that state and no inexecutability holds there in the new theory, we make the action executable in such a context. For those contexts not allowed by the old $\mathcal{S}$, we make a inexecutable (cf. the semantics). Algorithm 1 deals with that ($\mathcal{S} \star \varphi$ denotes the classical revision of $\mathcal{S}$ by φ using any standard method from the literature (Winslett 1988; Katsuno and Mendelzon 1992; Herzig and Rifi 1999)).

Algorithm 1 Revision by a static law

input: $\mathcal{T}, \varphi$
output: $\mathcal{T}^*_\varphi$
  if $\mathcal{T} \cup \{\varphi\} \not\models_{\mathsf{PDL}} \bot$ **then**
    $\mathcal{T}^*_\varphi := \mathcal{T} \cup \{\varphi\}$
  else
    $\mathcal{S}' := \mathcal{S} \star \varphi$, $\mathcal{E}' := \mathcal{E}$, $\mathcal{X}' := \emptyset$
    for all $\pi \in IP(\mathcal{S}')$ **do**
      for all $A \subseteq \overline{atm(\pi)}$ **do**
        $\varphi_A := \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \in A}} p_i \wedge \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \notin A}} \neg p_i$
        if $\mathcal{S}' \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \to \bot$ **then**
          if $\mathcal{S} \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \to \bot$ **then**
            if $\mathcal{T} \models_{\mathsf{PDL}} (\pi \wedge \varphi_A) \to \langle a \rangle \top$ **and** $\mathcal{S}', \mathcal{E}', \mathcal{X} \not\models_{\mathsf{PDL}} \neg(\pi \wedge \varphi_A)$ **then**
              $\mathcal{X}_a' := \{(\varphi_i \wedge \pi \wedge \varphi_A) \to \langle a \rangle \top : \varphi_i \to \langle a \rangle \top \in \mathcal{X}_a\}$
            else
              $\mathcal{E}' := \mathcal{E}' \cup \{(\pi \wedge \varphi_A) \to [a] \bot\}$
    $\mathcal{T}^*_\varphi := \mathcal{S}' \cup \mathcal{E}' \cup \mathcal{X}'$

Revision by an Effect Law

When revising a theory by a new effect law $\varphi \to [a]\psi$, we want to eliminate all possible executions of a leading to $\neg\psi$-states. To achieve that, we look at all φ-contexts and every time a transition to some $\neg\psi$-context is not always the case, i.e., $\mathcal{T} \not\models_{\mathsf{PDL}} \varphi \to \langle a \rangle \neg\psi$, we can safely force $[a]\psi$ for that context. On the other hand, if in such a context there is always an execution of a to $\neg\psi$, then we should strengthen the executability laws to make room for the new effect in that context we want to add. Algorithm 2 below does the job.

Revision by an Executability Law

Revising a theory by a new executability law will have as immediate consequence a change in the set of effect laws: all those laws preventing the execution of a shall be weakened. Besides that, in order to comply with minimal change, we shall ensure that in all models of the resulting theory there will be at most *one* transition by a from those worlds in which $\mathcal{T}$ precluded a's execution.

Let $\mathcal{E}_a^{\varphi,\bot}$ denote a minimum subset of $\mathcal{E}_a$ such that $\mathcal{S}, \mathcal{E}_a^{\varphi,\bot} \models_{\mathsf{PDL}} \varphi \to [a]\bot$. In the case the theory is modular (Herzig and Varzinczak 2005) (see further), interpolation guarantees that this set always exists. Moreover, note that there can be more than one such a set, in which case we denote them $(\mathcal{E}_a^{\varphi,\bot})_1, \ldots, (\mathcal{E}_a^{\varphi,\bot})_n$. Let

$$\mathcal{E}_a^- = \bigcup_{1 \le i \le n} (\mathcal{E}_a^{\varphi,\bot})_i$$

Algorithm 2 Revision by an effect law

input: $\mathcal{T}, \varphi \rightarrow [a]\psi$
output: $\mathcal{T}^*_{\varphi\rightarrow[a]\psi}$
  if $\mathcal{T} \cup \{\varphi \rightarrow [a]\psi\} \not\models_{\mathsf{PDL}} \bot$ **then**
    $\mathcal{T}^*_{\varphi\rightarrow[a]\psi} := \mathcal{T} \cup \{\varphi \rightarrow [a]\psi\}$
  else
    $\mathcal{T}' := \mathcal{T}$
    for all $\pi \in IP(\mathcal{S} \wedge \varphi)$ **do**
      for all $A \subseteq \overline{atm(\pi)}$ **do**
        $\varphi_A := \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \in A}} p_i \wedge \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \notin A}} \neg p_i$
        if $\mathcal{S} \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$ **then**
          for all $\pi' \in IP(\mathcal{S} \wedge \neg\psi)$ **do**
            if $\mathcal{T}' \models_{\mathsf{PDL}} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\pi'$ **then**

$$\mathcal{T}' := \begin{array}{l}(\mathcal{T}' \setminus \mathcal{X}'_a) \cup \\ \{(\varphi_i \wedge \neg(\pi \wedge \varphi_A)) \rightarrow \langle a\rangle\top : \\ \varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}'_a\}\end{array}$$

            $\mathcal{T}' := \mathcal{T}' \cup \{(\pi \wedge \varphi_A) \rightarrow [a]\psi\}$
            if $\mathcal{T}' \not\models_{\mathsf{PDL}} (\pi \wedge \varphi_A) \rightarrow [a]\bot$ **then**
              $\mathcal{T}' := \mathcal{T}' \cup \{(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow \langle a\rangle\top : \varphi_i \rightarrow \langle a\rangle\top \in \mathcal{T}\}$
    $\mathcal{T}^*_{\varphi\rightarrow[a]\psi} := \mathcal{T}'$

Algorithm 3 Revision by an executability law

input: $\mathcal{T}, \varphi \rightarrow \langle a\rangle\top$
output: $\mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top}$
  if $\mathcal{T} \cup \{\varphi \rightarrow \langle a\rangle\top\} \not\models_{\mathsf{PDL}} \bot$ **then**
    $\mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top} := \mathcal{T} \cup \{\varphi \rightarrow \langle a\rangle\top\}$
  else
    $\mathcal{T}' := \mathcal{T}$
    for all $\pi \in IP(\mathcal{S} \wedge \varphi)$ **do**
      for all $A \subseteq \overline{atm(\pi)}$ **do**
        $\varphi_A := \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \in A}} p_i \wedge \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \notin A}} \neg p_i$
        if $\mathcal{S} \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$ **then**
          if $\mathcal{T}' \models_{\mathsf{PDL}} (\pi \wedge \varphi_A) \rightarrow [a]\bot$ **then**

$$\mathcal{T}' := \begin{array}{l}(\mathcal{T}' \setminus \mathcal{E}'^{-}_a) \cup \\ \{(\varphi_i \wedge \neg(\pi \wedge \varphi_A)) \rightarrow [a]\psi_i : \\ \varphi_i \rightarrow [a]\psi_i \in \mathcal{E}'^{-}_a\} \cup \\ \{(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow [a] \bigoplus_{\substack{\pi' \in IP(\mathcal{S}) \\ A' \subseteq \overline{atm(\pi')}}} (\pi' \wedge \varphi_{A'}) : \\ \varphi_i \rightarrow [a]\psi_i \in \mathcal{E}'^{-}_a\}\end{array}$$

          for all $L \subseteq \mathfrak{Lit}$ **do**
            if $\mathcal{S} \models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bigwedge_{\ell \in L} \ell$ **then**
              for all $\ell \in L$ **do**
                if $\mathcal{T} \models_{\mathsf{PDL}} \ell \rightarrow [a]\bot$ **or** ($\mathcal{T} \not\models_{\mathsf{PDL}} \ell \rightarrow [a]\neg\ell$ **and** $\mathcal{T} \models_{\mathsf{PDL}} \ell \rightarrow [a]\ell$) **then**
                  $\mathcal{T}' := \mathcal{T}' \cup \{(\pi \wedge \varphi_A \wedge \ell) \rightarrow [a]\ell\}$
          $\mathcal{T}' := \mathcal{T}' \cup \{(\pi \wedge \varphi_A) \rightarrow \langle a\rangle\top\}$
    $\mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top} := \mathcal{T}'$

The effect laws in $\mathcal{E}^-_a$ will serve as guidelines to get rid of $[a]\bot$ in each φ-world allowed by the theory: they are the laws to be weakened to allow for $\langle a\rangle\top$.

The idea behind our algorithm is as follows: to force $\varphi \rightarrow \langle a\rangle\top$ to be true in all models of the resulting theory, we visit every possible φ-context allowed by it and make the following operations to ensure $\langle a\rangle\top$ is the case for that context: Given a φ-context, if $\mathcal{T}$ not always precludes a from being executed in it, we can safely force $\langle a\rangle\top$ without modifying other laws. On the other hand, if a is always inexecutable in that context, then we should weaken the laws in $\mathcal{E}^-_a$. The first thing we must do is to preserve all old effects in all other φ-worlds. To achieve that we specialize the above laws to each possible valuation (maximal conjunction of literals) satisfying φ but the actual one. Then, in the current φ-valuation, we must ensure that action a may have any effect, i.e., from this φ-world we can reach any other possible world. We achieve that by weakening the *consequent* of the laws in $\mathcal{E}^-_a$ to the exclusive disjunction of all possible contexts in $\mathcal{T}$. Finally, to get minimal change, we must ensure that all literals in this φ-valuation that are not forced to change are preserved. We do this by stating a conditional frame axiom of the form $(\varphi_k \wedge \ell) \rightarrow [a]\ell$, where φ_k is the above φ-valuation.

Algorithm 3 gives the pseudo-code for that.

Correctness of the Algorithms

Suppose we have two atoms p_1 and p_2, and only one action a. Let the action theory $\mathcal{T}_1 = \{\neg p_2, p_1 \rightarrow [a]p_2, \langle a\rangle\top\}$. The only model of $\mathcal{T}_1$ is $\mathscr{M}$ in Figure 4. Revising such a model by $p_1 \vee p_2$ gives us the models $\mathscr{M}'_i$, $1 \leq i \leq 3$, in Figure 4. Now, revising $\mathcal{T}_1$ by $p_1 \vee p_2$ will give us $\mathcal{T}_{1\,p_1\vee p_2} = \{p_1 \wedge \neg p_2, p_1 \rightarrow [a]p_2\}$. The only model of $\mathcal{T}_{1\,p_1\vee p_2}$ is $\mathscr{M}'_1$ in Figure 4. This means that the semantic revision produces models (viz. $\mathscr{M}'_2$ and $\mathscr{M}'_3$ in Figure 4) that are not models of the revised theories.

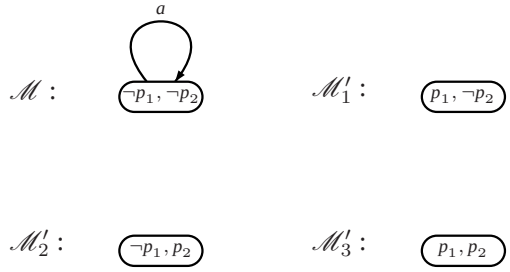


Figure 4: The model $\mathscr{M}$ of $\mathcal{T}$ and the semantic revision of $\mathscr{M}$ by $p_1 \vee p_2$.

The other way round, the algorithms may produce theories whose models do not result from the semantic revision of some model of the original theory. As an example, consider $\mathcal{T}_2 = \{(p_1 \vee p_2) \rightarrow [a]\bot, \langle a\rangle\top\}$, whose only model is $\mathscr{M}$ in Figure 4. The revision of $\mathscr{M}$ by $p_1 \vee p_2$ is as above. However $\mathcal{T}_{2\,p_1\vee p_2} = \{p_1 \vee p_2, (p_1 \vee p_2) \rightarrow [a]\bot\}$ has a model $\mathscr{M}'' = \langle\{\{p_1, p_2\}, \{p_1, \neg p_2\}, \{\neg p_1, p_2\}\}, \emptyset\rangle$ that is not in $\mathscr{M}^*_{p_1\vee p_2}$.

This happens because the possible states are not completely characterized by the static laws in $\mathcal{S}$. Fortunately

we get the right result by requiring $\mathcal{S}$ to be 'big enough'. This is connected with the principle of *modularity* (Herzig and Varzinczak 2005):

Definition 11 (Modularity (Herzig and Varzinczak 2005)) *$\mathcal{T}$ is* modular *if and only if for every $\varphi \in \mathfrak{Fml}$, if $\mathcal{T} \models_{\mathsf{PDL}} \varphi$, then $\mathcal{S} \models_{\mathsf{CPL}} \varphi$.*

Under modularity, revision of models of $\mathcal{T}$ by a law Φ in the semantics produces models of the output of the algorithms $\mathcal{T}^*_\Phi$:

Theorem 3 *Let $\mathcal{T}$ be modular and Φ be a law. For all models $\mathscr{M}'$, if $\mathscr{M}' \in \mathcal{M}^*_\Phi$, for some $\mathcal{M} = \{\mathscr{M} : \models^{\mathscr{M}} \mathcal{T}\}$, then $\models^{\mathscr{M}'} \mathcal{T}^*_\Phi$.*

Also under modularity, models of $\mathcal{T}^*_\Phi$ result from revision of models of $\mathcal{T}$ by Φ:

Theorem 4 *Let $\mathcal{T}$ be modular and Φ a law. For every $\mathscr{M}'$, if $\models^{\mathscr{M}'} \mathcal{T}^*_\Phi$, then $\mathscr{M}' \in \mathcal{M}^*_\Phi$, for some $\mathcal{M} = \{\mathscr{M} : \models^{\mathscr{M}} \mathcal{T}\}$.*

In (Herzig and Varzinczak 2005) algorithms are given to check whether $\mathcal{T}$ satisfies the principle of modularity and also to make $\mathcal{T}$ satisfy it, if that is not the case.

Modular theories have other interesting properties (Herzig and Varzinczak 2007): for example, consistency amounts to that of $\mathcal{S}$; deduction of effect laws does not need the executability ones and vice versa; prediction of an effect of a sequence of actions $a_1; \ldots; a_n$ does not need the effect laws for actions other than $a_1, \ldots, a_n$. This also applies to plan validation when deciding whether $\langle a_1; \ldots; a_n \rangle \varphi$ is the case.

Conclusion and Perspectives

Contrary to classical belief change, the problem of action theory change has only recently received attention in the literature, both in action languages (Baral and Lobo 1997; Eiter et al. 2005) and in dynamic logic (Herzig, Perrussel, and Varzinczak 2006; Varzinczak 2008a).

Here we have studied what revising action theories by a law means, both in the semantics and at the syntactical level. We have defined a semantics based on distances between models that also captures minimal change w.r.t. the preservation of effects of actions. With our algorithms and the correctness results under modularity we have established the link between the semantics and the syntax, and have also shown that the modularity notion is fruitful. Since modularity is preserved across revision (see Lemma 1 in the appendices), it has to be ensured only once during the evolution of the action theory.

Here we presented the case for revision. In (Varzinczak 2008a) we also define the contraction counterpart of action theory change. There we show that moreover our constructions satisfy all Katsuno and Mendelzon's postulates for contraction (Katsuno and Mendelzon 1992).

Our next step on the subject is to define a general framework in which to revise a theory by *any* formula of the language and not only laws. We believe that such a definition will use as basic operations semantic modifications like those we studied here (addition/removal of arrows and worlds). Hence our constructions will help us in better understanding what revision by a general formula means.

Acknowledgements

The author is thankful to Andreas Herzig and Laurent Perrussel for interesting discussions on the subject of this work.

This work has been partially supported by the government of the FEDERATIVE REPUBLIC OF BRAZIL. Grant: CAPES BEX 1389/01-7.

References

Baral, C., and Lobo, J. 1997. Defeasible specifications in action theories. In *Proc. IJCAI*, 1441–1446.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating action domain descriptions. In *Proc. IJCAI*, 418–423.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Harel, D.; Tiuryn, J.; and Kozen, D. 2000. *Dynamic Logic*. MIT Press.

Herzig, A., and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence* 115(1):107–138.

Herzig, A., and Varzinczak, I. 2005. On the modularity of theories. In *Advances in Modal Logic*, volume 5. King's College Publications. 93–109.

Herzig, A., and Varzinczak, I. 2007. Metatheory of actions: beyond consistency. *Artificial Intelligence* 171:951–984.

Herzig, A.; Perrussel, L.; and Varzinczak, I. 2006. Elaborating domain descriptions. In *Proc. ECAI*, 397–401.

Katsuno, H., and Mendelzon, A. 1992. On the difference between updating a knowledge base and revising it. In *Belief revision*. Cambridge. 183–203.

Kracht, M., and Wolter, F. 1991. Properties of independently axiomatizable bimodal logics. *J. of Symbolic Logic* 56(4):1469–1485.

Parikh, R. 1999. Beliefs, belief revision, and splitting languages. In *Logic, Language and Computation*, 266–278.

Quine, W. V. O. 1952. The problem of simplifying truth functions. *American Mathematical Monthly* 59:521–531.

Varzinczak, I. 2008a. Action theory contraction and minimal change. To appear in Proc. KR 2008.

Varzinczak, I. 2008b. Action theory revision. Technical Report IRIT/RT–2008-1–FR, IRIT, Toulouse.

Winslett, M.-A. 1988. Reasoning about action using a possible models approach. In *Proc. AAAI*, 89–93.

Proof of Theorem 3

Let Φ be a law, $\mathscr{M}' \in \mathcal{M}^*_\Phi$, and let $\mathcal{T}^*_\Phi$ be the output of our algorithms on input theory $\mathcal{T}$ and law Φ.

If $\mathcal{T} \cup \{\Phi\} \not\models_{\mathsf{PDL}} \bot$, then $\mathscr{M}' \in \mathcal{M} \setminus \{\mathscr{M} : \not\models^{\mathscr{M}} \Phi\}$ and $\mathscr{M}'$ is a model of $\mathcal{T}^*_\Phi = \mathcal{T} \cup \{\Phi\}$.

Let $\mathcal{T} \cup \{\Phi\} \models_{\mathsf{PDL}} \bot$. We analyze each case.

Let Φ be some $\varphi \in \mathfrak{Fml}$. Then $\mathscr{M}' = \langle W', R' \rangle$ where $W' = (W \setminus val(\neg\varphi)) \cup val(\varphi)$ is minimal w.r.t. W and $R' \subseteq R$ is maximal w.r.t. R, for some $\mathscr{M} = \langle W, R \rangle \in \mathcal{M}$.

As we have assumed the syntactical classical revision operator $\star$ is sound and complete w.r.t. its semantics and is moreover minimal, we have $\models^{\mathscr{M}'} \mathcal{S} \star \varphi$. Because $R' \subseteq R$, $\models^{\mathscr{M}'} \mathcal{E}$. Thus it is enough to show that $\mathscr{M}'$ is a model of the added laws.

Given $(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow \langle a\rangle\top \in \mathcal{T}^*_\varphi$, for every $w \in W'$, if $\models^{\mathscr{M}'}_w \varphi_i \wedge \pi \wedge \varphi_A$, then $w \in W$ (because $\mathcal{S} \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$). From $w \Vdash \varphi_i$ and $\varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}_a$, we have $R_a(w) \neq \emptyset$. Suppose $R'_a(w) = \emptyset$. As $\models^{\mathscr{M}'} \mathcal{S} \star \varphi \cup \mathcal{E}$ and R' is maximal, every $\mathscr{M}'' = \langle W'', R''\rangle$ s.t. $\models^{\mathscr{M}''} \mathcal{S} \star \varphi \cup \mathcal{E}$ is s.t. $R''_a(w) = \emptyset$, and then $\mathcal{S} \star \varphi \cup \mathcal{E} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot$. Because $\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\top$, and $\mathcal{S} \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$ and $\mathcal{S} \star \varphi \not\models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$, we get $\mathcal{S} \star \varphi, \mathcal{E}, \mathcal{X} \models_{\overline{\mathsf{PDL}}} \neg(\pi \wedge \varphi_A)$, and then $(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow \langle a\rangle\top \notin \mathcal{T}^*_\varphi$. Hence $R'_a(w) \neq \emptyset$, and $\models^{\mathscr{M}'} (\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow \langle a\rangle\top$.

If $(\pi \wedge \varphi_A) \rightarrow [a]\bot \in \mathcal{T}^*_\varphi$, then $\mathcal{S} \models_{\mathsf{CPL}} (\pi \wedge \varphi_A) \rightarrow \bot$. Thus, for every $w \in W'$, if $\models^{\mathscr{M}'}_w \pi \wedge \varphi_A$, $R'_a(w) = \emptyset$ and the result follows.

Let Φ now have the form $\varphi \rightarrow [a]\psi$, for $\varphi, \psi \in \mathfrak{Fml}$. Then $\mathcal{M}' = \langle W', R'\rangle$ for some $\mathscr{M} = \langle W, R\rangle \in \mathcal{M}$ s.t. $W' = W$ and $R' \subseteq R$, where R' is maximal w.r.t. R.

From $W' = W$, $\models^{\mathscr{M}'} \mathcal{S}$. As $R' \subseteq R$, $\models^{\mathscr{M}'} \mathcal{E}$. Because $\mathcal{S} \cup \mathcal{E} \subseteq \mathcal{T}^*_{\varphi \rightarrow [a]\psi}$, it suffices to show that $\mathscr{M}'$ is a model of the added laws.

By definition, $\models^{\mathscr{M}'} \varphi \rightarrow [a]\psi$, and then $\models^{\mathscr{M}'} (\pi \wedge \varphi_A) \rightarrow [a]\psi$ for every $\pi \in IP(\mathcal{S} \wedge \varphi)$.

If $(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow \langle a\rangle\top \in \mathcal{T}^*_{\varphi \rightarrow [a]\psi}$, then for every $w \in W'$, if $w \Vdash \varphi_i \wedge \pi \wedge \varphi_A$, we have $w \Vdash \varphi_i$. As $w \in W$, and $\varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}_a$, $R_a(w) = \emptyset$. If $R'_a(w) = \emptyset$, then $w' \Vdash \neg\psi$ for every $w' \in R_a(w)$. Thus as far as we added $(\pi \wedge \varphi_A) \rightarrow [a]\psi$ to $\mathcal{T}^*_{\varphi \rightarrow [a]\psi}$, we must have $\mathcal{T}^*_{\varphi \rightarrow [a]\psi} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot$. Hence $R'_a(w) \neq \emptyset$.

Let $(\varphi_i \wedge \bigwedge_{\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\neg\psi} \neg(\pi \wedge \varphi_A)) \rightarrow \langle a\rangle\top \in \mathcal{T}^*_{\varphi \rightarrow [a]\psi}$. For every $w \in W'$, if $\models^{\mathscr{M}'}_w \varphi_i \wedge \bigwedge_{\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\neg\psi} \neg(\pi \wedge \varphi_A)$, then $w \Vdash \varphi_i$, and as $w \in W$ and $\varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}_a$, we have $R_a(w) \neq \emptyset$. If $R'_a(w) = \emptyset$, because $\models^{\mathscr{M}'} \mathcal{S} \wedge \mathcal{E}$ and R' is maximal, every $\mathscr{M}'' = \langle W'', R''\rangle$ s.t. $\models^{\mathscr{M}''} \mathcal{S} \wedge \mathcal{E}$ is s.t. $R''_a(w) = \emptyset$. Then $\mathcal{S}, \mathcal{E} \models_{\overline{\mathsf{PDL}}} \bigwedge_{\ell \in w} \ell \rightarrow [a]\bot$. But then $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \bigwedge_{\ell \in w} \ell \rightarrow [a]\bot$, and as $\varphi_i \rightarrow \langle a\rangle\top \in \mathcal{X}_a$, $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \neg(\bigwedge_{\ell \in w} \ell \wedge \varphi_i)$, and then $w \notin W$, a contradiction. Hence $R'_a(w) \neq \emptyset$.

Finally, let Φ be of the form $\varphi \rightarrow \langle a\rangle\top$, for some $\varphi \in \mathfrak{Fml}$. Then $\mathscr{M}' = \langle W', R'\rangle$ for some $\mathscr{M} = \langle W, R\rangle \in \mathcal{M}$ s.t. $W' = W$ and $R' = R \cup R^{\varphi,\top}_a$, with

$$R^{\varphi,\top}_a = \{(w, w') : w' \in RelTgt(w, \varphi \rightarrow \langle a\rangle\top, \mathscr{M}, \mathcal{M})\}$$

such that R' is minimal w.r.t. R.

From $W' = W$, $\models^{\mathscr{M}'} \mathcal{S}$. As $R \subseteq R'$, $\models^{\mathscr{M}'} \mathcal{X}$. As far as $\mathcal{S} \cup \mathcal{X} \subseteq \mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$, it is enough to show that $\mathscr{M}'$ satisfies the added laws.

By definition, $\models^{\mathscr{M}'} \varphi \rightarrow \langle a\rangle\top$, and then $\models^{\mathscr{M}'} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\top$ for every $\pi \in IP(\mathcal{S} \wedge \varphi)$.

If $(\varphi_i \wedge \pi \wedge \varphi_A) \rightarrow [a](\psi_i \vee \bigoplus_{\substack{\pi' \in IP(\mathcal{S}) \\ A' \subseteq \overline{atm(\pi')}}} (\pi' \wedge \varphi_{A'})) \in \mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$, then for every $w \in W'$, if $w \Vdash \varphi_i \wedge \pi \wedge \varphi_A$, then $w \Vdash \varphi_i$. Because $\models^{\mathscr{M}} \varphi_i \rightarrow [a]\psi_i$, we have $\models^{\mathscr{M}}_{w'} \psi_i$ for all $w' \in W$ s.t. $(w, w') \in R_a$, and then $\models^{\mathscr{M}'}_{w'} \psi_i$ for every $w' \in W'$ s.t. $(w, w') \in R'_a \setminus R^{\varphi,\top}_a$. Now, given $(w, w') \in R^{\varphi,\top}_a$, we have $\models^{\mathscr{M}'}_{w'} \bigoplus_{\substack{\pi' \in IP(\mathcal{S}) \\ A' \subseteq \overline{atm(\pi')}}} (\pi' \wedge \varphi_{A'})$, and the result follows.

Let $(\varphi_i \wedge \bigwedge_{\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot} \neg(\pi \wedge \varphi_A)) \rightarrow [a]\psi_i \in \mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$. For every $w \in W'$, if $\models^{\mathscr{M}'}_w \varphi_i \wedge \bigwedge_{\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot} \neg(\pi \wedge \varphi_A)$, then $w \Vdash \varphi_i$, and as $\models^{\mathscr{M}} \varphi_i \rightarrow [a]\psi_i$, we have $\models^{\mathscr{M}}_{w'} \psi_i$ for all $w' \in W$ s.t. $(w, w') \in R_a$. Thus $\models^{\mathscr{M}'}_{w'} \psi_i$ for every $w' \in W'$ s.t. $(w, w') \in R'_a \setminus R^{\varphi,\top}_a$. Now, if $w \not\Vdash \varphi$, then $R^{\varphi,\top}_a = \emptyset$ and the result follows. Otherwise, if $w \Vdash \varphi$, then $\mathcal{T} \not\models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot$, and then $(\varphi_i \wedge \bigwedge_{\mathcal{T} \models_{\overline{\mathsf{PDL}}} (\pi \wedge \varphi_A) \rightarrow [a]\bot} \neg(\pi \wedge \varphi_A)) \rightarrow [a]\psi_i$ has not been put in $\mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$, a contradiction.

Let now $(\pi \wedge \varphi_A \wedge \ell) \rightarrow [a]\ell \in \mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$. For every $w \in W'$, if $\models^{\mathscr{M}'}_w \pi \wedge \varphi_A \wedge \ell$, then $\models^{\mathscr{M}'}_w \ell$, and then $\models^{\mathscr{M}}_w \ell$. From $(\pi \wedge \varphi_A \wedge \ell) \rightarrow [a]\ell \in \mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$, we have $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \ell \rightarrow [a]\bot$ or $\mathcal{T} \not\models_{\overline{\mathsf{PDL}}} \ell \rightarrow [a]\neg\ell$ and $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \ell \rightarrow [a]\ell$. In both cases, $\models^{\mathscr{M}}_{w'} \ell$ for every $w' \in R_a(w)$, and then $\models^{\mathscr{M}'}_{w'} \ell$ for every w' s.t. $(w, w') \in R' \setminus R^{\varphi,\top}_a$. It remains to show that $\models^{\mathscr{M}'}_{w'} \ell$ for every $w' \in W'$ s.t. $(w, w') \in R^{\varphi,\top}_a$.

Suppose $\not\models^{\mathscr{M}'}_{w'} \ell$. Then $\neg\ell \in w' \setminus w$. From the construction of $\mathscr{M}'$, there is $\mathscr{M}'' = \langle W'', R''\rangle \in \mathcal{M}$ s.t. there is $(u, v) \in R''_a$ and $\neg\ell \in v \setminus u$, i.e., $\models^{\mathscr{M}''}_u \ell$ and $\models^{\mathscr{M}''}_v \neg\ell$. From $(u, v) \in R''_a$, we do not have $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \ell \rightarrow [a]\bot$. From $\models^{\mathscr{M}''}_v \neg\ell$, we do not have $\mathcal{T} \models_{\overline{\mathsf{PDL}}} \ell \rightarrow [a]\ell$. Thus the algorithm has not put $(\pi \wedge \varphi_A \wedge \ell) \rightarrow [a]\ell$ in $\mathcal{T}^*_{\varphi \rightarrow \langle a\rangle\top}$, a contradiction. ■

Proof of Theorem 4

Lemma 1 *Let Φ be a law. If $\mathcal{T}$ is modular and $\mathcal{T} \cup \{\Phi\} \models_{\overline{\mathsf{PDL}}} \bot$, then $\mathcal{T}^*_\Phi$ is modular.*

Proof: Let Φ be nonclassical. Suppose $\mathcal{T}^*_\Phi$ is not modular. Then there is $\varphi' \in \mathfrak{Fml}$ s.t. $\mathcal{T}^*_\varphi \models_{\overline{\mathsf{PDL}}} \varphi'$ and $\mathcal{S}' \not\models_{\mathsf{CPL}} \varphi'$, where $\mathcal{S}'$ is static laws in $\mathcal{T}^*_\Phi$. Suppose $\mathcal{T} \not\models_{\overline{\mathsf{PDL}}} \varphi'$. Then we must have $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} \neg\varphi' \rightarrow [a]\bot$ and $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} \neg\varphi' \rightarrow \langle a\rangle\top$.

Suppose Φ has the form $\varphi \rightarrow [a]\psi$, for $\varphi, \psi \in \mathfrak{Fml}$. Then for all $\varphi \wedge \neg\varphi'$-contexts, as far as $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} (\varphi \wedge \neg\varphi') \rightarrow [a]\bot$, $(\varphi \wedge \neg\varphi') \rightarrow \langle a\rangle\top \notin \mathcal{T}^*_\Phi$. Then $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} \varphi'$ if and only if $\mathcal{S}' \models_{\mathsf{CPL}} \varphi'$, a contradiction.

Suppose Φ is of the form $\varphi \rightarrow \langle a\rangle\top$, for $\varphi \in \mathfrak{Fml}$. Then for all $\varphi \wedge \neg\varphi'$-contexts such that $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} (\varphi \wedge \neg\varphi') \rightarrow \langle a\rangle\top$, $\mathcal{T}^*_\Phi \models_{\overline{\mathsf{PDL}}} (\varphi \wedge \neg\varphi') \rightarrow [a]\bot$ is impossible as far as $\mathcal{E}^-_a$ has

been weakened. Then $\mathcal{T}^*_\Phi \models_{\mathsf{PDL}} \varphi'$ if and only if $\mathcal{S}' \models_{\mathsf{CPL}} \varphi'$, a contradiction.

Hence we have $\mathcal{T} \models_{\mathsf{PDL}} \varphi'$. Because Φ is nonclassical, $\mathcal{S}' = \mathcal{S}$. Then $\mathcal{T} \models_{\mathsf{PDL}} \varphi'$ and $\mathcal{S} \not\models_{\mathsf{CPL}} \varphi'$, and hence $\mathcal{T}$ is not modular.

Let now Φ be some $\varphi \in \mathfrak{Fml}$. Suppose $\mathcal{T}^*_\varphi$ is not modular, i.e., there is $\varphi'' \in \mathfrak{Fml}$ s.t. $\mathcal{T}^*_\varphi \models_{\mathsf{PDL}} \varphi''$ and $\mathcal{S}' = \mathcal{S} \star \varphi \not\models_{\mathsf{CPL}} \varphi''$.

From $\mathcal{S}' \not\models_{\mathsf{CPL}} \varphi''$, there is $v \in val(\mathcal{S}')$ s.t. $v \not\Vdash \varphi''$.

If $v \in val(\mathcal{S})$, as $\mathcal{T}$ is modular, $\mathcal{T} \not\models_{\mathsf{PDL}} \varphi''$. From this and $\mathcal{T}^*_\varphi \models_{\mathsf{PDL}} \varphi''$, we must have $\mathcal{T}^*_\varphi \models_{\mathsf{PDL}} \neg\varphi'' \rightarrow [a]\bot$ and $\mathcal{T}^*_\varphi \models_{\mathsf{PDL}} \neg\varphi'' \rightarrow \langle a\rangle\top$. From the latter, we get $\mathcal{T} \models_{\mathsf{PDL}} \neg\varphi'' \rightarrow \langle a\rangle\top$, and from the first we have $\mathcal{T} \models_{\mathsf{PDL}} \neg\varphi'' \rightarrow [a]\bot$. Putting both results together we get $\mathcal{T} \models_{\mathsf{PDL}} \varphi''$. As $\mathcal{S} \not\models_{\mathsf{CPL}} \varphi''$, we have a contradiction.

If $v \notin val(\mathcal{S})$, then $\mathcal{T}^*_\varphi \not\models_{\mathsf{PDL}} \neg\varphi'' \rightarrow \langle a\rangle\top$, as no executability for context $\neg\varphi''$ has been put into $\mathcal{T}^*_\varphi$. Hence $\mathcal{T}^*_\varphi \not\models_{\mathsf{PDL}} \varphi''$, a contradiction. ■

Lemma 2 *If $\mathscr{M}_{big} = \langle W_{big}, R_{big}\rangle$ is a model of $\mathcal{T}$, then for every $\mathscr{M} = \langle W, R\rangle$ such that $\models^{\mathscr{M}} \mathcal{T}$ there is a minimal (w.r.t. set inclusion) extension $R' \subseteq R_{big} \setminus R$ such that $\mathscr{M}' = \langle val(\mathcal{S}), R \cup R'\rangle$ is a model of $\mathcal{T}$.*

Proof: See (Varzinczak 2008b). ■

Lemma 3 *Let $\mathcal{T}$ be modular, and Φ be a law. Then $\mathcal{T} \models_{\mathsf{PDL}} \Phi$ if and only if every $\mathscr{M}' = \langle val(\mathcal{S}), R'\rangle$ such that $\models^{\langle W,R\rangle} \mathcal{T}$ and $R \subseteq R'$ is a model of Φ.*

Proof:
($\Rightarrow$): Straightforward, as $\mathcal{T} \models_{\mathsf{PDL}} \Phi$ implies $\models^{\mathscr{M}} \Phi$ for every $\mathscr{M}$ such that $\models^{\mathscr{M}} \mathcal{T}$, in particular for those that are extensions of some model of $\mathcal{T}$.

($\Leftarrow$): Suppose $\mathcal{T} \not\models_{\mathsf{PDL}} \Phi$. Then there is $\mathscr{M} = \langle W, R\rangle$ such that $\models^{\mathscr{M}} \mathcal{T}$ and $\not\models^{\mathscr{M}} \Phi$. As $\mathcal{T}$ is modular, the big model $\mathscr{M}_{big} = \langle W_{big}, R_{big}\rangle$ of $\mathcal{T}$ is a model of $\mathcal{T}$. Then by Lemma 2 there is a minimal extension R' of R w.r.t. R_{big} such that $\mathscr{M}' = \langle val(\mathcal{S}), R \cup R'\rangle$ is a model of $\mathcal{T}$. Because $\not\models^{\mathscr{M}} \Phi$, there is $w \in W$ such that $\not\models^{\mathscr{M}}_w \Phi$. If Φ is some $\varphi \in \mathfrak{Fml}$ or an effect law, any extension $\mathscr{M}'$ of $\mathscr{M}$ is such that $\not\models^{\mathscr{M}'}_w \Phi$. If Φ is of the form $\varphi \rightarrow \langle a\rangle\top$, then $\models^{\mathscr{M}}_w \varphi$ and $R_a(w) = \emptyset$. As any extension of $\mathscr{M}$ is such that $(u, v) \in R'$ if and only if $u \in val(\mathcal{S}) \setminus W$, only worlds other than those in W get a new leaving arrow. Thus $(R \cup R')_a(w) = \emptyset$, and then $\not\models^{\mathscr{M}'}_w \Phi$. ■

Lemma 4 *Let $\mathcal{T}$ be modular and Φ a law. If $\mathscr{M}' = \langle val(\mathcal{S}'), R'\rangle$ is a model of $\mathcal{T}^*_\Phi$, then there is $\mathcal{M} = \{\mathscr{M} : \models^{\mathscr{M}} \mathcal{T}\}$ s.t. $\mathscr{M}' \in \mathcal{M}^*_\Phi$.*

Proof: Let $\mathscr{M}' = \langle val(\mathcal{S}'), R'\rangle$ be such that $\models^{\mathscr{M}'} \mathcal{T}^*_\Phi$. If $\models^{\mathscr{M}'} \mathcal{T}$, the result follows. Suppose $\not\models^{\mathscr{M}'} \mathcal{T}$. We analyze each case.

Let Φ be of the form $\varphi \rightarrow [a]\psi$, for $\varphi, \psi \in \mathfrak{Fml}$. Let $\mathcal{M} = \{\mathscr{M} : \mathscr{M} = \langle val(\mathcal{S}), R\rangle\}$. As $\mathcal{T}$ is modular, by Lemmas 2 and 3, $\mathcal{M}$ is non-empty and contains only models of $\mathcal{T}$.

Suppose $\mathscr{M}'$ is not a minimal model of $\mathcal{T}^*_{\varphi\rightarrow[a]\psi}$, i.e., there is $\mathscr{M}''$ such that $\mathscr{M}'' \preceq_{\mathscr{M}} \mathscr{M}'$ for some $\mathscr{M} \in \mathcal{M}$. Then $\mathscr{M}'$ and $\mathscr{M}''$ differ only in the effect of a in a given φ-world, viz. a $\pi \wedge \varphi_A$-context, for some $\pi \in IP(\mathcal{S} \wedge \varphi)$ and $\varphi_A = \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \in A}} p_i \wedge \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \notin A}} \neg p_i$ such that $A \subseteq \overline{atm(\pi)}$.

Because $\not\models^{\mathscr{M}'} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\neg\psi$, we must have $\models^{\mathscr{M}''} (\pi \wedge \varphi_A) \rightarrow \langle a\rangle\neg\psi$, and then $\not\models^{\mathscr{M}''} \varphi \rightarrow [a]\psi$. Hence $\mathscr{M}'$ is minimal w.r.t. $\preceq_{\mathscr{M}}$.

When revising by an effect law, $\mathcal{S}' = \mathcal{S}$. Hence taking the right R and $R^{\varphi,\neg\psi}_a$ such that $\mathscr{M} = \langle val(\mathcal{S}), R\rangle$ and $R' = R \setminus R^{\varphi,\neg\psi}_a$, for some $R^{\varphi,\neg\psi}_a \subseteq \{(w, w') : \models^{\mathscr{M}}_w \varphi, \models^{\mathscr{M}}_{w'} \neg\psi$ and $(w, w') \in R_a\}$, we have $\mathscr{M} \in \mathcal{M}$ and then $\mathscr{M}' \in \mathcal{M}^*_{\varphi\rightarrow[a]\psi}$.

Let Φ have the form $\varphi \rightarrow \langle a\rangle\top$, for $\varphi \in \mathfrak{Fml}$. Let $\mathcal{M} = \{\mathscr{M} : \mathscr{M} = \langle val(\mathcal{S}), R\rangle\}$. As $\mathcal{T}$ is modular, by Lemmas 2 and 3, $\mathcal{M}$ is non-empty and contains only models of $\mathcal{T}$.

Suppose that $\mathscr{M}'$ is not a minimal model of $\mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top}$, i.e., there is $\mathscr{M}''$ such that $\models^{\mathscr{M}''} \mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top}$ and $\mathscr{M}'' \preceq_{\mathscr{M}} \mathscr{M}'$ for some $\mathscr{M} \in \mathcal{M}$. Then $\mathscr{M}'$ and $\mathscr{M}''$ differ only on the executability of a in a given φ-world, i.e., a $\pi \wedge \varphi_A$-context, for some $\pi \in IP(\mathcal{S} \wedge \varphi)$ and $\varphi_A = \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \in A}} p_i \wedge \bigwedge_{\substack{p_i \in \overline{atm(\pi)} \\ p_i \notin A}} \neg p_i$, such that $A \subseteq \overline{atm(\pi)}$. This means $\mathscr{M}''$ has no arrow leaving this $\pi \wedge \varphi_A$-world. Then $\models^{\mathscr{M}''} (\pi \wedge \varphi_A) \rightarrow [a]\bot$, and hence $\not\models^{\mathscr{M}''} \varphi \rightarrow \langle a\rangle\top$. Hence $\mathscr{M}'$ is a minimal model of $\mathcal{T}^*_{\varphi\rightarrow\langle a\rangle\top}$ w.r.t. $\preceq_{\mathscr{M}}$.

When revising by executability laws, $\mathcal{S}' = \mathcal{S}$. Thus taking the right R and a minimal $R^{\varphi,\top}_a$ such that $\mathscr{M} = \langle val(\mathcal{S}), R\rangle$ and $R' = R \cup R^{\varphi,\top}_a$, for some $R^{\varphi,\top}_a \subseteq \{(w, w') : \models^{\mathscr{M}}_w \varphi$ and $w' \in RelTgt(w, \varphi \rightarrow \langle a\rangle\top, \mathscr{M}, \mathcal{M})\}$, we get $\mathscr{M} \in \mathcal{M}$ and then $\mathscr{M}' \in \mathcal{M}^*_{\varphi\rightarrow\langle a\rangle\top}$.

Finally, let Φ be some $\varphi \in \mathfrak{Fml}$. Then $\mathscr{M}'$ is such that for every $w \in W'$, if $R'_a(w) \neq \emptyset$, then $w \in val(\mathcal{S})$ and $R_a(w) \neq \emptyset$ for every $\mathscr{M} = \langle W, R\rangle \in \mathcal{M}$. Choosing the right $\mathscr{M} \in \mathcal{M}$ the result follows. ■

Proof of Theorem 4

Let $\mathcal{T}^*_\Phi$ be the output of our algorithms on input theory $\mathcal{T}$ and law Φ. If $\mathcal{T}^*_\Phi = \mathcal{T} \cup \{\Phi\}$, then $\mathcal{T} \cup \{\Phi\} \not\models_{\mathsf{PDL}} \bot$, and hence every $\mathscr{M}'$ such that $\models^{\mathscr{M}'} \mathcal{T}^*_\Phi$ is such that $\mathscr{M}' \in \mathcal{M} \setminus \{\mathscr{M} : \not\models^{\mathscr{M}} \Phi\}$ and the result follows.

Suppose $\mathcal{T} \cup \{\Phi\} \models_{\mathsf{PDL}} \bot$. From the hypothesis that $\mathcal{T}$ is modular and Lemma 1, $\mathcal{T}'$ is modular. Then $\mathscr{M}' = \langle val(\mathcal{S}'), R\rangle$ is a model of $\mathcal{T}'$, by Lemma 2. From this and Lemma 3 the result follows. ■

Properties of Knowledge Forgetting

Yan Zhang and Yi Zhou
Intelligent Systems Laboratory
University of Western Sydney, Australia
E-mail: {yan,yzhou}@scm.uws.edu.au

Abstract

In this paper we propose a formal theory of knowledge forgetting based on the single agent S5 modal logic. We first present a model theoretic definition of knowledge forgetting and study its essential semantic properties. We show that knowledge forgetting is a generalization of variable forgetting in propositional logic and can be precisely characterized by four forgetting postulates. We then investigate the computational properties of knowledge forgetting. We observe that each propositional S5 formula can be transformed into a kind of disjunctive normal form - this result leads us to develop an algorithm for computing the syntactic representation of knowledge forgetting. By studying the major decision problems, we prove that the complexity of model checking problem for knowledge forgetting is NP complete, and the complexity of its related inference problems varies from co-NP complete to Π_2^{P} complete.

Key words: belief revision and update, knowledge update, computational aspects of knowledge representation

Introduction

Epistemic reasoning concerns the problem of how to reason about agents' epistemic states (knowledge) in a dynamic environment. In the last decade, it has been demonstrated that epistemic reasoning has many important applications in computer science and AI fields (Meyer & van der Hoek 1995). Amongst various theories and approaches, one major assumption in the study of epistemic reasoning is that agents always remember their previous knowledge (i.e. agents have perfect recall). However, as pointed by Fagin *et al.*: "*There are often scenarios of interest where we want to model the fact that certain information is discarded. In practice, for example, an agent may simply not have enough memory capacity to remember everything he has learned.*" (page 129 in (Fagin *et al.* 1995)). Hence knowledge forgetting is an important behaviour for an agent under certain circumstances.

An earlier formal study on the concept of knowledge forgetting was due to Baral and Zhang's work on knowledge update ((Baral & Zhang 2005)), where they treated knowledge forgetting as a special form of update with the effect $\neg K\phi \wedge \neg K\neg\phi$: after knowledge forgetting ϕ (ϕ is a propositional formula), the agent would neither know ϕ nor $\neg\phi$. Since Baral and Zhang's knowledge forgetting is specified from a knowledge update viewpoint, it does not seem to obey the general intuition of forgetting in classical propositional theories (Lin & Reiter 1994), and their semantics of forgetting update does not always represent an intuitive meaning.

As a logical notion, forgetting was first formally defined in propostional and first order logics by Lin and Reiter (Lin & Reiter 1994). Over the years, researchers have demonstrated many useful applications of the propositional forgetting theory in abductive reasoning, belief revision/update, and reasoning about knowledge (Lang, Liberator, & Marquis 2003; Lin 2001; Shu, Lv, & Zhang 2004). In recent years, various theories of forgetting have also been proposed under the answer set programming semantics and used in solving logic program conflicts (Eiter & Wang 2008 to appear; Zhang & Foo 2006). From these works, we can see that forgetting is an important and useful concept in knowledge representation and reasoning. Nevertheless, it is not difficult to see that existing forgetting definitions in propositional logic and answer set programming are not directly applicable in modal logics. For instance, in propositional forgetting theory, forgetting atom q from $T \equiv (p \rightarrow q) \wedge ((q \wedge r) \rightarrow s)$ is equivalent to a formula $T[q/\top] \vee T[q/\bot]$, where $T[q/\top]$ is a formula obtained from T by replacing each q with $\top$ and $T[q/\bot]$ is obtained from T by replacing each q with $\bot$, which is $(r \rightarrow s) \vee \neg p$. However, this method cannot be extended to a S5 modal logic formula. Consider an S5 formula $T' \equiv Kp \wedge \neg Kq \wedge \neg K\neg q$. If we want to forget atom q from T' by using the above method, we would have $T'[q/\top] \vee T'[q/\bot] \equiv \bot$. This is obviously not correct because after forgetting q, the agent's knowledge set should not become inconsistent!

From the above discussions, in order to have a formal theory of of forgetting in propositional single agent S5 modal logic, some new approach must be developed. In this paper, we present a semantic definition of knowledge forgetting and propose four forgetting postulates that we argue that the underlying knowledge forgetting should obey. We prove a representaiton theorem showing that our knowledge forgetting is precisely characterized by these four postulates. We also study computational properties of knowledge forgetting in details. We first develop an algorithm to compute the syntactic representation of knowledge forgetting from a

given knowledge set, and then investigate the complexity of decision problems in relation to knowledge forgetting. In particular, we show that the complexity of model checking problem for knowledge forgetting is NP complete, and the complexity of its related inference problem varies from co-NP complete to Π_2^P complete.

The rest of the paper is organized as follows. Section 2 presents a model based semantic definition of knowledge forgetting, while section 3 proves an important representation theorem for knowledge forgetting and studies other related semantic properties. Section 4 then addresses major computational properties of knowledge forgetting. Finally section 5 concludes this paper with some remarks.

Defining Knowledge Forgetting

Our knowledge forgetting will be defined on a basis of the finite proportional modal logic S5. Let $Atom$ be a finite set of *atoms* (also called *variables*). The *finite language* $\mathcal{L}$ of propositional S5 modal logic is defined recursively by $Atom$, classical connectives $\bot$, $\neg$, $\rightarrow$ and a modal operator K as follows:

$$\phi ::= \bot \mid p \mid \neg\phi \mid \phi \rightarrow \psi \mid K\phi,$$

where $p \in Atom$. $\top$, $\phi \wedge \psi$ and $\phi \vee \psi$, are defined as the standard way. Elements in $\mathcal{L}$ are called *formulas*. Formulas without modal operators are called *objective formulas*. A *knowledge set* is a finite set of formulas. *Literals* are atoms and their negations. Let ϕ be a formula and Γ a knowledge set, we write $Var(\phi)$ and $Var(\Gamma)$ to denote the set of atoms occurred in ϕ and Γ respectively.

For convenience, we usually use $a, b, c, \cdots, p, q, \cdots$ to denote atoms; $\phi, \psi, \upsilon, \cdots$ to denote formulas; and $\Gamma, T, \cdots$ to denote knowledge sets. Sometimes, we also use $\Gamma = \phi_1 \wedge \cdots \wedge \phi_n$ to represent a finite set of formulas $\{\phi_1, \cdots, \phi_n\}$.

A *Kripke structure* is a triple $S = \langle W, R, L\rangle$, where W is a set of possible worlds, R an equivalence relation on W, and L a set of interpretations for each world in W. As illustrated in (Meyer & van der Hoek 1995), an S5 Kripke interpretation for single agent may be simplified as $M = \langle W, w\rangle$, where W is the set of all possible worlds, each world is identified as a set of atoms, and $w \in W$ is called the actual world. In this case, we call $M = \langle W, w\rangle$ a *k-interpretation*.

The *satisfaction relation* $\models$ between k-interpretations and formulas in $\mathcal{L}$ is defined recursively as follows[1]:

(1) $\langle W, w\rangle \not\models \bot$;
(2) $\langle W, w\rangle \models p$ iff $p \in w$, where $p \in Atom$;
(3) $\langle W, w\rangle \models \neg\phi$ iff $\langle W, w\rangle \not\models \phi$;
(4) $\langle W, w\rangle \models \phi \rightarrow \psi$ iff $\langle W, w\rangle \not\models \phi$ or $\langle W, w\rangle \models \psi$;
(5) $\langle W, w\rangle \models K\phi$ iff $\forall w' \in W$, $\langle W, w'\rangle \models \phi$.

We say that M is a *k-model* of ϕ iff $M \models \phi$. We write $Mod(\phi)$ (or $Mod(\Gamma)$ if Γ is a finite set of formulas) to denote the set of all k-models of ϕ (or Γ resp.). We say that two S5 formulas (knowledge sets) ϕ and ψ are *equivalent*, denoted by $\phi \equiv \psi$, iff $Mod(\phi) = Mod(\psi)$. Let Γ and Γ' be two knowledge sets. We write $\Gamma \models \Gamma'$ iff $Mod(\Gamma) \subseteq Mod(\Gamma')$.

[1] We write $\langle W, w\rangle \not\models F$ if it is not the case that $\langle W, w\rangle \models F$.

To present a formal definition of knowledge forgetting, we first introduce a useful notion. Let w and w' be two worlds identified as two sets of atoms respectively, and $V \subseteq Atom$ a set of atoms. We say that w and w' are *identical with exception* on V, denoted by $w \leftrightarrow_V w'$, if for all $p \in Atom \setminus V$, $p \in w$ iff $p \in w'$. Let $M = \langle W, w\rangle$ and $M' = \langle W', w'\rangle$ be two k-interpretations, and $V \subseteq Atom$ a set of atoms, we say that M and M' are *bisimilar with exception* on V, denoted by $M \leftrightarrow_V M'$, iff there exists a binary relation $\sigma \subseteq W \times W'$ such that:

1. $\sigma(w, w')$;
2. $\forall w^* \in W$, $\exists w^{*'} \in W'$ such that $\sigma(w^*, w^{*'})$ (the forth condition); and
3. $\forall w^{*'} \in W'$, $\exists w^* \in W$ such that $\sigma(w^*, w^{*'})$ (the back condition).
4. if $\sigma(w^*, w^{*'})$ then $w^* \leftrightarrow_V w^{*'}$.

Proposition 1 *The relation $\leftrightarrow_V$ is an equivalence relation.*

Proof: It is easy to see that $\leftrightarrow_V$ satisfies reflexivity and symmetry. We now show that it satisfies transitivity as well. Suppose that M_0, M_1, M_2 are three Kripke interpretations such that $M_0 \leftrightarrow_V M_1$ via the binary relation σ_1 and $M_1 \leftrightarrow_V M_2$ via the binary relation σ_2. Construct a binary relation $\sigma_3 \subseteq W_0 \times W_2$: for any two interpretations $w_0 \in W_0$, $w_2 \in W_2$, $\sigma_3(w_0, w_2)$ iff there exists $w_1 \in W_1$ such that $\sigma_1(w_0, w_1)$ and $\sigma_2(w_1, w_2)$. It is easy to check that $M_0 \leftrightarrow M_2$ via σ_3. □

Now we define knowledge forgetting as follows.

Definition 1 ***(Knowledge forgetting)*** *Let Γ be a knowledge set and $p \in Atom$ an atom. A knowledge set, denoted as $\mathsf{KForget}(\Gamma, p)$, is the result of* knowledge forgetting *p from Γ, if the following condition holds:*

$$Mod(\mathsf{KForget}(\Gamma, p)) = \bigcup_{M \in Mod(\Gamma), M \leftrightarrow_{\{p\}} M'} M'.$$

Similarly to model based update formulations (Baral & Zhang 2005; Winslett 1988), $\mathsf{KForget}(\Gamma, p)$ denotes the result of knowledge forgetting atom p from Γ, where the set of all k-models of $\mathsf{KForget}(\Gamma, p)$ is specified as in the definition. Since we restrict to a finite propositional S5 modal logic, such knowledge set $\mathsf{KForget}(\Gamma, p)$ always exists. In section 5, we also provide an algorithm to compute a concrete S5 formula ϕ such that $\phi \equiv \mathsf{KForget}(\Gamma, p)$.

Example 1 Consider knowledge sets $K(p \vee q)$, $Kp \vee Kq$ and $K(p \wedge q)$. From Definition 1, it is easy to check that $\mathsf{KForget}(K(p \vee q), p) \equiv \top$, $\mathsf{KForget}(Kp \vee Kq, p) \equiv \top$, and $\mathsf{KForget}(K(p \wedge q), p) \equiv Kq$. □

We may extend Definition 1 to the case of knowledge forgetting an arbitrary set V of atoms from Γ.

Definition 2 *Let Γ be a knowledge set and $V \subseteq Atom$ a set of atoms. A knowledge set, denoted as $\mathsf{KForget}(\Gamma, V)$, is the result of knowledge forgetting V from Γ, if the following condition holds:*

$$Mod(\mathsf{KForget}(\Gamma, V)) = \bigcup_{M \in Mod(\Gamma), M \leftrightarrow_V M'} M'.$$

The following proposition ensures that Definitions 1 and 2 are semantically coherent.

Proposition 2 *Let Γ be a knowledge set and V a set of atoms and p an atom such that $p \notin V$. Then* $\mathsf{KForget}(\Gamma, \{p\} \cup V) \equiv \mathsf{KForget}(\mathsf{KForget}(\Gamma, p), V)$.

Proof: Let $M_1 = \langle W_1, s_1\rangle$ be a k-model of $\mathsf{KForget}(\Gamma, \{p\} \cup V)$. By the definition, there exists a k-model $M = \langle W, s\rangle$ of Γ, such that $M \leftrightarrow_{\{p\}\cup V} M_1$ via a binary relation σ. We construct a k-interpretation $M_2 = \langle W_2, s_2\rangle$ as follows: (1) for all pairs $w \in W$ and $w_1 \in W_1$ such that $\sigma(w, w_1)$, let $w_2 \in W_2$ and (a) $p \in w_2$ iff $p \in w_1$, (b) for all atoms $q \in V$, $q \in w_2$ iff $q \in w$, (c) for all other atoms q, $q \in w_2$ iff $q \in w_1$ iff $q \in w$; (2) delete duplicated k-interpretations in W_2; (3) let s_2 be the world such that (a) $p \in s_2$ iff $p \in s_1$, (b) for all atoms $q \in V$, $q \in s_2$ iff $q \in s$, (c) for all other atoms q', $q' \in s_2$ iff $q' \in s_1$ iff $q' \in s$. Then we have the following results:

- $M \leftrightarrow_{\{p\}} M_2$. Let $\sigma_1 \subseteq W \times W_2$ be a binary relation such that $\sigma(w, w_2)$ iff w_2 is constructed based on w (see the construction of M_2 above). It is easy to see that $M \leftrightarrow_{\{p\}} M_2$ via σ_1.
- $M_2 \leftrightarrow_V M_1$. Let $\sigma_2 \subseteq W_2 \times W_1$ be a binary relation such that $\sigma(w_2, w_1)$ iff w_2 is specified based on w_1 (see the construction of M_2 above). It is observed that $M_2 \leftrightarrow_V M_1$ via σ_2.

Thus, M_2 is a k-model of $\mathsf{KForget}(\Gamma, p)$. This follows that M_1 is a k-model of $\mathsf{KForget}(\mathsf{KForget}(\Gamma, p), V)$.

On the other hand, suppose that M_1 is a k-model of $\mathsf{KForget}(\mathsf{KForget}(\Gamma, p), V)$, then there exists M_2 such that M_2 is a k-model of $\mathsf{KForget}(\Gamma, p)$ and $M_2 \leftrightarrow_V M_1$, and there exists M such that M is a k-model of Γ and $M \leftrightarrow_{\{p\}} M_2$. Therefore, $M \leftrightarrow_{\{p\}\cup V} M_1$, and consequently, M_1 is also a k-model of $\mathsf{KForget}(\Gamma, \{p\} \cup V)$. □

Semantic characterizations

In this section we study essential semantic properties of knowledge forgetting. We will first propose a set of postulates and show that these postulates precisely characterize the semantics of knowledge forgetting. We then discuss other desired properties that our knowledge forgetting satisfies.

A representation theorem

Consider a formula ϕ. Intuitively, if a propositional atom (variable) a does not occur in $Var(\phi)$, we may consider that ϕ is *irrelevant* to variable a. It is not surprising that the notion of irrelevance plays an important role to characterize the semantics of knowledge forgetting. We first give the following formal definition.

Definition 3 ***(Irrelevance)*** *Let Γ be a knowledge set and V a set of atoms. We say that Γ is* irrelevant *to V, denoted by $IR(\Gamma, V)$, if there exists a knowledge set Γ' such that $\Gamma \equiv \Gamma'$ and $Var(\Gamma') \cap V = \emptyset$.*

Let Γ and Γ' be two knowledge sets, V a set of atoms. Now we propose the following postulates:

(W) Weakening: $\Gamma \models \Gamma'$.

(PP) Positive Persistence: if $IR(\phi, V)$ and $\Gamma \models \phi$, then $\Gamma' \models \phi$.

(NP) Negative Persistence: if $IR(\phi, V)$ and $\Gamma \not\models \phi$, then $\Gamma' \not\models \phi$.

(IR) Irrelevance: $IR(\Gamma', V)$.

By specifying $\Gamma' \equiv \mathsf{KForget}(\Gamma, V)$, we call (W), (PP), (NP) and (IR) are *four postulates for knowledge forgetting*. Let us take a closer look at these postulates. (W) seems an essential requirement for knowledge forgetting: after forgetting some knowledge from a knowledge set, the resulting knowledge set then becomes weaker. Indeed, as demonstrated in propositional variable forgetting (Lin & Reiter 1994; Lin 2001), forgetting weakens the original formula. The postulates of positive persistence (PP) and negative persistence (NP) simply state that knowledge forgetting a set of atoms should not affect those positive or negative information respectively that is irrelevant to this set of atoms. Also note that that (PP) and (NP) can be combined as: if $IR(\phi, V)$, then $\Gamma \models \phi$ iff $\Gamma' \models \phi$. Finally, irrelevance (IR) means that after knowledge forgetting, the resulting knowledge set should be irrelevant to those atoms which we have (knowledge) forgotten. We argue that these postulates precisely capture the basic properties that knowledge forgetting should satisfy.

Lemma 1 *Let V be a set of atoms, ϕ a formula such that $Var(\phi) \cap V = \emptyset$ and M a model of ϕ. Then for any M' such that $M \leftrightarrow_V M'$, M' is also a model of ϕ.*

Proof: We prove this assertion by induction on the structure of ϕ.

- If ϕ is $\bot$, this assertion holds obviously.
- If ϕ is an atom p, then $M = \langle W, w\rangle \models p$ iff $p \in w$ iff $p \in w'$, where $M' = \langle W', w'\rangle$ and $M \leftrightarrow_V M'$ via the binary relation σ iff $M' \models p$.
- If ϕ is $\neg\psi$, then $M \models \phi$ iff $M \not\models \psi$ iff $M' \not\models \psi$ iff $M' \models \phi$.
- If ϕ is $\phi_1 \rightarrow \phi_2$, then $M \models \phi$ iff $M \not\models \phi_1$ or $M \models \phi_2$ iff $M' \not\models \phi_1$ or $M' \models \phi_2$ iff $M' \models \phi$.
- If ϕ is $K\psi$, then $M \models \phi$ iff for all $w \in W$, $\langle W, w\rangle \models \psi$. Suppose that $M \leftrightarrow_V M'$ via the binary relation σ, given $w' \in W'$, there exists $w \in W$ such that $\sigma(w, w')$ and $w \leftrightarrow_V w'$. Thus, $\langle W, w\rangle \leftrightarrow_V \langle W', w'\rangle$. By induction hypothesis, $\langle W', w'\rangle \models \psi$. Thus, $M' \models \phi$.

This completes the induction proof. □

Lemma 2 *Let V be a set of atoms, ϕ a formula such that $IR(\phi, V)$ and M a k-model of ϕ. Then for any M' such that $M \leftrightarrow_V M'$, M' is also a k-model of ϕ.*

Proof: This assertion follows directly from Lemma 1 □

Now we have the following representation theorem which states that our forgetting postulates precisely characterize our knowledge forgetting semantics.

Theorem 1 *(**Representation theorem**) Let Γ and Γ' be two knowledge sets and $V \subseteq Atom$ a set of atoms. Then the following statements are equivalent:*

1. $\Gamma' \equiv \mathsf{KForget}(\Gamma, V)$;
2. $\Gamma' \equiv \{\phi \mid \Gamma \models \phi, IR(\phi, V)\}$;
3. *Postulates (**W**), (**PP**), (**NP**) and (**IR**) hold.*

Theorem 1 is significant in the sense that it provides an "if and only if" characterization on knowledge forgetting. That is, given a knowledge set Γ and a set of atoms V, an S5 formula Γ' represents a result of knowledge forgetting V from Γ if Γ' satisfies postulates (**W**), (**PP**), (**NP**) and (**IR**), and *vice versa*. We have observed that no such theorem has been proved in the previous work regarding propositional forgetting (Lang, Liberator, & Marquis 2003; Lin & Reiter 1994) and logic programming forgetting (Eiter & Wang 2008 to appear; Zhang & Foo 2006).

In order to prove Theorem 1, we need to first define the notion of *characteristic formula* that plays a central role in the proof of our representation theorem for knowledge forgetting. Intuitively, given a propositional interpretation π, π's characteristic formula $C(\pi, V)$ on a set V of atoms is a propositional formula which represents π by a restricted sources of atoms V. Formally, let π be an interpretation and V a set of atoms, the *characteristic formula* of π on V, denoted by $C(\pi, V)$, is defined as:

$$\bigwedge_{a \in \pi, a \in V} a \wedge \bigwedge_{b \notin \pi, b \in V} \neg b.$$

It is clear that $\pi \models C(\pi, V)$. Now we consider a k-interpretation $M = \langle W, w\rangle$ and a set V of atoms. Then the *characteristic formula* of M on V, denoted by $C(M, V)$, is defined as:

$$C(w, V) \wedge \bigwedge_{w' \in W} \neg K \neg C(w', V) \wedge \bigwedge_{\{\forall w'' \in 2^{Atom} \wedge w'' \notin W, \nexists w^* \in W s.t. w'' \leftrightarrow_{Atom \backslash V} w^*\}} K \neg C(w'', V).$$

It is not difficult to check that $M \models C(M, V)$. Moreover, $C(M, V)$ is irrelevant to $Atom \backslash V$, i.e. $IR(C(M, V), Atom \setminus V)$.

Proposition 3 *Let M and M' be two k-interpretations and V a set of atoms. $M' \models C(M, V)$ iff $M \leftrightarrow_{Atom \backslash V} M'$.*

Proof: Firstly, given two propositional interpretations π and π' and a set V of atoms, it is clear that $\pi' \models C(\pi, V)$ iff $\pi \leftrightarrow_{Atom \backslash V} \pi'$. Let $M = \langle W, s\rangle$ and $M' = \langle W', s'\rangle$. Now suppose that $M' \models C(M, Atom \backslash V)$. Then we have:

- $M' \models C(s, V)$, and hence $s' \models C(s, V)$. So $s \leftrightarrow_{Atom \backslash V} s'$.
- For all $w \in W$, $M' \models \neg K \neg C(w, V)$. That is, there exists some $w' \in W'$ such that $w' \models C(w, V)$. This follows $w \leftrightarrow_{Atom \backslash V} w'$.
- For all $w \notin W$ such that there does not exist $w_1 \in W$ and $w \leftrightarrow_{Atom \backslash V} w_1$, $M' \not\models \neg K \neg C(w, V)$. Thus, there does not exist $w' \in W'$ such that $w' \models C(w, V)$. That means that there does not exist w' such that $w \leftrightarrow_{Atom \backslash V} w'$.

Now we construct a binary relation $\sigma \subseteq W \times W'$ such that $\sigma(w, w')$ iff $w \leftrightarrow_{Atom \backslash V} w'$. Then we can see that $M \leftrightarrow_{Atom \backslash V} M'$ via the binary relation σ because:

1. $\sigma(s, s')$.
2. For all $w \in W$, there exists $w' \in W'$ such that $w \leftrightarrow_{Atom \backslash V} w'$.
3. For all $w' \in W'$, there exists $w \in W$ such that $w \leftrightarrow_{Atom \backslash V} w'$. Otherwise, $C(M, V) \models K \neg C(w', V)$. Thus, $M' \not\models M(C(w', V))$, a contradiction.
4. Finally, by the definition, for all pairs w, w' such that $\sigma(w, w')$, $w \leftrightarrow_{Atom \backslash V} w'$.

On the other hand, we suppose that $M \leftrightarrow_{Atom \backslash V} M'$ via the given binary relation σ above, then:

- We have that $s \leftrightarrow_{Atom \backslash V} s'$. Thus $s' \models C(s, V)$. Thus $M' \models C(s, V)$.
- For each $w \in W$, there exists $w' \in W'$ such that $\sigma(w, w')$ and $w \leftrightarrow_{Atom \backslash V} w'$. Thus, $w' \models C(w, V)$. Thus $M' \models \neg K \neg C(w, V)$.
- For each $w \notin W$ such that there does not exist $w_1 \in W$ and $w \leftrightarrow_{Atom \backslash V} w_1$. We have that $M' \models K \neg C(w, V)$. Otherwise, $M' \models \neg K \neg C(w, V)$. There exists $w' \in W'$ such that $w' \models C(w, V)$. Since $M \leftrightarrow_{Atom \backslash V} M'$, there exits $w_1 \in W$ such that $\sigma(w_1, w')$, which means that $w_1 \leftrightarrow_{Atom \backslash V} w'$. Thus $w_1 \models C(w, V)$, $w_1 \leftrightarrow_{Atom \backslash V} w$, a contradiction.

This shows that $M' \models C(M, V)$. □

Corollary 1 *Let M be a k-interpretation and V a set of atoms. $M \models C(M, V)$.*

Proof: This assertion follows directly from Proposition 3 since $M \leftrightarrow_{Atom \backslash V} M$. □

Proposition 4 *Let Γ be a knowledge base. Then*

$$\Gamma \equiv \bigvee_{M \in Mod(\Gamma)} C(M, Atom).$$

Proof: Let M be a k-model of Γ. By Corollary 1, $M \models C(M, Atom)$. Thus, M is also a k-model of $\bigvee_{M \in Mod(\Gamma)} C(M, Atom)$. On the other hand, suppose that M is a k-model of $\bigvee_{M \in Mod(\Gamma)} C(M, Atom)$. Then there is a $M' \in Mod(\Gamma)$ such that $M \models C(M', Atom)$. By Proposition 3, $M \leftrightarrow_{\emptyset} M'$. Therefore, M is also a k-model of Γ. □

Proof of Theorem 1

To prove $1 \Leftrightarrow 2$, We will show that

$$Mod(\mathsf{KForget}(\Gamma, V)) = Mod(\{\phi \mid \Gamma \models \phi, IR(\phi, V)\}) = Mod(\bigvee_{M \models \Gamma} C(M, Atom \backslash V)).$$

Firstly, suppose that M' is also a model of $\mathsf{KForget}(\Gamma, V)$. By Definition 2, there exists a k-interpretation M such that M is a k-mode of Γ and $M \leftrightarrow_V M'$. By Lemma 2, for all formula ϕ such that $\Gamma \models \phi$ and $IR(\phi, V)$, $M' \models \phi$. Thus, M' is a k-model of $\{\phi \mid \Gamma \models \phi, IR(\phi, V)\}$.

Secondly, suppose that M' is a k-model of $\{\phi \mid \Gamma \models \phi, IR(\phi, V)\}$. Thus,

$M' \models \bigvee_{M \in Mod(\Gamma)} C(M, Atom \backslash V)$,

since $\bigvee_{M \in Mod(\Gamma)} C(M, Atom \backslash V)$ is irrelevant to V.

Finally, suppose that M' is a k-model of $\bigvee_{M \in Mod(\Gamma)} C(M, Atom \backslash V)$. Then there exists $M \in Mod(\Gamma)$ such that $M' \models C(M, Atom \backslash V)$. By Proposition 3, $M \leftrightarrow_V M'$. Thus M' is also a k-model of $\mathsf{KForget}(\Gamma, V)$.

Now we show $3 \Rightarrow 2$. Suppose that all postulates hold. By Positive Persistence, $\Gamma' \models \{\phi \mid \Gamma \models \phi, IR(\phi, V)\}$. Now we show that $\{\phi \mid \Gamma \models \phi, IR(\phi, V)\} \models \Gamma'$. Otherwise, there exists formula ψ such that $\Gamma' \models \psi$ but $\{\phi \mid \Gamma \models \phi, IR(\phi, V)\} \not\models \psi$. There are three cases:

1. ψ is relevant to V. Thus, Γ' is also relevant to V, a contradiction to Irrelevance.
2. ψ is irrelevant to V and $\Gamma \models \psi$. This contradicts to our assumption.
3. ψ is irrelevant to V and $\Gamma \not\models \psi$. By Negative Persistence, $\Gamma' \not\models \psi$, a contradiction.

Thus, Γ' is equivalent to $\{\phi \mid \Gamma \models \phi, IR(\phi, V)\}$.

Finally $2 \Rightarrow 3$ is quite obvious. □

Corollary 2 *Let Γ be a knowledge base, ϕ a formula and V a set of atoms. $\mathsf{KForget}(\Gamma, V) \models \phi$ iff $IR(\phi, V)$ and $\Gamma \models \phi$.*

Other semantic properties

As we have mentioned in Introduction, the notion of forgetting has been defined and used in a variety of contexts under propositional logic (Lang, Liberator, & Marquis 2003; Lin & Reiter 1994). It is important to know the relationship between variable forgetting in propositional logic and knowledge forgetting in S5 propositional modal logic.

Let ϕ be an objective formula. We use $\phi[p/\bot]$ and $\phi[p/\top]$ to denote the formulas obtained from ϕ by replacing atom p with $\bot$ and $\top$ respectively. Then formula $\mathsf{Forget}(\phi, p)$ is obtained from ϕ by *forgetting* p from ϕ, if $\mathsf{Forget}(\phi, p) \equiv \phi[p/\bot] \vee \phi[p/\top]$. By forgetting a set of atoms V in ϕ, we recursively define $\mathsf{Forget}(\phi, V \cup \{p\}) = \mathsf{Forget}(\mathsf{Forget}(\phi, p), V)$, where $\mathsf{Forget}(\phi, \emptyset) = \emptyset$. We first have the following result.

Theorem 2 *Let ϕ be an objective formula and V a set of atoms. Then we have $\mathsf{KForget}(\phi, V) \equiv \mathsf{Forget}(\phi, V)$ and $\mathsf{KForget}(K\phi, V) \equiv K(\mathsf{Forget}(\phi, V))$.*

Proof: We prove Result 1 as follows. Suppose that $M = \langle W, w\rangle$ is a model of $KForget(\phi, V)$. Then there exists $M' = \langle W', w'\rangle \in Mod(\phi)$ such that $M' \leftrightarrow_V M$. Thus, $w' \leftrightarrow_V w$. Since $w' \models \phi$, $w \models Forget(\phi, V)$. Hence, M is also a model of $Forget(\phi, V)$. On the other hand, suppose that $M = \langle W, w\rangle$ is a model of $Forget(\phi, V)$. Then there exists $w' \models \phi$ and $w' \leftrightarrow_V w$. Construct a k-interpretation $M' = \langle W', w'\rangle$ such that $W' = W \cup \{w'\} \backslash \{w\}$. It's clear that M' is a model of ϕ and $M' \leftrightarrow_V M$. Hence, M is also a model of $KForget(\phi, V)$.

Now consider Result 2. Suppose that $M = \langle W, w\rangle$ is a model of $KForget(K\phi, V)$. Then there exists $M' = \langle W', w'\rangle \in Mod(K\phi)$ such that $M' \leftrightarrow_V M$. We have that for all $w'_1 \in W'$, $w'_1 \models \phi$. Since $M' \leftrightarrow_V M$, for all $w_1 \in W$, there exists $w'_1 \in W'$ such that $w_1 \leftrightarrow_V w'_1$. Therefore $w_1 \models Forget(\phi, V)$. This shows that $M \models K(Forget(\phi, V))$. On the other hand, suppose that $M = \langle W, w\rangle$ is a model of $K(Forget(\phi, V))$. Then for all $w_1 \in W$, $w_1 \models Forget(\phi, V)$, and there exists $w'_1 \models \phi$ and $w'_1 \leftrightarrow_V w$. Construct a k-interpretation $M' = \langle W', w'\rangle$ such that W' is the set of all w'_1 mentioned above and $w' \leftrightarrow_V w$. It's clear that M' is a model of $K(\phi)$ and $M' \leftrightarrow_V M$. Hence, M is also a model of $KForget(\phi, V)$. □

Theorem 2 simply reveals that propositional variable forgetting is a special case of knowledge forgetting, and also knowledge forgetting of formulas with the form $K\phi$ (where ϕ is objective) can be achieved through the corresponding propositional variable forgetting. However, the syntactic definition of variable forgetting $\mathsf{Forget}(\phi, p) \equiv \phi[p/\bot] \vee \phi[p/\top]$ cannot be extended to knowledge forgetting. Consider that we wan to knowledge forget atom p from formula $K(p \equiv q)$. According to Definition 1, we have $\mathsf{KForget}(K(p \equiv q), p) \equiv \top$, while $K(p \equiv q)[p/\bot] \vee K(p \equiv q)[p/\top] \equiv Kq \vee K\neg q$.

The following results further illustrate other essential semantic properties of knowledge forgetting.

Theorem 3 *Let Γ, Γ_1 and Γ_2 be three knowledge sets, ϕ_1 and ϕ_2 two formulas, and V a set of atoms. Then the following results hold:*

1. *$\mathsf{KForget}(\Gamma, V)$ is satisfiable iff Γ is satisfiable;*
2. *If $\Gamma_1 \equiv \Gamma_2$, then $\mathsf{KForget}(\Gamma_1, V) \equiv \mathsf{KForget}(\Gamma_2, V)$;*
3. *if $\Gamma_1 \models \Gamma_2$, then $\mathsf{KForget}(\Gamma_1, V) \models \mathsf{KForget}(\Gamma_2, V)$;*
4. *$\mathsf{KForget}(\phi_1 \vee \phi_2, V) \equiv \mathsf{KForget}(\phi_1, V) \vee \mathsf{KForget}(\phi_2, V)$;*
5. *$\mathsf{KForget}(\phi_1 \wedge \phi_2, V) \models \mathsf{KForget}(\phi_1, V) \wedge \mathsf{KForget}(\phi_2, V)$.*

Proof: To prove Result 1, suppose that M is a k-model of Γ. Then M is also a model of $\mathsf{KForget}(\Gamma, V)$. This shows that $\mathsf{KForget}(\Gamma, V)$ is satisfiable. On the other hand, suppose that Γ is unsatisfiable. Then, $Mod(\Gamma) = \emptyset$. It follows that $Mod(\mathsf{KForget}(\Gamma, V)) = \emptyset$.

Result 2 directly follows from Definition 1 and the fact $Mod(\Gamma_1) = Mod(\Gamma_2)$.

Now we prove Result 3. Suppose that M is a k-model of $\mathsf{KForget}(\Gamma_1, V)$, then there exists a k-model M' of Γ_1, V such that $M \leftrightarrow_V M'$. Since $\Gamma_1 \models \Gamma_2$, M' is also a k-model of Γ_2. Hence, M is a k-model of $\mathsf{KForget}(\Gamma_2, V)$ as well.

To prove Result 4, we need to show $Mod(\mathsf{KForget}(\phi_1 \vee \phi_2, V)) = Mod(\mathsf{KForget}\ (\phi_1, V) \vee \mathsf{KForget}(\phi_2, V))$. Suppose that M is a k-model of $\mathsf{KForget}(\phi \vee \psi, V)$, then there exists a k-model M_0 of $\phi \vee \psi$ such that $M_0 \leftrightarrow_V M$. Since M_0 is a k-model of $\phi \vee \psi$, M_0 is a k-model of ϕ or ψ. Without loss of generality, suppose that M_0 is a k-model of ϕ. We have that M is a k-model of $\mathsf{KForget}(\phi, V)$. Thus, M is a k-model of $\mathsf{KForget}(\phi, V) \vee \mathsf{KForget}(\psi, V)$.

On the other hand, suppose that M is a k-model of $\mathsf{KForget}(\phi, V) \vee \mathsf{KForget}(\psi, V)$, then M is a k-model of

$\mathsf{KForget}(\phi, V)$ or a k-model of $\mathsf{KForget}(\psi, V)$. Without loss of generality, suppose that M is a k-model of $\mathsf{KForget}(\phi, V)$, then there exists a model M_0 of ϕ such that $M_0 \leftrightarrow_V M$. M_0 is also a k-model of $\phi \vee \psi$. Thus, M is a k-model of $\mathsf{KForget}(\phi \vee \psi, V)$.

Finally we prove Result 5. Suppose that M is a k-model of $\mathsf{KForget}(\phi \wedge \psi, V)$, then there exists a k-model M_0 of $\phi \wedge \psi$ such that $M_0 \leftrightarrow_V M$. Therefore, M_0 is a k-model of ϕ. Thus, M is also a k-model of $\mathsf{KForget}(\phi, V)$. Similarly, M is a k-model of $\mathsf{KForget}(\psi, V)$ as well. □

In Theorem 3, we should note that the converse of Result 5 in Theorem 3 does not hold generally. For instance, let ϕ be $q \equiv p$; ϕ be $q \equiv r$. Then, $\mathsf{KForget}(\phi \wedge \psi, p)$ is equivalent to $q \equiv r$, while $\mathsf{KForget}(\phi, V) \wedge \mathsf{KForget}(\psi, V) \equiv \top$.

Computational properties

In this section, we will develop an algorithm to compute the syntactic representation of knowledge forgetting. We then prove the complexity results of major decision problems in relation to knowledge forgetting, and also characterize some tractable subclasses of knowledge forgetting problems.

An algorithm for computing knowledge forgetting

From a technical consideration, we first introduce modal operator B in our language. B is viewed as a dual modal operator of K, where $B\phi$ can be read as "the agent believes ϕ". Its semantics is defined as follows:

$\langle W, w\rangle \models B\phi$ iff $\exists w' \in W, \langle W, w'\rangle \models \phi$.

It is easy to see that $K\phi \equiv \neg B\neg\phi$. Then we extend the defintion of S5 formulas which may also contain modal operator B. Obviously, introducing modal operator B in the language does not affect the original semantics of finite propositional S5 modal logic.

Proposition 5 *Every S5 formula can be equivalently transformed into a formula without nested modal operators in polynomial time.*

Proof: By applying the following rules, we can transform each S5 formula into a form that only consists of propositional literals, connectives $\wedge$ and $\vee$ and modal operators K and B, and such transformation can be done in polynomial time. That is, we may remove negations from the original formula. These rules are:

1. $\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$.
2. $\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$.
3. $\neg K\phi \equiv B\neg\phi$.
4. $\neg B\phi \equiv K\neg\phi$.

Then we remove nested nested modal operators in the following way: for every formula obtained in the above step, it can be further transformed into a equivalent formula without nested modal operators according to the following transformation rules. Again, such transformation can be done in polynomial time. These rules are as follows:

1. $K(\phi \wedge \psi) \equiv K\phi \wedge K\psi$.
2. $B(\phi \vee \psi) \equiv B\phi \vee B\psi$.
3. $K(\phi \vee K\psi) \equiv K\phi \vee K\psi$.
4. $K(\phi \vee B\psi) \equiv K\phi \vee B\psi$.
5. $B(\phi \wedge K\psi) \equiv B\phi \wedge K\psi$.
6. $B(\phi \wedge B\psi) \equiv B\phi \wedge B\psi$.

□

Proposition 6 *Every S5 formula can be equivalently transformed into a disjunction of clauses of the following form:*

$$\phi_0 \wedge K\phi_1 \wedge B\phi_2 \wedge \ldots \wedge B\phi_n, \quad (1)$$

where all ϕ_i ($0 \leq i \leq n$) are propositional formulas, and any of ϕ_i may be absent.

Proof: By Proposition 5, every formula can be transformed into an S5 formula without nested modal operators. By replacing each modal sub-formula with a new atom, we can view this formula as a propositional formula. Then this formula can be transformed into a disjunctive normal form (DNF). Notice that every clause in the DNF is actually an S5 formula, which can be transformed into the form 1 using rules illustrated in the proof of Proposition 5. □

Now we propose the following algorithm to compute a syntactic representation of knowledge forgetting.

Algorithm 1
Input: an S5 formula ϕ; an atom p.
Output: the S5 formula obtained from ϕ by knowledge forgetting p.

1. transform ϕ into a set of clauses of form (1);
2. for each formula of the form:
   $\psi_0 \wedge K\psi_1 \wedge B\psi_2 \wedge \ldots \wedge B\psi_n$,
   obtained from Step 1, replace it with the following formula:
   $\mathsf{Forget}(\psi_0, p) \wedge K(\mathsf{Forget}(\psi_1, p)) \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_2, p)) \wedge \ldots \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_n), p)$.
3. **return** the disjunction of all formulas obtained from Step 2.

From Algorithm 1, we can see that knowledge forgetting can be precisely computed through propositional variable forgetting.

Example 2 Let $\Gamma \equiv Ka \wedge (Kb \vee \neg Kc)$ be a knowledge set. We consider to knowledge forget c from Γ by using Algorithm 1. First, Γ is transformed into the form of (1): $K(a \wedge b) \vee Ka \wedge B\neg c$. Then for each clause we do the transformation as showed in Step 2 in the algorithm. That is, clause $K(a \wedge b)$ is replaced by $K\mathsf{Forget}(a \wedge b, c)$, and clause $Ka \wedge B\neg c$ is replaced by $K\mathsf{Forget}(a, c) \wedge B\mathsf{Forget}(a \wedge \neg c, c)$, where $\mathsf{Forget}(a \wedge b, c) \equiv K(a \wedge b)$, and $K\mathsf{Forget}(a, c) \wedge B\mathsf{Forget}(a \wedge \neg c, c) \equiv Ka \wedge Ba \equiv Ka$. So by using Algorithm 1, we have $\mathsf{KForget}(\Gamma, c) \equiv K(a \wedge b) \vee Ka \equiv Ka$, which is also equivalent to the result obtained using Definition 1. □

Theorem 4 *Algorithm 1 is sound.*

Proof: By Result (5) in Theorem 3, we only need to prove that

$$\mathsf{KForget}(\psi_0 \wedge K\psi_1 \wedge B\psi_2 \wedge \ldots \wedge B\psi_n, p) \equiv \mathsf{Forget}(\psi_0, p) \wedge K(\mathsf{Forget}(\psi_1, p)) \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_2, p)) \wedge \ldots \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_n), p).$$

First suppose that $M = \langle W, s\rangle$ is a k-model of $\mathsf{KForget}(\psi_0 \wedge K\psi_1 \wedge M\psi_2 \wedge \ldots \wedge M\psi_n, p)$, then there exists a k-model $M_1 = \langle W_1, s_1\rangle$ of $\psi_0 \wedge K(\psi_1) \wedge B(\psi_2) \wedge \ldots \wedge B(\psi_n)$ such that $M_1 \leftrightarrow_{\{p\}} M$ via a binary relation σ. We will show that M is also a k-model of $\mathsf{Forget}(\psi_0, p)$, $K(\mathsf{Forget}(\psi_1, p))$, and $B(\mathsf{Forget}(\psi_1 \wedge \psi_i, p))$ $(2 \leq i \leq n)$ respectively. We consider the following cases.

Case 1. From Result (5) in Theorem 3, we know that $M \models \mathsf{KForget}(\psi_0, p)$. Since ψ_0 is a propositional formula, from Theorem 2, we have $M \models \mathsf{Forget}(\psi_0, p)$.

Case 2. $M_1 \models K(\psi_1)$. That is, for every $w_1 \in W_1$, $w_1 \models \psi_1$. On the other hand, for each $w \in W$, there exists $w_1 \in W_1$ such that $\sigma(w_1, w)$, i.e. $w_1 \leftrightarrow_{\{p\}} w$. So we have $w \models \mathsf{Forget}(\psi_1, p)$, which follows $M \models K(\mathsf{Forget}(\psi_1, p))$.

Case 3. For any i $(2 \leq i \leq n)$, $M_1 \models B(\psi_i)$. Thus, there exists some $w_1 \in W_1$ such that $w_1 \models \psi_i$. Moreover, $w_1 \models \psi_1 \wedge \psi_i$. On the other hand, there also exists some $w \in W$ such that $\sigma(w_1, w)$. That is, $w_1 \leftrightarrow_{\{p\}} w$. So we have $w \models \mathsf{Forget}(\psi_1 \wedge \psi_i, p)$, which follows $M \models B(\mathsf{Forget}(\psi_1 \wedge \psi_i, p))$.

Hence M is a k-model of $\mathsf{Forget}(\psi_0, p) \wedge K(\mathsf{Forget}(\psi_1, p)) \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_2, p)) \wedge \ldots \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_n), p)$.

Now we consider a k-model $M = \langle W, s\rangle$ of formula $\mathsf{Forget}(\psi_0, p) \wedge K(\mathsf{Forget}(\psi_1, p)) \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_2, p)) \wedge \ldots \wedge B(\mathsf{Forget}(\psi_1 \wedge \psi_n), p)$. For this purpose, we construct a k-interpretation $M_1 = \langle W_1, s_1\rangle$ and a binary relation σ simultaneously as follows:

(a) Since $M \models \mathsf{Forget}(\psi_0, p)$, $s \models \mathsf{Forget}(\psi_0, p)$. That is, $s \models \psi_0[p/\bot] \vee \psi_0[p/\top]$. Without loss of generality, suppose that $s \models \psi_0[p/\bot]$. Let s_1 be the interpretation obtained from s by assigning p to $\bot$. We have that $s_1 \models \psi_0$ and $s_1 \leftrightarrow_{\{p\}} s$. Let $\sigma(s_1, s)$.

(b) For each i $(2 \leq i \leq n$, $M \models B(\mathsf{Forget}(\psi_1 \wedge \psi_i, p))$. Thus, there exists some $w \in W$ such that $w \models \mathsf{Forget}(\psi_1 \wedge \psi_i, p)$. Similarly, we can construct a w_i such that $w_i \models \psi_1 \wedge \psi_i$ and $w_i \leftrightarrow_{\{p\}} w$. Let $w_i \in W_1$ and $\sigma(w_i, w)$.

(c) For each $winW$ such that for all i $(2 \leq ileqn)$ w does not satisfy $\mathsf{Forget}(\psi_1 \wedge \psi_i, \wedge\psi_i, p)$, we have that $w \models \mathsf{Forget}(\psi_1, p)$. Similarly, we can construct a w_j such that $w_j \models \psi_1$ and $w_j \leftrightarrow_{\{p\}} w$. Let $w_j \in W_1$ and $\sigma(w_j, w)$.

Then it is not difficult to verify that $M_1 \models \psi_0 \wedge K\psi_1 \wedge B\psi_2 \wedge \ldots \wedge B\psi_n$ and $M_1 \leftrightarrow_{\{p\}} M$. This shows that M is a k-model of $\mathsf{KForget}(\psi_0 \wedge K\psi_1 \wedge B\psi_2 \wedge \ldots \wedge B\psi_n, p)$. This completes our proof. □

From Proposition 2 in section 2, we know that forgetting a set V of atoms from a given knowledge set Γ can be sequentially computed by forgetting a single atom of V from Γ one by one using Algorithm 1. This means that Algorithm 1 can be used for a general knowledge forgetting computation.

It is not difficult to observe that the time complexity of Algorithm 1 is generally exponential in the size of input knowledge set Γ because transforming Γ into the form (1) (Step 1 in the algorithm) could be expensive. However, if the input knowledge set Γ is already in the form (1), computing knowledge forgetting of atom p from Γ can be done in polynomial time.

Main complexity results

In this subsection we will mainly study the complexity of three major decision problems in relation to knowledge forgetting: model checking, inference and irrelevance.

First we introduce some useful notions of complexity theory. Two basic complexity classes are P and NP. The class of P includes all decision problems solvable by a polynomial-time deterministic Turing machine, while the class of NP includes all decision problems solvable by a polynomial-time nondeterministic Turing machine. Suppose $\mathcal{C}$ is a class of decision problem. The class $\mathrm{P}^{\mathcal{C}}$ consists of the problems solvable by a polynomial-time deterministic Turing machine with an oracle for a problem from $\mathcal{C}$, and the class $\mathrm{NP}^{\mathcal{C}}$ includes the problems solvable by a nondeterministic Turing machine with an oracle for a problem in $\mathcal{C}$. By co-$\mathcal{C}$ we mean the class that consists of the complements of the problems in $\mathcal{C}$.

The classes Σ_k^{P} and Π_k^{P} of the *polynomial hierarchy* are defined as follows:

$\Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P}$, and

$\Sigma_k^{\mathrm{P}} = \mathrm{NP}^{\Sigma_{k-1}^{\mathrm{P}}}$, Π_k^{P} =co-Σ_k^{P} for all $k > 1$.

Note that NP=Σ_1^{P} and co-NP=Π_1^{P}.

Theorem 5 ***(Model checking complexity)*** *Let Γ be a knowledge set, V a set of atoms and M a k-interpretation. Deciding whether M is a k-model of $\mathsf{KForget}(\Gamma, V)$ is NP-complete.*

Proof: The problem can be determined by first guessing a k-interpretation M' polynomial in the size of M and then checking if $M' \models \Gamma$ and $M \leftrightarrow_V M'$. It is easy to see that the checking part can be done in polynomial time. Hence, the problem is in NP. For the hardness, we show that model checking for propositional variable forgetting is NP hard (considering that propositional variable forgetting is a special case of knowledge forgetting). Now we show that an objective formula ϕ is satisfiable iff $\pi \models \mathsf{Forget}(\phi, Var(\phi))$, where π is a propositional interpretation in which all atoms occurring in ϕ are assigned true. Firstly, if ϕ is satisfiable, then $\mathsf{Forget}(\phi, Var(\phi))$ becomes $\top$, which certainly should be satisfied in π. Second, if $\pi \models \mathsf{Forget}(\phi, Var(\phi))$, then from Result (1) in Theorem 3 we know that ψ must be satisfiable. This follows that the model checking for propositional variable forgetting is NP-hard. Consequently, the model checking for knowledge forgetting is also NP-hard. □

Now we consider two types of decision problems related to the inference of knowledge forgetting. Given a knowledge set Γ, an S5 formula ϕ and a set of atoms V, we like to decide: (1) whether $\mathsf{KForget}(\Gamma, V) \models \phi$; and (2) whether $\phi \models \mathsf{KForget}(\Gamma, V)$. Quite interestingly, our following result shows that these two types of inference have different complexity.

Theorem 6 *(**Inference complexity**) Let Γ be a knowledge set, ϕ a formula, and V a set of atoms. Then we have the results: (1) deciding whether $\mathsf{KForget}(\Gamma, V) \models \phi$ is co-NP complete; and (2) deciding whether $\phi \models \mathsf{KForget}(\Gamma, V)$ is Π_2^{P}-complete.*

Proof: For Result (1), the hardness is easy to see by setting $\mathsf{KForget}(\Gamma, Var(\Gamma))$. For membership, from Corollary 2, we have $\mathsf{KForget}(\Gamma, V) \models \phi$ iff $\Gamma \models \phi$ and $IR(\phi, V)$. Clearly, in single agent S5 modal logic, deciding $\Gamma \models \phi$ is in co-NP. We show that deciding whether $IR(\phi, V)$ is also in co-NP. Without loss of generality, we assume that ϕ is satisfiable. Then ϕ has a k-model in the polynomial size of ϕ. We consider the complement of the problem: deciding whether ϕ is not irrelevant to V. It is easy to see that ϕ is not irrelevant to V iff there exist a k-model M of ϕ and an k-interpretation M' in the polynomial size of ϕ such that $M \leftrightarrow_V M'$ and $M' \not\models \phi$. So checking whether ϕ is not irrelevant to V can be achieved in the following steps: (1) guess two k-interpretations M and M' in the polynomial size of ϕ, (2) check if $M \models \phi$ and $M' \not\models \phi$, and (3) check $M' \not\models \phi$. Obviously (1) can be done in polynomial time with a non-deterministic Turing machine while (2) and (3) can be done in polynomial time.

Now we consider Result (2). Membership. First it is well known that an S5 formula ψ is not valid iff there is a polynomial size k-interpretation M such that $M \not\models \psi$ (Fagin *et al.* 1995). We consider the complement of the problem. We may guess a polynomial size model M and check whether $M \models \phi$ and $M \not\models \mathsf{KForget}(\Gamma, V)$. From Theorem 5, we know that this is in Σ_2^{P}. So the original problem is in Π_2^P. Hardness. Let $\phi \equiv \top$. Then the problem is reduced to decide $\mathsf{KForget}(\Gamma, V)$'s validity. Since a propositional variable forgetting is a special case knowledge forgetting, the hardness is directly followed from the proof of Proposition 24 in (Lang, Liberator, & Marquis 2003). □

As we showed in Section 3, irrelevance plays an important role in characterizing knowledge forgetting. Recall that a formula ϕ is irrelevant to a set of atoms V if there is a ϕ' such that $\phi \equiv \phi'$ and $Var(\phi') \cap V = \emptyset$. In the following, we consider the complexity of deciding whether an S5 formula is irrelevant to a set of atoms V.

Theorem 7 *(**Irrelevance complexity**) Let ϕ be an S5 formula and V a set of atoms. Then deciding whether ϕ is irrelevant to V is co-NP complete.*

Proof: From the proof of Theorem 6, we know that deciding whether $IR(\phi, V)$ is in co-NP. For the hardness, we show that for any propositional formula ϕ, ϕ is valid iff $q \wedge (p \vee \phi)$ is irrelevant to p, where p, q are two different atoms not occurring in $Var(\phi)$. First, if ϕ is valid, then $q \wedge (p \vee \phi) \equiv q$ which is irrelevant to p. If $q \wedge (p \vee \phi)$ is irrelevant to p, again from Corollary 2, we know that $\mathsf{KForget}(q \wedge (p \vee \phi), p) \models q \wedge (p \vee \phi)$. Then from Theorem 2, we can derive $q \equiv q \wedge (p \vee \phi)$. This implies that ϕ must be valid. □

Conclusion

In this paper, we have developed a formal theory of knowledge forgetting under the propositional S5 single agent modal logic. We showed that knowledge forgetting is not only an important action that an agent may need in certain situations, but also a useful notion in representing various knowledge changes. Although the basic concept and intuition of knowledge forgetting were discussed in previous research. e.g. (Baral & Zhang 2005; Fagin *et al.* 1995), its formal semantics and computational properties have not been thoroughly studied. Our work presented in this paper provided a first formalization on knowledge forgetting and addressed its computational properties.

Some related issues remain for our further study. In this paper we only considered the problem of knowledge forgetting in a single agent S5 modal logic. In a multi-agent system, it is more common that an agent not only needs to forget his own knowledge due to a memory limit, but also has to forget other agents' knowledge for various reasons. So generalizing our knowledge forgetting to the mutli-agent S5 modal logic (and other multi-agent modal logics) will be a challenge. One particular concern we should take into account in this development is common knowledge which does not occur in single agent modal logic.

References

Baral, C., and Zhang, Y. 2005. Knowledge updates: Semantic and complexity issues. *Artificial Intelligence* 164:209–243.

Eiter, T., and Wang, K. 2008 (to appear). Semantic forgetting in answer set programming. *Artificial Intelligence*.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about Knowledge*. MIT Press.

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

Lin, F., and Reiter, R. 1994. Forget it! In *Working Notes of AAAI Fall Symposium on Relevance*, 154–159.

Lin, F. 2001. On the strongest necessary and weakest sufficient conditions. *Artificial Intelligence* 128:143–159.

Meyer, J.-J., and van der Hoek, W. 1995. *Epistemic Logic for AI and Computer Science*. Cambridge University Press.

Su, K.; Lv, G.; and Zhang, Y. 2004. Reasoning about knowledge by variable forgetting. In *Proceedings of the 9th Internaitonal Conference on Knowledge Representation and Reasoning (KR-2004)*, 576–586.

Winslett, M. 1988. Reasoning about action using a possible models approach. In *Proceedings of AAAI-88*, 89–93.

Zhang, Y., and Foo, N. 2006. Solving logic program conflict through strong and weak forgettings. *Artificial Intelligence* 170:739–778.

Embedding General Default Logic into the Logic of GK

Yi Zhou
School of Computing and Mathematics
UWS
yzhou@scm.uws.edu.au

Fangzhen Lin
Department of Computing Science
HKUST
flin@cs.ust.hk

Yan Zhang
School of Computing and Mathematics
UWS
yan@scm.uws.edu.au

Abstract

In this paper, we show that the logic of GK is indeed a general framework for nonmonotonic reasoning by embedding general default logic into it. More importantly, we illustrate that it is also a powerful tool for studying nonmonotonic formalisms. We first show that checking for weak equivalence and strong equivalence between two rule bases in general default logic can both be captured in the logic of GK and the complexities for both problems are coNP complete. Then, we show that each rule base is strongly equivalent to a set of rules of normal form. Finally, we prove that auto-epistemic logic is equivalent to a proper subset of general default logic, and the self-introspection operator in auto-epistemic logic indeed plays the same role as the double negation-as-failure operator in general default logic.

Introduction

Recently, Zhou *et al.* (2007) proposed a nonmonotonic logic, called general default logic, which extended both Gelfond *et al.*'s disjunctive default logic (1991) (therefore Reiter's default logic (1980)) by allowing arbitrary nested rule connectives and Ferraris's general logic programming (2005a) by allowing classical connectives.

However, many properties in relation to general default logic remain unclear, for instance, whether it is a strict extension of disjunctive default logic (or Reiter's default logic). In this paper, we aim to study some fundamental properties of general default logic by embedding it into the logic of GK (Lin & Shoham 1992).

The logic of knowledge and justified assumptions (the logic of GK for short), proposed by Lin and Shoham (1992), is a non-standard modal logic with two modal operators K (for knowledge) and A (for assumptions). The logic of GK is a general framework for nonmonotonic formalisms. It has been shown in the early 1990s that both Reiter's default logic (1980) and Moore's auto-epistemic logic (1987) can be embedded into the logic of GK. Recently, Lin and Zhou (2007) showed that Ferraris's general logic programming (2005a) can be embedded into the logic of GK as well.

A question naturally arises whether general default logic can also be embedded into the logic of GK. This paper answers it positively. In section 3, we shall show that the logic of GK is flexible enough to capture general default logic as well. Thus, our result confirms the generality of the logic of GK.

More importantly, we shall also show that the logic of GK is not only a general framework of nonmonotonic logics but also a powerful tool to study them for three reasons. Firstly, although the logic of GK is a non-standard logic, it is based on a standard bi-modal logic. There are a lot of useful existing techniques for modal logics, for instance, the standard complexity analysis techniques (Halpern & Moses 1992). As an application of this aspect, in Section 4, we shall first show that both weak equivalence and strong equivalence in general default logic can be captured in the logic of GK, and then show that the complexity of checking weak equivalence and strong equivalence between two rules are both coNP complete.

Secondly, there are many useful properties which can be used in standard modal logic. For instance, $K(P) \wedge K(Q)$ is equivalent to $K(P \wedge Q)$. As an application of this aspect, in Section 5, we shall show that each rule base in general default logic is strongly equivalent to a set of rules of the following form:

$$C_1 \,\&\, \ldots \,\&\, C_n \,\&\, -C_{n+1} \,\&\, \ldots \,\&\, -C_m \Rightarrow C_{m+1} \mid \ldots \mid C_k \mid -C_{k+1} \mid \ldots \mid -C_l, \qquad (1)$$

where $C_i, (1 \leq i \leq l)$ are propositional clauses. Moreover, this form cannot be further transformed into disjunctive default rules. This shows that, to some extent, general default logic is a strict extension of disjunctive default logic (thus Reiter's default logic).

Finally, as we mentioned earlier, the logic of GK is a general framework of nonmonotonic logics. Thus, it can serve as a platform for comparing different nonmonotonic formalisms. As an application of this aspect, in Section 6, we shall show that Moore's auto-epistemic logic (1987) is equivalent to a proper subclass of general default logic. As a consequence, the self-introspection operator in auto-epistemic logic indeed plays the same role as the double negation-as-failure operator in general default logic. Furthermore, as Gottlob (1995) showed that auto-epistemic logic can not be embedded into Reiter's default logic by a modular translation, this result also confirms that general default logic is a strict extension of Reiter's default logic.

Preliminaries

We begin by recalling the basic concepts of propositional logic, general default logic and the logic of GK.

The classical propositional language $\mathcal{L}$ is defined recursively by a set $Atom$ of atoms and a set of classical connectives $\bot$, $\neg$ and $\rightarrow$ as follows:

$$F ::= \bot \mid p \mid \neg F \mid F \rightarrow F,$$

where p is an atom. $\top$, $\wedge$, $\vee$ and $\leftrightarrow$ are defined as usual. Formulas in $\mathcal{L}$ are called *facts*. A *theory* T is a set of facts which is closed under classical entailment. Let Γ be a set of facts, we write $Th(\Gamma)$ to denote the logical closure of Γ under classical entailment. We write Γ to represent the theory $Th(\Gamma)$ if clear from the context. A theory T is *inconsistent* if there is a fact F such that $F \in T$ and $\neg F \in T$, otherwise T is *consistent*. *Literals* are atoms and their negations. *Clauses* are disjunctions of sets of literals.

General default logic

The propositional rule language $\mathcal{R}$ (Zhou, Lin, & Zhang 2007) is defined upon $\mathcal{L}$ by adding a set of *rule connectives* $\Rightarrow$ (for *rule implication*), $\&$ (for *rule and*) and $\mid$ (for *rule or*):

$$R ::= F \mid R \Rightarrow R \mid R \,\&\, R \mid R \mid R,$$

where F is a fact. $-R$ and $R \Leftrightarrow S$ are considered as shorthands of $R \Rightarrow \bot$ and $(R \Rightarrow S)\ \&\ (S \Rightarrow R)$ respectively. $-$ is for *negation as failure* (or *rule negation*); $\Leftrightarrow$ is for *rule equivalence*. Formulas in $\mathcal{R}$ are called *rules*. A *rule base* Δ is a set of rules.

The *satisfaction relation* $\models$[1] between theories and rules is defined recursively as follows:

- If R is a fact, then $T \models R$ iff R is entailed by T in classical propositional logic.
- $T \models R \,\&\, S$ iff $T \models R$ and $T \models S$;
- $T \models R \mid S$ iff $T \models R$ or $T \models S$;
- $T \models R \Rightarrow S$ iff $T \not\models R$ or $T \models S$.

Thus, if T is consistent, then $T \models -R$ iff $T \not\models R$. If T is inconsistent, then for every rule R, $T \models R$. We say that T *satisfies* R, or T is a *model* of R iff $T \models R$.

The *subrule* relationship between two rules is defined recursively as follows:

- R is a subrule of R;
- R and S are subrules of $R \Rightarrow S$, $R \,\&\, S$ and $R \mid S$,

where R and S are rules.

The *reduct* of a rule R relative to a theory T, denoted by R^T, is the rule obtained from R by replacing each maximal subrule of R which is not satisfied by T with $\bot$. Let T be a theory and Δ a rule base, the reduct of Δ relative to T, denoted by Δ^T, is the set of all the reducts of rules in Δ relative to T.

[1] In this paper, we overload the notation $\models$. We use it to denote the satisfaction relation in classical logic, general default logic, the logic of GK and auto-epistemic logic. Which one it stands for should be clear from the context.

Definition 1 (extensions) *Let T be a theory and Δ a rule base. We say that T is an extension of Δ iff:*

1. $T \models \Delta^T$.
2. *There is no theory T_1 such that $T_1 \subset T$ and $T_1 \models \Delta^T$.*

We say that two rules are *weakly equivalent* if they have the same set of models. The notion of weak equivalence, introduced in (Zhou, Lin, & Zhang 2007), plays an important role in computing extensions and simplifying default rules after reduction. Notice that weak equivalence is not as the same as equivalence defined for default logic as usual. Two rule bases are said to be *equivalent* iff they have the same set of extensions. For instance, $-p \Rightarrow q$ is equivalent to q since both of them have a unique extension $Th\{q\}$. However, they are not weakly equivalent. On the other hand, $-p \Rightarrow q$ and $p \mid q$ are weakly equivalent but not equivalent.

Zhou *et al.* (2007) showed that Reiter's default logic (Reiter 1980) in propositional case is a special case of general default logic by restricting the rules of the following form

$$F \,\&\, -G_1 \,\&\, \ldots \,\&\, -G_n \Rightarrow H,$$

where $n \geq 0$, F, $G_i, (1 \leq i \leq n)$ and H are facts, and F may be absent. Yet, under the context of Reiter's default logic, this form is represented as

$$F : M(\neg G_1), \ldots, M(\neg G_n)/H.$$

They also showed that Ferraris's general logic programming (Ferraris 2005a) is a special case of general default logic by restricting the facts occurred in rules with atoms. However, under the context of general logic programming, rule connectives are represented by corresponding classical connectives.

The logic of GK

The language $\mathcal{L}_{\mathcal{GK}}$ of the logic of GK (Lin & Shoham 1992) is extended from $\mathcal{L}$ with two modal operators K (for knowledge) and A (for assumption). Formulas in $\mathcal{L}_{\mathcal{GK}}$ are defined recursively as follows:

$$F ::= \bot \mid p \mid \neg F \mid F \rightarrow F \mid K(F) \mid A(F),$$

where $p \in Atom$. $\top$, $\wedge$, $\vee$ and $\leftrightarrow$ are defined the same as in the classical modal logic. Formulas in $\mathcal{L}_{\mathcal{GK}}$ are called *GK formulas*. Formulas constructed from $K(F)$ and $A(F)$, where F is a fact, and the connectives $\bot$, $\neg$ and $\rightarrow$ are called *subjective formulas*. In other words, subjective formulas are those GK formulas without nested modal operator and each atom p has to be in the scope of a modal operator.

A *Kripke interpretation* M is a tuple $\langle W, \pi, R_K, R_A, s\rangle$, where W is a nonempty set, called the set of *possible worlds*, π a function that maps $Atom$ to the power set of W, R_K and R_A binary relations on W, which represent the accessibility relations for K and A respectively, and $s \in W$, called the *actual world* of M. The *satisfaction relation* $\models$ between Kripke interpretations and GK formulas is defined inductively as follows:

- $M \not\models \bot$;
- If $p \in Atom$, $M \models p$ iff $s \in \pi(p)$;

- $M \models \neg F$ iff $M \not\models F$;
- $M \models F \rightarrow G$ iff $M \not\models F$ or $M \models G$;
- $M \models K(F)$ iff $\langle W, \pi, R_K, R_A, w\rangle \models F$ for any $w \in W$, such that $(s, w) \in R_K$;
- $M \models A(F)$ iff $\langle W, \pi, R_K, R_A, w\rangle \models F$ for any $w \in W$, such that $(s, w) \in R_A$.

We say that a Kripke interpretation M *satisfies* a GK formula F, or M is a *model* of F iff $M \models F$. We say that two GK formulas are *equivalent* in the logic of GK if they have the same set of models.

Let

$$K(M) = \{F \mid F \text{ is a fact and } M \models K(F)\}$$
$$A(M) = \{F \mid F \text{ is a fact and } M \models A(F)\}.$$

It is clear that both $K(M)$ and $A(M)$ are theories.

Definition 2 (GK Models) *Let M be an interpretation and F a formula. We say that M is a minimal model of F if*

1. *M is a model of F;*
2. *there is no interpretation M_1 such that M_1 is also a model of F and $A(M_1) = A(M)$, $K(M_1) \subset K(M)$.*

We say that M is a GK model *if M is a minimal model of F and $K(M) = A(M)$.*

Embedding General Default Logic into the Logic of GK

In this section, we show that general default logic can be embedded into the logic of GK as well.

Let R be a rule in $\mathcal{R}$. By R_A we denote the GK formula obtained from R by adding a modal operator A in front of every fact and then replacing all occurrences of rule connectives with corresponding classical connectives. By R_{GK} we denote the GK formula obtained from R recursively as follows:

- If R is a fact, then $R_{GK} = K(R)$.
- If R is $F \,\&\, G$, then R_{GK} is $F_{GK} \wedge G_{GK}$.
- If R is $F \mid G$, then R_{GK} is $F_{GK} \vee G_{GK}$.
- If R is $F \Rightarrow G$, then R_{GK} is $(F_{GK} \rightarrow G_{GK}) \wedge (F_A \rightarrow G_A)$.

Thus, if R is $-F$, then R_{GK} is $(F_{GK} \rightarrow \bot) \wedge (F_A \rightarrow \bot)$, which is equivalent to $\neg F_{GK} \wedge \neg F_A$; if R is $F \Leftrightarrow G$, then R_{GK} is equivalent to $(F_{GK} \leftrightarrow G_{GK}) \wedge (F_A \leftrightarrow G_A)$.

For every rule R, it is clear that both R_A and R_{GK} are well defined subjective formulas in $\mathcal{L}_{\mathcal{GK}}$. Let Δ be a rule base. By Δ_{GK} we denote the set of GK formulas:

$$\Delta_{GK} = \{R_{GK} \mid R \in \Delta\}.$$

As general logic programming is a special case of general default logic, this mapping is a natural generalization of the translation from general logic programming into the logic of GK proposed in (Lin & Zhou 2007).

Example 1 Let R be the rule $-(p \vee q) \Rightarrow \neg p \mid q$. Then R has a unique extension $\{\neg p\}$. On the other hand, R_{GK} is

$$((-p \vee q)_{GK} \rightarrow (\neg p)_{GK} \vee q_{GK}) \wedge ((-p \vee q)_A \rightarrow (\neg p)_A \vee q_A),$$

which is which is equivalent to

$$(K(p \vee q) \vee K(p \vee q) \vee K\neg p \vee Kq) \wedge (A(p \vee q) \vee A\neg p \vee Aq),$$

which has a unique (in the sense that two interpretations M_1 and M_2 such that $K(M_1) = K(M_2)$ and $A(M_1) = A(M_2)$ are the same) GK model M such that $K(M) = A(M) = Th(\{\neg p\})$.

We shall show that general default logic can be embedded into the logic of GK with this mapping. We first present the following lemma.

Lemma 1 [2] *Let R be a rule and M an interpretation such that $K(M) = T_1$ and $A(M) = T_2$. $T_1 \models R^{T_2}$ iff M is a model of R_{GK}.*

The following theorem shows that general default logic can be embedded into the logic of GK with this mapping.

Theorem 2 *Let Δ be a rule base and T a consistent theory. T is an extension of Δ iff there is a GK model M of Δ_{GK} such that $K(M) = A(M) = T$.*

The translation is not surprising since there are a large number of similar translations (Lin & Shoham 1992; Lin 2002; Lin & Zhou 2007; Pearce, Tompits, & Woltran 2001; Ferraris, Lee, & Lifschitz 2007) from answer set programming or default logic into other nonmonotonic logics. Lin and Shoham (1992) translated Reiter's default logic into the logic of GK. As a special case, Lin (2002) translated normal logic programming into the logic of GK and showed that checking strong equivalence between two normal logic programs can be reduced into classical propositional logic. Lin and Zhou (2007) extended it into general logic programming and also lifted it into first order case. Pearce et al. (2001) showed that answer set programming can be translated into QBF. Ferraris et al. (2007) translated a first order sentence to second order one and treated it as the answer set semantics for first order logic programs.

Of course, these translations work very well. However, the intuitions behind them still remain unclear. Consider another translation from general default logic into the logic of GK as follows:

- If R is a fact, then $R_{GK'} = K(R) \wedge A(R)$.
- If R is $F \,\&\, G$, then $R_{GK'}$ is $(F_{GK'} \wedge G_{GK'}) \wedge (F_A \wedge G_A)$.
- If R is $F \mid G$, then $R_{GK'}$ is $(F_{GK'} \vee G_{GK'}) \wedge (F_A \vee G_A)$.
- If R is $F \Rightarrow G$, then $R_{GK'}$ is $(F_{GK'} \rightarrow G_{GK'}) \wedge (F_A \rightarrow G_A)$.

Thus, $(-F)_{GK'}$ is $\neg F_{GK'} \wedge \neg F_A$.

Proposition 3 $\bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F) \models R_{GK} \leftrightarrow R_{GK'}$.

Corollary 4 *Let R be a rule. R_{GK} and $R_{GK'}$ have the same set of GK models.*

[2] See the appendix for some of the proof sketches.

According to this reformulation, the intuitions behind our translation are very clear. Roughly speaking, each rule connective is transformed into the classical conjunction of two parts of corresponding classical connective. Then, a fixed point semantics (here is the logic of GK) is used.

Another related result is due to Truszczyński (2007). He showed that (disjunctive) default logic can be embedded into modal default theories in S4F. Clearly, his work can be applied to general default logic as well. Lifschitz (1991) also introduced another nonmonotonic modal logic, called MKNF, and translated Reiter's default logic into it.

The topic of which nonmonotonic modal logic (GK, S4F, MKNF) is more interesting is beyond the scope of this paper. Here, we are not intending to argue which one is better since each has its own merits. For example, S4F contains only one modal operator and then seems more natural, whilst MKNF allows first order components, which can be a basis for first order nonmonotonic logics (Motik & Rosati 2007). On the other hand, the logic of GK is based on a standard bi-modal language, which is simple and well studied. Since there are many useful techniques and properties for standard modal logics, we can take advantages of them. As an example, in the next two sections, we study weak equivalence, strong equivalence, and normal forms in general default logic via the logic of GK.

Weak Equivalence, Strong Equivalence and Complexity Issues

In this section, we first show that the satisfiability problem for subjective formulas in the logic of GK is NP complete. Then we show that checking weak equivalence and strong equivalence can both be reduced into checking the validity of a certain subjective formula in the logic of GK.

Ladner (1977) showed that every satisfiable $S5$ formula F must have a model polynomial in the length of F. Then he proved that the satisfiability problem for $S5$ is NP complete. Halpern and Moses (1992) proved the same result for $KD45$. Thus, the satisfiability problem for $KD45$ is also NP complete.

Here, we prove a similar result for subjective formulas in the logic of GK. All the complexity results addressed in this section mainly follow from the following proposition.

Proposition 5 *Let F be a subjective formula in $\mathcal{L}_{\mathcal{GK}}$. F is satisfiable iff F has a model with at most $2|F|+1$ possible worlds, where $|F|$ is the length of the formula F.*

Notice that Proposition 5 only holds for subjective formulas in L_{GK}. It does not hold in the general case since the language L_{GK} is a standard modal logic language with two modal operators, whose satisfiability problem is generally beyond NP complete (Halpern & Moses 1992).

Similar to the NP completeness proof of satisfiability of $S5$ (Ladner 1977) and $KD45$ (Halpern & Moses 1992), we have the following result.

Corollary 6 *The complexity of checking whether a subjective formula is satisfiable is NP complete.*

The following theorem shows that checking weak equivalence between two rules can be captured in the logic of GK.

Theorem 7 *Let R_1 and R_2 be two rules. R_1 and R_2 are weakly equivalent iff $(R_1)_A$ and $(R_2)_A$ are equivalent in the logic of GK.*

Thus, checking weak equivalence can be reduced into checking the validity of a subjective formula in the logic of GK.

Corollary 8 *Checking whether two rules are weakly equivalent is in coNP.*

The notion of strong equivalence, proposed by Lifschitz *et al.* (2001) for logic programs, plays a crucial role in answer set programming. Lin and Zhou (2007) showed that checking strong equivalence between two general logic programs can be captured in the logic of GK. The notion of strong equivalence is introduced into default logic by Turner (2001). Here, we show that the strong equivalence relationship between two rules can be captured in the logic of GK as well.

We say that two rules R_1 and R_2 are *strongly equivalent*, denoted by $R_1 \equiv R_2$, if for every rule R_3, R_1 & R_3 has the same set of extensions as R_2 & R_3.

Given a rule R, we specify

$$Fact(R) = \{F \mid F \text{ is a fact, } F \text{ is a subrule of } R\}.$$

Theorem 9 *Let R_1 and R_2 be two rules. The following four statements are equivalent:*

1. *R_1 and R_2 are strongly equivalent.*
2. *$\bigwedge_{F\in Fact(R_1\ \&\ R_2)} K(F) \rightarrow A(F) \models (R_1)_{GK} \leftrightarrow (R_2)_{GK}$.*
3. *$\bigwedge_{F\in\mathcal{L}} K(F) \rightarrow A(F) \models (R_1)_{GK} \leftrightarrow (R_2)_{GK}$.*
4. *For every rule R_3 such that R_1 is a subrule of it, and R_4 be the rule obtained from R_3 by replacing each occurrence of R_1 into R_2, R_3 has the same set of extensions as R_4.*

Theorem 7 and Theorem 9 are convenient for checking whether or not two rules are weak equivalent or strongly equivalent.

Example 2 Let $-p \Rightarrow q$ and $p \,|\, q$ be two rules. We have that $(-p \Rightarrow q)_A$ is $\neg Ap \rightarrow Aq$, which is equivalent to $Ap \vee Aq$. On the other hand, $(p \mid q)_A$ is $Ap \vee Aq$. This shows that $-p \Rightarrow q$ and $p \mid q$ are weakly equivalent. However, $(-p \Rightarrow q)_{GK}$ is $((\neg Ap \wedge \neg Kp) \rightarrow Kq) \wedge (\neg Ap \rightarrow Aq)$, which is equivalent to $Ap \vee Kq$ under $\bigwedge_{F\in\mathcal{L}} K(F) \rightarrow A(F)$. On the other hand, $(p \mid q)_{GK}$ is $Kp \mid Kq$. Obviously, they are not equivalent in the logic of GK. Thus, $-p \Rightarrow q$ and $p \mid q$ are not strongly equivalent.

Theorem 9 shows that checking whether two rules are strongly equivalent can be reduced into checking whether a certain GK formula is valid. Since this GK formula (the formula in Condition 2, Theorem 9) is exactly a subjective formula and polynomial in the length of these two rules, we have the following result.

Corollary 10 *Checking whether two rules are strongly equivalent in general default logic is in coNP.*

Finally, we show that both checking weak equivalence and checking strong equivalence between two rules are coNP hard by the following lemma.

Lemma 11 *Let F and G be two facts. F is equivalent to G in classical propositional logic iff F is weakly equivalent to G iff F is strongly equivalent to G.*

Theorem 12 *The complexity of checking both whether two rules are strongly equivalent and whether two rules are weakly equivalent are coNP complete.*

The notion of strong equivalence can be extended for rule bases. Given two rule bases Δ_1 and Δ_2, we say that Δ_1 is *strongly equivalent* to Δ_2 if for every rule base Δ_3, $\Delta_1 \cup \Delta_3$ has the same set of extensions as $\Delta_2 \cup \Delta_3$. Obviously, Theorem 7, 9 and 12 holds for rule bases as well.

Truszczyński (2007) showed that strong equivalence between default theories can be captured in S4F according to his translation. However, the complexity issue is not addressed in his approach.

Normal Forms of General Default Logic

In this section, we show that each rule base can be strongly equivalently transformed into a set of rules of form (1). The key technique of proving this is Theorem 9.

By Theorem 9, we have that

Proposition 13 *Let F and G be two facts. $F \wedge G \equiv F \,\&\, G$.*

Corollary 14 *Each rule R is strongly equivalent to a rule R_1 such that $Fact(R_1)$ is a set of clauses.*

Proposition 13 also indicates that the two connectives $\&$ and $\wedge$ coincide with each other to some extent.

Furthermore, the following proposition describes more strongly equivalent transformations.

Proposition 15 *For any rules F, G, H and R,*

1. *$F \,\&\, G$ is strongly equivalent to $\{F, G\}$.*
2. *$-\bot \equiv \top$, $-\top \equiv \bot$*
3. *$F \,\&\, \bot \equiv \bot$, $F \mid \bot \equiv F$;*
4. *$F \,\&\, \top \equiv F$, $F \mid \top \equiv \top$;*
5. *$F \,\&\, G \equiv G \,\&\, F$, $F \mid G \equiv G \mid F$;*
6. *$F \,\&\, (G \,\&\, H) \equiv (F \,\&\, G) \,\&\, H$, $F \mid (G \mid H) \equiv (F \mid G) \mid H$;*
7. *$F \,\&\, (G \mid H) \equiv (F \,\&\, G) \mid (F \,\&\, H)$, $F \mid (G \,\&\, H) \equiv (F \mid G) \,\&\, (F \mid H)$.*
8. *$-(F \,\&\, G) \equiv -F \mid -G$, $-(F \mid G) \equiv -F \,\&\, -G$;*
9. *$---F \equiv -F$;*
10. *$-(F \Rightarrow G) \equiv --F \,\&\, -G$;*
11. *$(F \mid G) \Rightarrow H \equiv (F \Rightarrow H) \,\&\, (G \Rightarrow H)$;*
12. *$F \Rightarrow (G \,\&\, H) \equiv (F \Rightarrow G) \,\&\, (F \Rightarrow H)$;*
13. *$(F \Rightarrow G) \mid H$ is strongly equivalent to $\{F \Rightarrow G \mid H, H \mid -F \mid --G\}$;*
14. *$(F \Rightarrow G) \,\&\, R \Rightarrow H$ is strongly equivalent to $\{G \,\&\, R \Rightarrow H, R \Rightarrow F \mid H \mid -G, R \Rightarrow H \mid --F\}$;*
15. *$F \Rightarrow (G \Rightarrow H)$ is strongly equivalent to $\{F \,\&\, G \Rightarrow H, F \Rightarrow -G \mid --H\}$;*
16. *$F \,\&\, --G \Rightarrow H \equiv F \Rightarrow H \mid -G$;*
17. *$F \Rightarrow G \mid --H \equiv F \,\&\, -H \Rightarrow G$.*

By Corollary 14 and 1-9 in Proposition 15, we have the following proposition.

Proposition 16 *Each rule base without $\Rightarrow$ and $\Leftrightarrow$ is strongly equivalent to a set of rules of the following form*

$$C_1 | \ldots | C_n | -C_{n+1} | \ldots | -C_m | --C_{m+1} | \ldots | --C_k, \quad (2)$$

where $C_i, (1 \leq i \leq k)$ are propositional clauses.

Proposition 17 *Each rule of the form $-R$ is strongly equivalent to a set of rules of the following form*

$$-C_1 \mid \ldots \mid -C_n \mid --C_{n+1} \mid \ldots \mid --C_m, \quad (3)$$

where $C_i, (1 \leq i \leq m)$ are propositional clauses.

By Proposition 15 and Proposition 16,

Theorem 18 *Each rule base is strongly equivalent to a set of rules of the following form:*

$$C_1 \,\&\, \ldots \,\&\, C_n \,\&\, -C_{n+1} \,\&\, \ldots \,\&\, -C_m \Rightarrow C_{m+1} \mid \ldots \mid C_k \mid -C_{k+1} \mid \ldots \mid -C_l,$$

where $C_i, (1 \leq i \leq l)$ are propositional clauses.

Notice that form (1) can not further be strongly equivalently transformed into a set of disjunctive default rules in (Gelfond *et al.* 1991). A simple example is $p \mid -p$, where p is an atom. This, to some extent, indicates that general default logic is more expressive than disjunctive default logic (thus Reiter's default logic).

As stated in (Zhou, Lin, & Zhang 2007), general logic programming (Ferraris 2005a) is a special case of general default logic by restricting facts into atoms, Theorem 18 can be viewed as a generalization of recent work of Cabalar and Ferraris (2007), who proved that each general logic program can be strongly equivalently transformed into a set of extended disjunctive rules. Moreover, Theorem 18 also indicates that general answer set programming with classical negation also has a similar normal form result.

On the Relationships between Default Logics and Auto-epistemic Logic

Auto-epistemic is another dominant formalism for nonmonotonic reasoning. The relationships between default logic and nonmonotonic reasoning is, of course, one of the most important topics in nonmonotonic reasoning.

We first briefly introduce Moore's auto-epistemic logic (Moore 1987) and related issues. The language $\mathcal{L}_{\mathcal{AEL}}$ of auto-epistemic logic is extended from $\mathcal{L}$ with a modal operator L for self introspection. Formulas in $\mathcal{L}_{\mathcal{AEL}}$ are defined recursively as follows:

$$F ::= \bot \mid p \mid \neg F \mid F \rightarrow F \mid L(F),$$

where $p \in Atom$. $\top$, $\wedge$, $\vee$ and $\leftrightarrow$ are defined as usual. Formulas in $\mathcal{L}_{\mathcal{AEL}}$ are called *AEL formulas*.

Let Γ be a set of AEL formulas. A set E of AEL formulas is a *stable expansion* of Γ if:

$$E = Th(\Gamma \cup \{L(F) \mid F \in E\} \cup \{\neg L(F) \mid F \notin E\}).$$

A stable expansion is uniquely determined by the set of propositional formulas in it. This is called the *kernel* of a stable expansion (Konolige 1988). Hence, we can identify a stable extension with its kernel, which is obviously a propositional theory.

Konolige (1988) proved that for every set Γ of AEL formulas, there is a set Γ' of AEL formulas of the following form

$$\neg L(F) \vee L(G_1) \vee \ldots \vee L(G_n) \vee H \quad (4)$$

such that Γ' has the same set of stable expansions with Γ, where $n \geq 0$, F, $G_i, (1 \leq i \leq n)$ and H are facts, F may be absent. Based on Konolige's result, Lin and Shoham (1992) showed that Moore's auto-epistemic logic can be embedded into the logic of GK by translating each AEL formula of the form (4) into

$$\neg A(F) \vee A(G_1) \vee \ldots \vee A(G_n) \vee K(H).$$

Theorem 19 (Lin and Shoham (1992)) *Let Γ be a set of AEL formulas with form (4). A theory T is the kernel of a stable expansion of a set Γ of AEL formulas iff there is a GK model M of Γ_{GK} such that $K(M) = T$.*

Now we show that auto-epistemic logic can be embedded into general default logic via the logic of GK. Without loss of generality, we consider AEL formulas of form (4). Let S be an AEL formula of the form

$$\neg L(F) \vee L(G_1) \vee \ldots \vee L(G_n) \vee H.$$

By $\Theta(S)$ we denote the following rule

$$---F \mid --G_1 \mid \ldots \mid --G_n \mid H.$$

A set Γ of AEL formulas with form (4) is translated into the rule base $\Theta(\Gamma) = \{\Theta(S) \mid S \in \Gamma\}$.

It is easy to see that $(\Theta(S))_{GK}$ is equivalent to S_{GK} under $\bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F)$ in the logic of GK. Therefore, the following result follows directly from Theorem 2 and 19.

Theorem 20 *A theory T is the kernel of a stable expansion of a set Γ of AEL formulas iff T is an extension of $\Theta(\Gamma)$.*

On the other hand, suppose that $\mathcal{R}^{\mathcal{A}}$ is the subclass of $\mathcal{R}$ such that each rule in $\mathcal{R}^{\mathcal{A}}$ is a set of rules of the form:

$$---F \mid --G_1 \mid \ldots \mid --G_n \mid H.$$

In contrast with Theorem 20, we have the following result.

Theorem 21 *A theory T is an extension of a rule base Δ in $\mathcal{R}^{\mathcal{A}}$ iff T is the kernel of a stable expansion of $\Theta^{-1}(\Delta)$, where Θ^{-1} translates each rule in $\mathcal{R}^{\mathcal{A}}$ of the form $---F \mid --G_1 \mid \ldots \mid --G_n \mid H$ into $\neg L(F) \vee L(G_1) \vee \ldots \vee L(G_n) \vee H$.*

According to Theorem 20 and Theorem 21, it can be concluded that auto-epistemic logic is equivalent to $\mathcal{R}^{\mathcal{A}}$, which is a subclass of general default logic. Moreover, the self introspection operator L indeed plays the same role as the double negation as failure operator $--$.

We may be interested in whether a translation is *modular* or not. Rough speaking, a modular translation means that the translation can be applied one by one. Modularity is important in translation among non-monotonic formalisms both from a conceptual and computational point of view (Gottlob 1995). We say that a translation tr from auto-epistemic logic into general default logic is *modular* iff for any two sets Γ_1 and Γ_2 of AEL formulas, $tr(\Gamma_1 \cup \Gamma_2) = tr(\Gamma_1) \cup tr(\Gamma_2)$. It is obvious that our translation from auto-epistemic logic to general default logic is a modular translation.

As Gottlob pointed out in (Gottlob 1995), there is no modular translation from Reiter's default logic to auto-epistemic logic. Then he concluded that auto-epistemic logic is strictly more expressive than Reiter's default logic. Hence, our result shows that, in the sense of Gottlob's idea of expressiveness, general default logic is a non-trivial extension of Reiter's default logic.

Janhunen (1999) proposed another perspective on comparing expressiveness among nonmonotonic formalisms. He also adopted Gottlob's idea of translation as a basic tool. However, in contrast, auxiliary atoms are allowed to introduce. Interestingly, he concluded that, on the contrary, Reiter's default logic is more expressive than auto-epistemic logic. We believe that it is important to prohibit new atoms since they also bring new information. Another reason comes from Ferraris's recent work (Ferraris 2005b), which proves that, in terms of expressiveness among classes of answer set programs, modular translation without auxiliary atoms is identical to strong equivalence.

Ferraris's idea of treating strong equivalence as a criterion for expressiveness can be generalized into default logic as well. Reiter's default logic (Gelfond et al.'s disjunctive default logic) can also be considered as a subclass $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$) of general default logic by restricting the rules into Reiter's original default rules (disjunctive default rules). Moreover, $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$) is distinct with $\mathcal{R}^{\mathcal{A}}$. That is, there exists a rule in $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$), for instance $p \Rightarrow q$, which is not strongly equivalent to any rules in $\mathcal{R}^{\mathcal{A}}$. On the other hand, there exists a rule in $\mathcal{R}^{\mathcal{A}}$, for instance $---p \mid q$, which is not strongly equivalent to any rules in $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$). Hence, Reiter's default logic (Gelfond et al.'s disjunctive default logic) and Moore's auto-epistemic logic are indeed two disjoint nonmonotonic formalisms. However, both of them are subclasses of general default logic. This also confirms that general default logic is a non-trivial extension of Reiter's default logic (disjunctive default logic).

The three subclasses of general logic can be compared by considering the normal forms of them respectively under strong equivalence.

Theorem 22 *1. Each rule in $\mathcal{R}^{\mathcal{N}}$ (i.e. Reiter's default logic) is strongly equivalent to a set of rules of the following form*

$$C_1 \,\&\, \ldots \,\&\, C_n \,\&\, -C_{n+1} \,\&\, \ldots \,\&\, -C_m \Rightarrow C_{m+1},$$

where $C_i, (1 \leq i \leq m+1)$ are propositional clauses.

2. *Each rule in $\mathcal{R}^{\mathcal{D}}$ ((i.e. disjunctive default logic)) is strongly equivalent to a set of rules of the following form*

$$C_1 \& \ldots \& C_n \& -C_{n+1} \& \ldots \& -C_m \Rightarrow C_{m+1} \mid \ldots \mid C_k,$$

where $C_i, (1 \leq i \leq k)$ are propositional clauses.

3. *Each rule in $\mathcal{R}^{\mathcal{A}}$ (corresponding to auto-epistemic logic) is strongly equivalent to a set of rules of the following form*

$$-C_1 \,\&\, \ldots \,\&\, -C_n \Rightarrow -C_{n+1} \mid \ldots \mid -C_m \mid C_{m+1},$$

where $C_i, (1 \leq i \leq m+1)$ are propositional clauses.

Conclusion

The contribution of this paper are three folds. Firstly, we showed that the logic of GK (Lin & Shoham 1992) is indeed a general framework for nonmonotonic reasoning by embedding general default logic (Zhou, Lin, & Zhang 2007) into it. We also clarified the intuitions behind this translation and other similar translations among nonmonotonic formalisms (Lin & Shoham 1992; Lin 2002; Lin & Zhou 2007; Pearce, Tompits, & Woltran 2001; Ferraris, Lee, & Lifschitz 2007) by reformulating another equivalent translation.

Secondly, we demonstrated that the logic of GK is not only a general framework of nonmonotonic formalisms but also a powerful tool to study them. This is not pointed out by previous work (Lin & Shoham 1992; Lin 2002; Lin & Zhou 2007) before. As an example, we showed that both weak equivalence and strong equivalence can be captured in the logic of GK and the complexities for both of them are coNP complete. We also showed that each rule base in general default logic can be strongly equivalently transformed into a set of rules of form (1), which cannot be further transformed into disjunctive default rules. This result can also be viewed as a generalization of a similar result for general logic programming (Ferraris 2005a) since answer set programming is a special case of general default logic. Meanwhile, it also indicates that, to some extent, general default logic is a strict extension of disjunctive default logic (Gelfond *et al.* 1991) (thus Reiter's default logic (Reiter 1980)).

Finally, we proved that Moore's auto-epistemic logic (Moore 1987) is equivalent to a proper subset $\mathcal{R}^{\mathcal{A}}$ of general default logic via the logic of GK. Hence, the self-introspection operator L in auto-epistemic logic indeed plays the same role as the double negation-as-failure operator $--$ in general default logic. Since Reiter's default logic (disjunctive default logic) can also be considered as a proper subset $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$) of general default logic, and $\mathcal{R}^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{N}}$ ($\mathcal{R}^{\mathcal{D}}$) are disjoint with each other, it can be concluded that auto-epistemic logic and Reiter's default logic are actually two disjoint nonmonotonic formalisms. However, both of them are proper subclasses of general default logic. As a consequence, general default logic is a strict extension of Reiter's default logic (disjunctive default logic). Indeed, this fact is also confirmed by Gottlob's result (Gottlob 1995), stating that auto-epistemic logic cannot be translated into Reiter's default logic by a modular translation without auxiliary atoms.

To sum up, in comparison with related work of translations among nonmonotonic formalisms, we are not only interested in the translation itself, but also interested in the benefits of it. As we have shown in this paper, the translation from general default logic into the logic of GK is indeed useful. First of all, it confirms the generality of the logic of GK. And then, it solves a number of important problems in relation to general default logic via the logic of GK. Last but not least, it provides better understandings of both two nonmonotonic formalisms and others, for instance, auto-epistemic logic.

References

Cabalar, P., and Ferraris, P. 2007. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming* 7(6):745–759.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2007. A new perspective on stable models. In *IJCAI*, 372–379.

Ferraris, P. 2005a. Answer sets for propositional theories. In *Proceedings of the LPNMR'2005*, 119–131.

Ferraris, P. 2005b. On modular translations and strong equivalence. In *Proceedings of the LPNMR'2005*, 79–91.

Gelfond, M.; Lifschitz, V.; Przymusinska, H.; and Truszczyński, M. 1991. Disjunctive defaults. In *Proceedings of the KR'91*, 230–237.

Gottlob, G. 1995. Translating default logic into standard autoepistemic logic. *Journal of ACM* 42(4):711–740.

Halpern, J. Y., and Moses, Y. 1992. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence* 54(3):319–379.

Janhunen, T. 1999. On the intertranslatability of nonmonotonic logics. *AMAI* 27(1-4):79–128.

Konolige, K. 1988. On the relation between default and autoepistemic logic. *Artificial Intelligence* 35(3):343–382.

Ladner, R. E. 1977. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing* 6(3):467–480.

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

Lifschitz, V. 1991. Nonmonotonic databases and epistemic queries. In *IJCAI*, 381–386.

Lin, F., and Shoham, Y. 1992. A logic of knowledge and justified assumptions. *Artificial Intelligence* 57:271–289.

Lin, F., and Zhou, Y. 2007. From answer set logic programming to circumscription via the logic of gk. In *IJCAI*, 441–446.

Lin, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *KR*, 170–176.

Moore, R. 1987. Possible-world semantics for autoepistemic logic. 137–142.

Motik, B., and Rosati, R. 2007. A faithful integration of description logics with logic programming. In *IJCAI*, 477–482.

Pearce, D.; Tompits, H.; and Woltran, S. 2001. Encodings for equilibrium logic and logic programs with nested expressions. In *EPIA*, 306–320.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Truszczynski, M. 2007. The modal logic s4f, the default logic, and the logic here-and-there. In *AAAI*, 508–.

Turner, H. 2001. Strong equivalence for logic programs and default theories (made easy). In *LPNMR*, 81–92.

Zhou, Y.; Lin, F.; and Zhang, Y. 2007. General default logic. In *Proceedings of the LPNMR'2007*, 241–253.

Appendix

In the appendix, we outline the proof sketches of some properties presented in this paper.

Proof:[proof of Lemma 1] We prove this assertion by induction on the structure of R.

- If R is a fact, then this assertion holds obviously.
- If R is $R_1|R_2$, then R^{T_2} is weakly equivalent to $R_1^{T_2}|R_2^{T_2}$. $T_1 \models R^{T_2}$ iff $T_1 \models R_1^{T_2} \,|\, R_2^{T_2}$ iff $T_1 \models R_1^{T_2}$ or $T_1 \models R_2^{T_2}$ iff M is a model of $(R_1)_{GK}$ or M is a model of $(R_2)_{GK}$ iff M is a model of R_{GK}.
- If R is R_1 & R_2, then R^{T_2} is weakly equivalent to $R_1^{T_2}$ & $R_2^{T_2}$. $T_1 \models R^{T_2}$ iff $T_1 \models R_1^{T_2}$ & $R_2^{T_2}$ iff $T_1 \models R_1^{T_2}$ and $T_1 \models R_2^{T_2}$ iff M is a model of $(R_1)_{GK}$ and M is a model of $(R_2)_{GK}$ iff M is a model of R_{GK}.
- If R is $R_1 \Rightarrow R_2$, then R^{T_2} is weakly equivalent to $R_1^{T_2} \Rightarrow R_2^{T_2}$. $T_1 \models R^{T_2}$ iff $T_1 \models R_1^{T_2} \Rightarrow R_2^{T_2}$ iff $T_1 \not\models R_1^{T_2}$ or $T_1 \models R_2^{T_2}$ iff M is not a model of $(R_1)_{GK}$ or M is a model of $(R_2)_{GK}$ iff M is a model of R_{GK}.

This completes the induction proof. ■

Proof:[proof of Theorem 2] $\Rightarrow$: Suppose that T is an extension of Δ. Construct a Kripke interpretation M such that $K(M) = A(M) = T$. By Lemma 1, M is a model of Δ_{GK}. Moreover, M is a GK model of Δ_{GK}. Otherwise, suppose M_1 is a model of Δ_{GK} and $K(M_1) \subset K(M)$, $A(M_1) = A(M) = T$. By Lemma 1, $K(M_1) \models \Delta^T$. This shows that T is not an extension of Δ, a contradiction.

$\Leftarrow$: Suppose that there is a GK model M of Δ_{GK} such that $K(M) = A(M) = T$. By Lemma 1, $T \models \Delta^T$. Moreover, there is no proper subset T_1 of T such that T_1 is also a model of Δ^T. Otherwise, we can construct a Kripke interpretation M_1 such that $K(M_1) = T_1$ and $A(M_1) = T$. By Lemma 1, M_1 is also a model of Δ_{GK}. This shows that M is not a GK model of Δ_{GK}, a contradiction. ■

Proof:[proof of Proposition 5] Let M be a model of F. Let Γ be the set of facts that F is constructed from. Let $\Gamma_1 = \{G \,|\, G \in \Gamma, M \models K(G)\}$. Then, for every $P \in \Gamma \backslash \Gamma_1$, there is a truth assignment satisfies $Q \wedge \bigwedge_{G \in \Gamma_1} G$. Symmetrically, the same thing can be done for modal operator A.

Construct a Kripke interpretation M_1 such that the K accessible worlds of the actual world are exactly the truth assignments mentioned above, so are the A accessible worlds. Then, for all formula $G \in \Gamma$, $M \models K(G)$ iff $M_1 \models K(G)$; $M \models A(G)$ iff $M_1 \models A(G)$. Hence, M_1 is also a model of F. Moreover, M_1 has at most $2|F| + 1$ possible worlds. ■

Proof:[proof of Theorem 9] $2 \Rightarrow 3$ and $4 \Rightarrow 1$ are obvious.

$3 \Rightarrow 4$: Firstly, if $\bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F) \models (R_1)_{GK} \leftrightarrow (R_2)_{GK}$, then R_1 and R_2 are weakly equivalent. Thus, by Theorem 7, $(R_1)_A$ and $(R_2)_A$ are equivalent in the logic of GK. By induction on the structure, $(R_3)_{GK}$ and $(R_4)_{GK}$ are equivalent. Thus they have the same set of GK models. By Theorem 2, R_3 and R_4 have the same set of extensions.

$1 \Rightarrow 2$: Suppose otherwise M is a model of $(R_1)_{GK}$ but not a model of $(R_2)_{GK}$. Let T_1 be $K(M)$ and T_2 be $A(M)$. There are two cases. (a)$T_2 \models R_2^{T_2}$. Let R_3 be the rule conjunction of $\{F \mid F \in Fact(R_1 \,\&\, R_2), T_1 \models F\}$ and $\{F \Rightarrow G | F, G \in Fact(R_1 \& R_2); T_2 \models F, G; T_1 \not\models F, G.\}$. We have that T_2 is an extension of R_2 & R_3 but not an extension of R_1 & R_3. (b)$T_2 \not\models R_2^{T_2}$. Let R_3 be the rule conjunction of $\{F \mid F \in Fact(R_1 \,\&\, R_2), T_2 \models F\}$. We have that T_2 is an extension of R_1 & R_3 but not an extension of R_2 & R_3. In both cases, R_1 is not strongly equivalent to R_2, a contradiction. ■

Proof:[proof of Proposition 13] $(F \wedge G)_{GK}$ is $K(F \wedge G)$; while $(F \,\&\, G)_{GK}$ is $F_{GK} \wedge G_{GK}$, which is $K(F) \wedge K(G)$. Thus, $(F \wedge G)_{GK}$ is equivalent to $(F \,\&\, G)_{GK}$ in the logic of GK. By Theorem 9, $F \wedge G \equiv F \,\&\, G$. ■

Proof:[proof of Proposition 15] All these assertions can be proved the same way as the proof of Proposition 13 by Theorem 9. As an example, here we only outline the proof of 13.

Notice that if $\models \bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F)$, then by induction on the structure, for every rule R, $R_{GK} \models R_A$. Consider 13, $((F \Rightarrow G) \mid H)_{GK}$ is

$$((F_{GK} \rightarrow G_{GK}) \wedge (F_A \rightarrow G_A)) \vee H_{GK},$$

which is equivalent to

$$(\neg F_{GK} \vee G_{GK} \vee H_{GK}) \wedge (\neg F_A \vee G_A \vee H_{GK})$$

under $\bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F)$. On the other hand,

$$((F \Rightarrow G \mid H) \,\&\, (H \mid -F \mid --G))_{GK}$$

is equivalent to

$$(F_{GK} \rightarrow G_{GK} \vee H_{GK}) \wedge (F_A \rightarrow G_A \vee H_A) \wedge (H_{GK} \vee \neg F_A \vee G_A),$$

which is also equivalent to

$$(\neg F_{GK} \vee G_{GK} \vee H_{GK}) \wedge (\neg F_A \vee G_A \vee H_{GK}),$$

under $\bigwedge_{F \in \mathcal{L}} K(F) \rightarrow A(F)$. Thus by Theorem 9, 13 holds. ■

Proof:[proof of Theorem 18] We first prove a lemma by induction on the structure that each rule base is strongly equivalent to a set of rules of the form $C \Rightarrow D$, where D has the form of (3), and C has the form of

$$C_1 \& \ldots \& C_n \& -C_{n+1} \& \ldots \& -C_m \& --C_{m+1} \& \ldots \& --C_k,$$

where $C_i, (1 \leq i \leq k)$ are propositional clauses. A tedious step of proving this lemma is to reduce the rule $(C_1 \Rightarrow D_1) \,\&\, (C_2 \Rightarrow D_2) \Rightarrow (C_3 \Rightarrow D_3)$ mainly by 14 and 15 in Proposition 15.

Then, by 9, 16 and 17 in Proposition 15, this form can be strongly equivalently transformed to form (1). ■

Proof:[Proof of Theorem 22] Point 1 and point 2 follow easily from Theorem 3 and 4 in (Zhou, Lin, & Zhang 2007) and Corollary 14. Point 3 follows from Corollary 14 and the fact that $-R_1$ & $R_2 \Rightarrow R_3$ is strongly equivalent to $R_2 \Rightarrow R_3 | --R_1$, where R_1, R_2 and R_3 are three rules. ■

Special Session on Applications

A number of systems which implement nonmonotonic reasoning (NMR) or make extensive use of NMR techniques have been developed in the past decade. The efficiency of such systems has been boosted by the increasing performance of computer hardware as well as advances in algorithm design. The current performance level is already sufficient to enable industrial applications of nonmonotonic reasoning. Indeed, a wide range of applications has emerged along with the development of NMR systems. This special session aims to attract researchers who have interest and/or practical experience in significant applications of NMR. The aim is to share views about the current state of the art and to look for new emerging areas of application.

Session Chairs

Tomi Janhunen, Helsinki University of Technology, Finland
Eugenia Ternovska, Simon Fraser University, Canada

Program Committee

Leopoldo Bertossi, Carleton University, Canada
Jim Delgrande, Simon Fraser University, Canada
Wolfgang Faber, University of Calabria, Italy
Michael Fink, Vienna University of Technology, Austria
Paolo Liberatore, University of Rome "La Sapienza", Italy
Leora Morgenstern, Stanford University, USA
Pascal Nicolas, University of Angers, France
Simona Perri, University of Calabria, Italy
Jussi Rintanen, NICTA, Australia
Riccardo Rosati, University of Rome "La Sapienza", Italy
Torsten Schaub, University of Potsdam, Germany
Mirek Truszczynski, University of Kentucky, USA
Jia You, University of Alberta, Canada
Mary-Anne Williams, University of Technology Sydney, Australia

Anton: Answer Set Programming in the Service of Music

Georg Boenn
Cardiff School of Creative & Cultural Industries
University of Glamorgan
Cardiff, CF24 2FN, UK
gboenn@glam.ac.uk

Martin Brain and **Marina De Vos** and **John ffitch**
Department of Computer Science
University of Bath
Bath, BA2 7AY, UK
{mjb,mdv,jpff}@cs.bath.ac.uk

Abstract

With the increasing efficiency of answer set solvers and a better understanding of program design, answer set programming has reached a stage where it can be more successfully applied in a wider range of applications and where it attracts attention from researchers in other disciplines. One of these domains is music synthesis. In this paper we approach the automation and analysis of composition of music as a knowledge representation and advanced reasoning task. Doing so, it is possible to capture the underlying rules of melody and harmony by a very small, simple and elegant set of logic rules that can be interpreted under the answer set semantics. Our system, ANTON is the first algorithmic composer to combine both harmonic and melodic composition. In addition to describing the composition system thus created we consider the advantages of constructing an algorithmic composer this way, and also the limitations of current solvers.

Introduction

Originally computers were seen as machines to assist in numerical calculations, and it was soon realised that they could do other things, starting with commerce, but extending to symbolic operations and eventually to near-universal use in all technical areas. More recently the application of computers to artistic activity has become a subject of interest.

In this paper we report on the use of declarative logic programming as a significant component of an artistic endeavour, the composition of music. We show that it is possible to use Answer Set Programming (ASP) to create *ab initio* short musical pieces that are both melodic and harmonic. After a description of the computational basis we describe the musical context of this work, and why it is neither a trivial task, nor a tractable one. Our system, ANTON, named in honour of our favourite composer of the second Viennese School, is presented as both a design and as a practical working system. We report on our experience in using ASP for this system, and indicate a number of potentially exciting directions in which this system could develop, both musically and computationally.

Answer Set Programming

Due to space constraints, only a brief overview of the answer set semantics and Answer Set Programming (ASP) is given here. The interested reader is referred to (Baral 2003) for a more in-depth coverage of the definitions and ideas presented in this section.

The *answer set semantics* is a model based semantics for normal logic programs.

Following the notation of (Baral 2003), we refer to the language over which the answer set semantics is defined as $AnsProlog$.

The basic components of the language are atoms, elements that can be assigned a truth value. An atom can be negated using *negation as failure* in order to create the *literal* $\mathrm{not}\ a$. If a is true then $\mathrm{not}\ a$ is false and vice versa. Atoms and literals are used to create rules of the form:

$$a \leftarrow B, \mathrm{not}\ C.$$

where a is an atom, B and C are sets of atoms. Intuitively, this means "if all element of B are known/true and no element of C is known/true, then a is known/true". The set of conditions of a rule (on the right hand side of the arrow) are known as the *body*, written as $B(r)$, and the atom that is the consequence of the rule is referenced as the *head* of the rule, written $H(r)$. The body is split further in two sets of atoms, $B^+(r)$ and $B^-(r)$ depending on whether the atom appears positively or negatively. Rules with empty bodies are called *facts*; their head should always be true.
A *program* in $AnsProlog$ is a finite set of rules.

If a program Π contains no negated atoms ($\forall r \in \Pi \,.\, B^-(r) = \emptyset$) its semantics is unambiguous and can easily be computed as the fixed point of the T_p(the immediate consequence) operator. Starting from the empty set, we check in each iteration which rule bodies are true. The heads of those rules are added to the set for the next iteration. This is a monotonic process, so we obtain a unique fixpoint, denoted $T_p^\infty(\emptyset)$. This fixpoint is called the *answer set*.

For example, given the following program:

$$\begin{aligned} a &\leftarrow b, c. \\ b &\leftarrow c. \\ c &\leftarrow . \\ d &\leftarrow e. \\ e &\leftarrow d. \end{aligned}$$

the unique answer set is $\{a, b, c\}$, as $T_p(\emptyset) = \{c\}$, $T_p(\{c\}) = \{b, c\}$, $T_p(\{b, c\}) = \{a, b, c\}$ and $T_p(\{a, b, c\}) = \{a, b, c\}$. Note that d and e are not included in the model as their is no way of concluding e without knowing d and vice versa. This is different to the classical interpretation of this program (via Clark's completion) which would have two models, one of which would contain d and e.

The natural mechanism for computing negation in logic programs in *negation as failure*, which tends to be characterised as epistemic negation ("we do not know this is true"), rather than classical negation ("we know that this is not true"). This correspondence is motivated by the intuition that we should only claim to know things that can be proven; thus anything that can not be proven is not known. To extend the semantics to support this type of negation, the *Gelfond-Lifschitz reduct* is used. This takes a set of proposed atoms and gives a reduced, positive program by removing any rule which depends on the negation of any atom in the set and dropping all other negative dependencies.

Definition 1 *Given an $AnsProlog$ program Π and a set of atoms A, the Gelfond-Lifschitz transform of Π with respect to A is the following set of rules:*

$$\Pi^A = \{H(r) \leftarrow B^+(r) | r \in \Pi, B^-(r) \cap A = \emptyset\}$$

This allows us to extend the concept of answer sets to programs with negation. Intuitively, these are sets of possible beliefs about the world which are consistent with all of the rules and have acyclic support for every atom that is known, and thus in the set.

Definition 2 *Given an $AnsProlog$ program Π, A is an answer set of Π $\iff$ A is the unique answer set of Π^A.*

For example, the following program has two answer sets:

$$\begin{aligned} a &\leftarrow \text{not } b. \\ b &\leftarrow \text{not } a. \\ c &\leftarrow \text{not } d. \\ d &\leftarrow b. \\ d &\leftarrow e, \text{not } a, \text{not } c. \\ e &\leftarrow d, \text{not } a. \end{aligned}$$

$\{a, c\}$ and $\{b, d, e\}$. Computing the reduct with respect to $\{a, c\}$ gives:

$$\begin{aligned} a &\leftarrow . \\ c &\leftarrow . \\ d &\leftarrow b. \end{aligned}$$

which results in $T_p^\infty(\emptyset) = \{a, c\}$.

A given program will have zero or more answer sets. With $AnsProlog$ we can represent and reason about NP-complete problems in such a way that the answer sets of the program correspond to the solutions of the problem.

When used as a knowledge representation and programming language, $AnsProlog$ is enhanced to contain constraints (e.g. $\leftarrow b, \text{not } c$) and choice rules (*e.g.* $\{a, b, c\} \leftarrow b, \text{not } c$). The former are rules with an empty head, stating that an answer set cannot meet the conditions given in the body. The latter is a short hand notation for a conditional choice; if the conditions in the body are met then a number of atoms in the head may (a non-deterministic choice) be contained in answer set. These additions are syntactic sugar and can be removed with linear, modular transformations (see (Baral 2003)). Variables and predicated rules are also used and are handled, at the theoretical level and in most implementations, by instantiation (referred to as *grounding*).

Answer set programming (ASP) is a programming paradigm in which a problem is *represented* as an $AnsProlog$ program in such a way that the answer sets can be *interpreted* to give the solutions. A reasoning engine is then used to produce the answer sets of the program. Typically these are composed of two components, a *grounder* which removes the variables from the program by instantiation and an *answer set solver* which compute answer sets of the propositional program. GRINGO(Gebser, Schaub, and Thiele 2007) and LPARSE(Syrjänen 2000) are the grounders most commonly used and CLASP(Gebser et al. 2007), SMODELS(Syrjänen and Niemelä 2001), CMODELS(Lierler and Maratea 2004) and DLV(Eiter et al. 1998) represent the state of the art of solver development.

ASP has been used to tackle a variety of problems, including: planning and diagnosis (Eiter et al. 2002; Lifschitz 2002; Nogueira et al. 2001), modelling and rescheduling of the propulsion system of the NASA Space Shuttle (Nogueira et al. 2001), multi-agent systems (Baral and Gelfond 2000; Buccafurri and Caminiti 2005; Cliffe, De Vos, and Padget 2006), Semantic Web and web-related technologies (Polleres 2005; Ruffolo et al. 2005), superoptimisation (Brain et al. 2006), reasoning about biological networks (Grell, Schaub, and Selbig 2006), voting theory (Konczak 2006) and investigating the evolution of language (Erdem et al. 2003).

The Musical Background

Music is a world-wide phenomenon across all cultures. The details of what constitutes music may vary from nation to nation, but it is clear that music is an important component of being human.

In this paper we are concentrating on western traditional musics, but as we will consider later in the section on future music research, much of the technology can be translated to other traditions.

The particular area of interest here is composition; that is creating new musical pieces.

Creating melodies, that is sequences of pitched sounds, is not as easy as it looks (sounds). We have cultural preferences for certain sequences of notes and preferences dictated by the biology of how we hear. This may be viewed as an artistic (and hence not scientific) issue, but most of us would be quick to challenge the musicality of a composition created purely by random whim. Students are taught rules of thumb to ensure that their works do not run counter to cultural norms and also fit the algorithmically definable rules of pleasing harmony when sounds are played together.

"Western tonal" simply refers to what most people in the West think of as "classical music", the congenial Bach through Brahms music which feels comfortable to the modern western ear because of its adherence to familiar rules. Students of composition in conservatoires are taught to write this sort of music as basic training. They learn to write melodies and to harmonise given melodies in a number of sub-versions. If we concentrate on early music then the scheme often called "Palestrina Rules" is an obvious example for the basis of this work. Similarly, harmonising Bach chorales is a common student exercise, and has been the subject of many computational investigations using a variety of methods.

In this paper, we take the somewhat arid technical rules and embed them within a modern computational system, which enables us to contemplate many original ways of exploiting the fact that they are simultaneously available; the rules themselves can be explored, extended and refined, or student exercises can be evaluated to ensure that they are indeed "valid". We will be able to complete partial systems, such as producing a melody consonant with a given harmony structure, as well as, more adventurously, to create new melodies.

For this paper we have opted to work with a sub-type of the Palestrina Rules called Renaissance Counterpoint. This style was used by composers like Josquin, Dufay or Palestrina and is very distinct from the Baroque Counterpoint used by composers like Bach, Haendel.

We have used the teaching at one conservatoire in Köln to provide the basic rules, which were then refined in line with the general style taught. The point about generating melodies is that the "tune" must be capable of being accompanied by one or more other lines of notes, to create a harmonious whole. The requirement for the tune to be capable of harmonisation is a constraint that turns a simple sequence (a *monody*) to a *melody*.

Our experience with this work is to realise how many acceptable melodies can be created with only a few rules, and as we add rules, how much better the musical results are. This concept is developed further in the section on ANTON.

In this particular style of music complete pieces are not usually created in one go. Composers create a number of sections of melody, harmonising them as needed, and possibly in different ways, and then structuring the piece around these basic sections. Composing between 4 bars and 16 bars is not only a computationally convenient task, it is actually what the human would do, creating components from which the whole is constructed. So although the system described here may be limited in its melodic scope, it has the potential to become a useful tool across a range of sub-styles.

Automatic Composition

A common problem in musical composition can be summarised in the question "where is the next note coming from?". For many composers over the years the answer has been to use some process to generate notes. It is clear that in many pieces from the Baroque period that simple note sequences are being elaborated in a fashion we would now call algorithmic. For this reason we can say that algorithmic composition is a subject that has been around for a very long time. It is usual to credit Mozart's Musikalisches Würfelspiel (Musical Dice Game) (Chuang 1995) as the oldest classical algorithmic composition, although there is some doubt if the game form is really his. In essence the creator provides a selection of short sections, which are then assembled according to a few rules and the roll of a set of dice to form a Minuet[1]. Two dice are used to choose the 16 minuet measures from a set of 176, and another die selects the 16 trio measures[2], this time from 96 possible. This gives a total number of 1.3×10^{29} possible pieces. This system however, while using some rules, relies on the coherence of the individual measures. It remains a fun activity, and recently web pages have appeared that allow users to create their own original(ish) "Mozart" compositions.

In the music of the second Viennese school ("12-tone", serial music) there is a process in action, rotating, inverting and use of retrograde, but usually performed by hand.

More recent algorithmic composition systems have concentrated on the generation of monody[3], either from a mathematical sequence, chaotic processes, or Markov chains, trained by consideration of acceptable other works. Frequently the systems rely on a human to select which monodies should be admitted, based on judgement rather than rules. Great works have been created this way, in the hands of great talents. Major descriptions of mathematical note generators can be found for example in *Formalized Music* (Xenakis 1992). Probably the best known of the Markov chain approach is Cope's significant corpus of Mozart pastiche (Cope 2006).

In another variation on this approach, the accompanist, either knowing the chord structure and style in advance, or using machine-listening techniques, infers a style of accompaniment. The former of these approaches can be found in commercial products, and the latter has been used by some jazz performers to great effect, for example by George E. Lewis.

A more recent trend is to cast the problem as one of constraint satisfaction. For example PWConstraints is an extension for IRCAM's Patchwork, a Common-Lisp-based graphical programming system for composition. It uses a custom constraint solver employing backtracking over finite integer domains. OMSituation and OMClouds are similar and are more recently developed for Patchwork's successor OpenMusic. A detailed evaluation of them can be found in (Anders 2007), where the author gives an example of a 1st-species counterpoint (two voices, note against note) after (Fux 1965 orig 1725) developed with Strasheela, a constraint system for music built on the multi-paradigm language Oz. Our musical rules however implement the melody and counterpoint rules described by (Thakar 1990), which we find give better musical results.

[1] A dance form in triple time, *i.e.* with 3 beats in each measure

[2] A Trio is a short contrasting section played before the minuet is repeated

[3] A monody is a single solo line, in opposition to homophony and polyphony

One can distinguish between *improvisation* systems and *composition* systems. In the former the note selection progresses through time, without detailed knowledge of what is to come. In practice this is informed either by knowing the chord progression or similar musical structures (Brothwell and ffitch 2008), or using some machine listening. In this paper we are concerned with *composition*, so the process takes place out of time, and we can make decisions in any order.

It should also be noted that these algorithmic systems compose pieces of music of this style in either a melodic or a harmonic fashion, and are frequently associated with computer-based synthesis. The system we will propose later is unique as it deals with both simultaneously.

Melodic Composition

In melodic generation a common approach is the use of some kind of probabilistic finite state automaton or an equivalent scheme, which is either designed by hand (some based on chaotic oscillators or some other stream of numbers) or built via some kind of learning process. Various Markov models are commonly used, but there have been applications of n-grams, genetic algorithms and neural nets. What these methods have in common is that there is no guarantee that melodic fragments generated have acceptable harmonic derivations. Our approach, described below is fundamentally different in this respect, as our rules cover both aspects simultaneously.

In contrast to earlier methods, which rely on learning, and which are capable of giving only local temporal structure, a common criticism of algorithmic melody (Leach 1999), we do not rely on learning and hence we can aspire to a more global, whole melody, approach. In addition we are no longer subject to the limitations of the kind of process which, because it only works in time in one direction, is hard to use in a partially automated fashion; for example operations like "fill in the 4 notes between these sections" is not a problem for us.

We are also trying to move beyond experiments with random note generation, which we have all tried and abandoned because the results are too lacking in structure. Predictably, the alternative of removing the non-determinism at the design stage (or replacing with a probabilistic choice) runs the risk of 'sounding predictable'! There have been examples of good or acceptable melodies created like this, but the restriction inherent in the process means it probably works best in the hands of geniuses.

Harmonic Composition

A common usage of algorithmic composition is to add harmonic lines to a melody; that is notes played at the same time as the melody that are in general consonant and pleasing. This is exemplified in the harmonisation of 4-part chorales, and has been the subject of a number of essays in rule-based or Markov-chain systems. Perhaps a pinnacle of this work is (Ebcioğlu 1986) who used early expert system technology to harmonise in the style of Bach, and was very successful. Subsequently there have been many other systems, with a range of technologies. There is a review included in (Rohrmeier 2006).

```
% At every time step, every part either steps
% to the next note in the key or leaps to a
% further note in the key
1 { stepUp(P,T), stepDown(P,T), leapUp(P,T),
   leapDown(P,T) } 1 :- part(P), time(T).

% A leap can only be over a consonant interval
% (3,4,5,7 or 12 semitones)
1 { leapBy(P,T,I) : consonantInterval(I) } 1
    :- leapUp(P,T).

% When a part leaps up by I, the note at time T+1
% is I steps higher than the current note
chosenNote(P,T+1,N+I) :- chosenNote(P,T,N),
                         leapBy(P,T,I).

% Every note must be in the chosen mode
% (major, minor, etc.)
:- chosenNote(P,T,N), mode(M), not inMode(N,M).

% The interval between parts must not be dissonant
:- chosenNote(P1,T,N1), chosenNote(P2,T,N2),
   interval(N1,N2,C), not consonantInterval(C).
```

Figure 1: A simplified ANTON fragment

Clearly harmonisation is a good match to constraint programming based systems, there being accepted rules[4]. It also has a history from musical education.

But these systems all start with a melody for which at least one valid harmonisation exists, and the program attempts to find one, which is clearly soluble. This differs significantly from our system, as we generate the melody and harmonisation together, the requirement for harmonisation affecting the melody.

The ANTON Composition System

What we are seeking to do, which is a new application in both music and computing, is to apply ASP techniques to compositional rules to produce an algorithmic composition system which can be applied more widely and freely than has previously been possible. $AnsProlog$ is used to create a description of the rules that govern the melodic and harmonic properties of a correct piece of music. The $AnsProlog$ program works as a model for music composition that can be used to assist the composer by suggesting, completing and verifying short pieces.

The rules of composition are modelled so that the $AnsProlog$ program defines the requirements for a piece to be valid, and thus every answer set corresponds to a valid piece. In generating an new piece, the composition system simply has to generate an (arbitrary) answer set. Rather than the traditional problem/solution mapping of answer set programming, this is using an $AnsProlog$ program to create a 'random' (arbitrary) example of a complex, structured object.

Figure 1 presents a simplified fragment of the $AnsProlog$ program used in ANTON. The model is defined over a number of time steps, given by the variable `T`. The key proposi-

[4]For example see: `http://www.wikihow.com/Harmonise-a-Chorale-in-the-Style-of-Bach`

tion is `chosenNote(P,T,N)` which represents the concept "At time `T`, part `P` plays note `N`". To encode the options for melodic progress ("the tune either steps up or down one note in the key, or it leaps more than one note"), choice rules are used. To encode the melodic limits on the pattern of notes and the harmonic limits on which combinations of notes may be played at once, constraints are included.

To allow for verification and diagnosis, each rule is given an error message:

```
% No tri-tones:  No note can be within two notes
% of a tritone (a note +/- 6 semitones)
#const err_tt="Tri-tone".
reason(err_tt).
error(P,T,err_tt) :-  chosenNote(P,T,N1),
                      chosenNote(P,T+2,N1+6).
error(P,T,err_tt) :-  chosenNote(P,T,N1),
                      chosenNote(P,T+2,N1-6).
```

Depending on how you want to use the system, composition or diagnosis, you will either be interested in those pieces that do not result in errors at all, or in an answer set that mentions the error messages. For the former we simply specify the constraint `:- error(P,T,R).`, effectively making any error rule into a constraint. For the latter we include the rules: `errorFound :- error(P,T,R).` and `:- not errorFound.`, requiring that an error is found (i.e. returning no answers if the diagnosed piece is error free).

By adding constraints on which notes can be included, it is possible to specify part or all of a melody, harmony or complete piece. This allows ANTON to be used for a number of other tasks beyond automatic composition. By fixing the melody it is possible to use it as an automatic harmonisation tool. By fixing part of a piece, it can be used as computer aided composition tool.

The complete system consists of three major phases; building the program, running the solver and interpreting the results. As a simple example suppose we wish to create a 4 bar piece in E major one would use the Perl wrapper and write

```
$ programBuilder.pl --task=compose \
                    --mode=major \
                    --time=16 > program
```

which builds the ASP program, giving the length and mode. Then

```
$ lparse -W all < program | \
  shuffle.pl 6298 | \
  smodels 1 > tunes
```

runs the grounder and solver and generates a representation of the piece. Using another Perl script we provide a number of output formats, one of which is a CSOUND (Boulanger 2000) program with a suitable selection of sounds.

```
$ parse.pl --fundamental=e --output=csound \
  < tunes > tunes.csd
```

generates the CSOUND input from the generic format, and then

```
$ csound tunes.csd -o dac
```

plays the melody. We provide in addition to CSOUND, output in text, $AnsProlog$ facts or the LILYPOND score language(Nienhuys and Nieuwenhuizen 2003). Naturally we provide scripts for all main ways of using the system.

```
keyMode(lydian).
chosenNote(1,1,25).
chosenNote(1,2,24).
chosenNote(1,8,19).
chosenNote(1,9,20).
chosenNote(1,10,24).
chosenNote(1,14,29).
chosenNote(1,15,27).
chosenNote(1,16,25).
#const t=16.
configuration(solo).
part(1).
```

Figure 2: musing.lp: An example of a partial piece

Alternatively we could request the system to complete part of a piece. In order to do so, we provide the system with a set of $AnsProlog$ facts expressing the mode (major, minor, etc.), the notes which are already fixed, the number of notes in your piece, the configuration and the number of parts. Figure 2 contains an example of such file. The format is the same as the one returned from the system except that all the notes in the piece will have been assigned.

We then run the system just as before with the exception of adding `--piece=musing.lp` when we run `programBuilder.pl`. The system will then return all possible valid compositions that satisfy the criteria set out in the partial piece.

The $AnsProlog$ programs used in ANTON contains less than 200 lines (not including comments, empty lines and user defined pieces) and encodes 28 melodic and harmonic rules. Once instantiated, the generated programs range from 3,500 atoms and 13,400 rules (a solo piece with 8 notes) to 11,000 atoms and 1,350,000 rules (a 16 note duet). The system is licensed under the GPL and is available, along with example pieces, from `http://www.cs.bath.ac.uk/~mjb/` . Figure 3 contains an extract from a series of simple duets composed by ANTON.

It should be noted that ANTON's 200 lines of code contrast with the 8000 lines in Strasheela (Anders 2007) and 88000 in Bol (Bel 1998). For this reason we claim that our representation of the musical problem is easily read and understood.

Evaluation of ANTON

Practical Use

All this construction is of little use if the system is not practical to use, so we benchmarked a variety of solvers using the programs ANTON generated. Table 1 contains the times taken by a number of answer set solvers (SMODELS (Syrjänen and Niemelä 2001), SMODELS-IE (Brain, De Vos, and Satoh 2007), SMODELSCC (Ward and Schlipf 2004), CMODELS (Lierler and Maratea 2004) and CLASP (Gebser et al. 2007)) in composing a single piece of a given length. Likewise Table 2 contains the times taken to compose a two part piece of a given length. LPARSE (Syrjänen 2000) was used to ground the programs and its run time, typically around 30-60 seconds, is omitted from the results.

All times were recorded using a 2.4GHz AMD Athlon X2 4600+ processor, running a 64 bit version of OpenSuSE 10.3. All solvers were built in 32 bit mode. Each run was limited to 20 minutes of CPU time and 2Gb of

	smodels 2.32		smodels-ie 1.0.0		smodelscc 1.08	cmodels 3.75		clasp 1.0.5
Length	Default	Restarts	Default	Restarts	No lookahead	w/ zchaff	w/ MiniSAT	Default
4	1.02	1.03	0.09	0.09	1.17	0.33	0.39	0.22
6	2.43	2.43	0.38	0.38	2.58	0.64	0.85	0.46
8	5.16	5.16	1.03	1.04	4.94	1.06	1.62	1.01
10	12.25	11.72	2.58	2.59	8.55	1.54	2.63	1.33
12	28.25	46.13	8.08	15.14	11.36	2.42	4.04	2.27
14	40.62	140.00	10.50	43.54	18.78	3.14	6.05	3.48
16	101.05	207.25	29.40	69.53	27.94	4.01	9.40	4.62

Table 1: Time taken (in seconds) for a number of solvers generating a solo piece

	smodels 2.32		smodels-ie 1.0.0		smodelscc 1.08	cmodels 3.75		clasp 1.0.5
Length	Default	Restarts	Default	Restarts	No lookahead	w/ zchaff	w/ MiniSAT	Default
4	3.77	3.77	0.31	0.32	4.08	1.18	1.26	0.77
6	10.36	11.24	1.89	1.89	13.90	2.17	2.81	1.60
8	54.64	77.10	14.71	21.84	26.07	3.88	5.93	3.73
10	Time out	Time out	Time out	500.26	78.72	9.51	11.12	9.34
12	Time out	Time out	Time out	Time out	103.81	14.50	18.14	16.84
14	Time out	Time out	Time out	Time out	253.92	32.41	32.34	25.59
16	Time out	Time out	Time out	Time out	452.38	82.64	49.29	29.63

Table 2: Time taken (in seconds) for a number of solvers generating a duet

RAM. The $AnsProlog$ programs used are available from http://www.cs.bath.ac.uk/~mjb/ .

These results show that the system, when using the more powerful solvers, is fast enough to be used as a component in an interactive composition tool. Further work would be needed to support real time generation of music, but we are not too far away. We also note that the only solvers able to generate longer sequences in two parts all implement clause learning strategies, which suggests that the problem is particularly susceptible to this kind of technique.

Music Quality

Judging quality is a subjective process; after all we do not all like the same music. However we assert that the music is acceptable, at least by the standards of a student of composition, and at times there are moments of excitement. For the reader to judge we show in Figure 3 part of ANTON's Opus 1: *Twenty Short Pieces*; the audio files of the complete work can be found in http://cs.bath.ac.uk/~mjb.

ASP as the Representation and Reasoning Language

One of the main results reported in this paper is how easy it was to encode the rules in terms both of ease of expression and of ease of capturing the rules. Composers can think of ANTON as a testbed for experimentation with musical expertise that can be formulated as essential musical rules for all musical parameters[5] in order to build relations between those that either comply with a certain musical style or that open up new musical experiences. We have made the case that a sub-style of Renaissance Counterpoint and its melodic style can be represented with $AnsProlog$. The flexibility of creating different solutions based on the same rules offers the composer the opportunity to discover areas that he might have never thought of. It is recognised among musicians that an important component of composition is the use of the listener's expectations to obtain an effect. The composer plays on the listener by either satisfying his expectations or surprising him by not doing what he was anticipating. This facet of music is not confined to the experience of someone listening to a completed piece of music. It is inherent in the entire creative process, since a composer is also his own first listener. Subtle changes of $AnsProlog$ facts given to ANTON can give surprising results, as can small and skillful adjustments to the set of rules to produce a break with the previous set, thus allowing the composer to create the moldings for his own creations in a step-by-step process. This is a multi-faceted feedback-loop between writing the rules, listening to and examining the musical outcomes and modifying the rules if necessary. One of the most important musical tasks is to be able to work with impulse and resolution on multiple levels and with different voices simultaneously. This gathering and dissipation of musical energy (Thakar 1990) can happen in an infinite number of ways where all musical lines, defined by their various parameters, participate together. This process has to take into account all past musical events as well as the fragile balance between different voices and their parameters. Therefore, the modelling of rules that can capture musical impulse and resolution proves to be the most challenging aspect of writing programs with ANTON. There are currently no other programs known to us offering solutions in this direction and much of our future work will need to focus on this particular challenge.

There are also some negative points. One persistent problem was the lack of mature development support

[5]For a single note commonly known as pitch, loudness, timbre (sound quality) and duration.

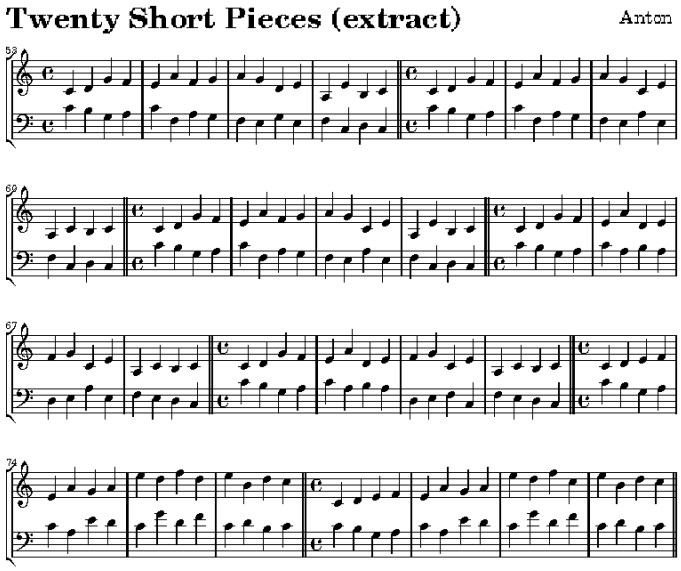


Figure 3: Part of a set of pieces composed by the system

tools, particularly debugging tools. SPOCK (Brain et al. 2007) was used, but as its focus is on computing the reasons behind the error, rather than the interface issues of explaining these reasons to the user, it was normally quicker to find bugs by looking at the last changes made and which regression tests failed. Generally, the bugs that were encountered where due to subtle mismatches between the intended meaning of a rule and the declarative reading of the rule used. For example the predicate `stepUp(P,T)` is used to represent the proposition "At time T, part P steps up to give the note at time T+1"; however, it could easily be misinterpreted as "At time T-1, part P steps up to give the note at time T". Which of these is used is not important, as long as the same declarative reading is used for all rules. With the first "meaning" selected for ANTON, the rule:

```
chosenNote(P,T,N+S) :-  chosenNote(P,T-1,N),
                        stepUp(P,T),
                        stepBy(P,T,S).
```

would not encode the intended progression of notes. To avoid these errors it would be possible to develop a system that translated rules into natural language, given the declarative reading of the propositions involved. It should then be relatively straightforward to check that the rule encoded what was intended.

The Future

Music Research

This system could develop in a number of novel ways. For example we might throw light on the compositional process by learning aspects of the rules, finding which are inconsistent or redundant, or determining the importance of rules. We could investigate whether there are "unspoken" rules, and experiment to find unacknowledged rules of composition. One particularly interesting possibility is using the system to generate a large set of pieces, acquiring human evaluations of the 'quality' of each and then using techniques such as inductive logic programming to infer rules for composing 'good' pieces.

So far we have only considered a particular style of Western music. However the framework should be applicable to other styles, especially formal ones. *e.g.* the rules of Hindustani classical music are taught in a traditional, oral, fashion, but we see no reason why our framework could not capture these. Recent work (Endrich 2008) indicates that there are indeed universal melodic rules, and the combination of the ASP methodology with this musical insight is an intriguing one.

In real life pieces some of the rules are sometimes broken. This could be simulated by one of a number of extensions to answer set semantics (preferences (Brain and De Vos 2003), consistency restoring rules, defensible rules, etc.). However how to systematise the knowledge of when it is acceptable to break the rules and in which contexts it is 'better' to break them is an open problem.

A major deficiency of the current system is the lack of rhythm, as all parts play all the time (with no rests), with notes of equal duration, which, while usual in some styles, stands in the way of a whole range of interesting variety. We have not considered rhythm so far, but one of us is already researching rhythmic structures and performance gesture (Boenn 2007), so in the longer term this may be incorporated.

Systems Development

The current system can write short melodies effectively and efficiently. Development work is still needed to extend this to entire pieces; we can start from these melodic fragments but a longer piece needs a variety of different harmonisations for the same melody, and related melodies with the same harmonic structure and a number of similar techniques. We have not solved the difficult global structure problem but our system is a starting point on which we can build a structure

that is hierarchical over time scales; we have a mechanism for building syntactically correct sentences, but these need to be built into paragraphs and chapters, as it were.

Our results seem to suggest that a real-time composition system is possible, which would open up the possibility for performance and improvisation. Profiling of the current system has indicated that some conceptually simple tasks, like parsing, are taking a disproportionate fraction of the runtime, and some engineering would assist in removing these problems. Clearly this is one of a number of system-like issues that need to be addressed. Also, the availability of a parallel answer set solver that implements clause learning would help in building this type of application.

An obvious extension to the composition of duets is to expand this to three and four parts, by adding inner voices, with their different rules.

Answer Set Synthesis

What we are doing is not answer set *programming* in the classical sense because we are not solving a problem *per se*; we are generating an arbitrary representative instance. Although this may seem like a subtle shift of emphasis, it has a number of interesting implications. Firstly for applications, this takes NMR into 'procedural synthesis' of all kinds of things — by describing what objects are, we can construct arbitrary examples. This has some interesting possibilities for computer games and virtual worlds where 'randomly generated' content is needed. This content has to be non-repetitive; increasingly it needs to be complex and structured (and have some fixed, hard properties — such as there must be a way out of the maze). In the case of things such as background music, high 'artistic merit' is not as important as consistent and non-repetitive.

This also raises questions for solver and language design. Most solvers are calibrated towards 'hard' problems, that are large (but tractable search spaces) with relatively few answer sets. However programs from this answer set synthesis school of application tend to have huge (intractable) search spaces with a very large number of answer sets. Thus the emphasis in solving shifts towards getting to an answer quickly (assuming that many paths lead to solutions) rather than trying to reduce and cover the search space efficiently. This is likely to influence the choice of branching heuristic. There are also a number of other interesting areas which come to mind. A metric for distance between answer sets would allow a solver to generate "things similar to the last solution", "things similar to the last solution but not too similar", "things as different as possible to the last solution", *etc*. Also there is a need for schemes for handling preferences ("it's not impossible to have A and B, but it should be avoided if possible") and probabilities ("there is a choice between A and B but in 90% of solutions it should be A"), particularly as part of the solving process, rather than needing to compute multiple solutions and then refining or optimising the choice between them, again probably in the heuristic.

Conclusion

In this paper, we have presented ANTON, the first algorithmic composing system that is capable of both melodic and harmonic composition. By using answer set programming as our modelling language for the technical rules that underpin music composition, we have obtained a highly flexible, extremely compact and efficient system. As all the rules are simultaneously available the system enables us to explore the rules themselves, to evaluate pieces for rule compliance, to complete partial systems, such as producing a melody consonant with a given harmony structure, as well as, more adventurously, to create new melodies.

We have demonstrated that current ASP systems can be used to generate aesthetically acceptable music within an appropriate time frame.

The development of ANTON opens up interesting research ideas both in the musicological direction and in declarative programming in general and more specifically answer set programming. In particular we have identified a number of ways in which ASP solvers could be extended so as to widen their application.

References

Anders, T. 2007. *Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System*. Ph.D. Dissertation, Queen's University, Belfast, Department of Music.

Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, 257–279. Kluwer Academic Publishers.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 1st edition.

Bel, B. 1998. Migrating Musical Concepts: An Overview of the Bol Processor. *Computer Music Journal* 22(2):56–64.

Boenn, G. 2007. Composing Rhythms Based Upon Farey Sequences. In *Digital Music Research Network Conference*.

Boulanger, R., ed. 2000. *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press.

Brain, M., and De Vos, M. 2003. Implementing OCLP as a Front End for Answer Set Solvers: From Theory to Practice. In *Proceedings of Answer Set Programming: Advances in Theory and Implementation (ASP'03)*. Ceur-WS.

Brain, M.; Crick, T.; De Vos, M.; and Fitch, J. 2006. TOAST: Applying Answer Set Programming to Superoptimisation. In *International Conference on Logic Programming*, LNCS. Springer.

Brain, M.; Gebser, M.; Pührer, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2007. "That is illogical captain!" – The Debugging Support Tool spock for Answer-Set Programs: System Description. In *Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07)*, 71–85.

Brain, M.; De Vos, M.; and Satoh, K. 2007. Smodels-ie : Improving the Cache Utilisation of Smodels. In Costantini, S., and Watson, R., eds., *Proceedings of the 4th Workshop on Answer Set Programming*, 309–314.

Brothwell, A., and ffitch, J. 2008. An Automatic Blues Band. In Barknecht, F., and Rumori, M., eds., *6th International Linux Audio Conference*, 12–17. Kunsthochscule für Medien Köln: LAC2008.

Buccafurri, F., and Caminiti, G. 2005. A Social Semantics for Multi-agent Systems. In *8th International Conference of Logic Programming and Nonmonotonic Reasoning*, volume 3662 of *LNCS*, 317–329. Springer.

Chuang, J. 1995. Mozart's Musikalisches Würfelspiel. `http://sunsite.univie.ac.at/Mozart/dice/`.

Cliffe, O.; De Vos, M.; and Padget, J. 2006. Specifying and Analysing Agent-based Social Institutions using Answer Set Programming. In Boissier, O.; Padget, J.; Dignum, V.; Lindemann, G.; Matson, E.; Ossowski, S.; Sichman, J.; and Vazquez-Salceda, J., eds., *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*, 99–113. Springer Verlag.

Cope, D. 2006. A Musical Learning Algorithm. *Computer Music Journal* 28(3):12–27.

Ebcioğlu, K. 1986. *An Expert System for Harmonization of Chorales in the Style of J.S. Bach*. Ph.D. Dissertation, State University of New York, Buffalo, Department of Computer Science.

Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1998. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In *KR'98: Principles of Knowledge Representation and Reasoning*. San Francisco, California: Morgan Kaufmann. 406–417.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2002. The DLV^K Planning System. In Flesca, S.; Greco, S.; Leone, N.; and Ianni, G., eds., *European Conference, JELIA 2002*, volume 2424 of *LNAI*, 541–544. Cosenza, Italy: Springer Verlag.

Endrich, A. 2008. *Building Musical Relationships*. In preparation. *seen in manuscript*.

Erdem, E.; Lifschitz, V.; Nakhleh, L.; and Ringe, D. 2003. Reconstructing the Evolutionary History of Indo-European Languages Using Answer Set Programming. In Dahl, V., and Wadler, P., eds., *PADL*, volume 2562 of *LNCS*, 160–176. Springer.

Fux, J. 1965, orig 1725. *The Study of Counterpoint from Johann Joseph Fux's Gradus ad Parnassum*. W.W. Norton.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-Driven Answer Set Solving. In *Proceeding of IJCAI07*, 386–392.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A New Grounder for Answer Set Programming. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *LPNMR*, volume 4483 of *LNCS*, 266–271. Springer.

Grell, S.; Schaub, T.; and Selbig, J. 2006. Modelling biological networks by action languages via answer set programming. In Etalle, S., and Truszczyński, M., eds., *Proceedings of the International Conference on Logic Programming (ICLP'06)*, volume 4079 of *LNCS*, 285–299. Springer-Verlag.

Konczak, K. 2006. Voting Theory in Answer Set Programming. In Fink, M.; Tompits, H.; and Woltran, S., eds., *Proceedings of the Twentieth Workshop on Logic Programming (WLP'06)*, number INFSYS RR-1843-06-02 in Technical Report Series, 45–53. Technische Universität Wien.

Leach, J. L. 1999. *Algorithmic Composition and Musical Form*. Ph.D. Dissertation, University of Bath, School of Mathematical Sciences.

Lierler, Y., and Maratea, M. 2004. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *LNCS*, 346–350. Springer.

Lifschitz, V. 2002. Answer set programming and plan generation. *J. of Artificial Intelligence* 138(1-2):39–54.

Nienhuys, H.-W., and Nieuwenhuizen, J. 2003. Lilypond, A System For Automated Music Engraving. In *Proceedings of the XIV Colloquium on Musical Informatics*.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. A A-Prolog Decision Support System for the Space Shuttle. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Represenation and Reasoning*. Stanford (Palo Alto), California, US: American Association for Artificial Intelligence Press.

Polleres, A. 2005. Semantic Web Languages and Semantic Web Services as Application Areas for Answer Set Programming. In *Nonmonotonic Reasoning, Answer Set Programming and Constraints*. IBFI.

Rohrmeier, M. 2006. Towards modelling harmonic movement in music: Analysing properties and dynamic aspects of pc set sequences in Bach's chorales. Technical Report DCRR-004, Darwin College, University of Cambridge.

Ruffolo, M.; Leone, N.; Manna, M.; Saccà, D.; and Zavatto, A. 2005. Exploiting ASP for Semantic Information Extraction. In De Vos, M., and Provetti, A., eds., *Answer Set Programming*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Syrjänen, T., and Niemelä, I. 2001. The Smodels System. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*.

Syrjänen, T. 2000. *Lparse 1.0 User's Manual*. Helsinki University of Technology.

Thakar, M. 1990. *Counterpoint*. New Haven.

Ward, J., and Schlipf, S. 2004. Answer Set Programming with Clause Learning. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *LNCS*. Springer.

Xenakis, I. 1992. *Formalized Music*. Stuyvesant, NY, USA: Bloomington Press.

Tools for Representing and Reasoning about Biological Models in Action Language $\mathcal{C}$

Steve Dworschak and **Torsten Grote** and **Arne König** and **Torsten Schaub** and **Philippe Veber**
Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam, Germany

Abstract

We elaborate upon the usage of action language $\mathcal{C}$ for representing and reasoning about biological models. First, we provide a simple extension of $\mathcal{C}$ allowing for variables and show its usefulness in modeling biochemical reactions according to the well-known model of BIOCHAM. Second, we show how the biological action description language $\mathcal{C}_{TAID}$ can be mapped onto $\mathcal{C}$. Finally, we describe a toolbox for using action languages, including among them, a compiler mapping $\mathcal{C}$ and $\mathcal{C}_{TAID}$ to logic programs under answer sets semantics along with a web-service integrating different front- and back-ends for addressing dynamical systems by means of action description languages via answer set programming. This is accompanied by an empirical evaluation with existing systems for processing action description languages.

Introduction

We elaborate upon *action languages* (Gelfond and Lifschitz 1998) for qualitative modeling of biological networks. Action languages are formal models used for reasoning about the effects of actions, while being close to natural language. Central to this approach to formalizing actions is the concept of a transition system, which constitutes its semantic underpinning. The first action language for representing and reasoning about biological networks was introduced in (Tran and Baral 2004; Baral et al. 2004; Tran 2006) and further extended in (Dworschak et al. 2007) leading to action language $\mathcal{C}_{TAID}$.

In what follows, we extend the overall approach in several ways while centering it on the classical action language $\mathcal{C}$ (Giunchiglia and Lifschitz 1998). To begin with, we provide a simple extension of $\mathcal{C}$ allowing for variables and show its usefulness in modeling biochemical reactions according to the well-known model of BIOCHAM. Similar to the approach taken in the dlv^k system based on action language $\mathcal{K}$ (Eiter et al. 2003a), we delegate the treatment of variables to Answer Set Programming (ASP; (Baral 2003)) in order to be able to use ASP grounders for variable instantiation. Second, we provide a translation mapping the biologically motivated action description language $\mathcal{C}_{TAID}$ onto $\mathcal{C}$ and give a result fixing the formal correspondence. This allows us to further develop $\mathcal{C}_{TAID}$ within a broader and well-established framework, avoiding further dedicated implementations. Moreover, it provides $\mathcal{C}_{TAID}$ with access to further implementations of $\mathcal{C}$, like $CCalc$ (Giunchiglia et al. 2004) or $CPlan$ (Castellini, Giunchiglia, and Tacchella 2003) (even though they cannot harness existing ASP grounders for variable treatment). Finally, we describe a toolbox for using action languages, including among them, a compiler mapping $\mathcal{C}$ and $\mathcal{C}_{TAID}$ to logic programs under answer sets semantics along with a web-service integrating different front- and back-ends for addressing dynamical systems by means of action description languages via answer set programming. Our tools are designed for an easy and flexible integration with existing open source tools via pipes, in particular, ASP grounders and solvers, as well as further front- and back-ends. This is accompanied by an empirical evaluation with existing systems for processing action description languages.

Background

Answer Set Programming. Our language is built from a set $\mathcal{F}$ of *function* symbols (including the natural numbers), a set $\mathcal{V}$ of *variable* symbols, and a set $\mathcal{P}$ of *predicate* symbols. The set $\mathcal{T}$ of *terms* is the smallest set containing $\mathcal{V}$ and all expressions of the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$ and $t_i \in \mathcal{T}$ for $1 \leq i \leq n$. The set $\mathcal{A}$ of *atoms* contains expressions of the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ and $t_i \in \mathcal{T}$ for $1 \leq i \leq n$. A *literal* is an atom a or its negation $\neg a$; both can be preceded by default negation, denoted as $not\ a$ and $not\ \neg a$, respectively. For $a \in \mathcal{A}$, we let $\overline{a} = \neg a$ and $\overline{\neg a} = a$. A *logic program* over $\mathcal{A}$ is a set of *rules* of the form

$$a \leftarrow b_1, \ldots, b_m, not\ c_{m+1}, \ldots, not\ c_n \qquad (1)$$

where a, b_i, c_j are literals over $\mathcal{A}$ for $0 < i \leq m < j \leq n$. For a rule r as in (1), let $head(r) = a$, $body(r)^+ = \{b_1, \ldots, b_m\}$, and $body(r)^- = \{c_{m+1}, \ldots, c_n\}$. Given an expression $e \in \mathcal{T} \cup \mathcal{A}$, let $var(e)$ denote the set of all variables occurring in e; analogously, $var(r)$ gives all variables in rule r. The *ground instantiation* of a program P is defined as $grd(P) = \{r\theta \mid r \in P, \theta : var(r) \rightarrow \mathcal{U}\}$, where $\mathcal{U} = \{t \in \mathcal{T} \mid var(t) = \emptyset\}$; analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$ is the set of all ground atoms. A set $X \subseteq grd(\mathcal{A}) \cup \overline{grd(\mathcal{A})}$ is a (consistent) *answer set* of a program P over $\mathcal{A}$, if X is the $\subseteq$-smallest model of

$$\{head(r) \leftarrow body(r)^+ \mid r \in grd(P), body(r)^- \cap X = \emptyset\}\ .$$

Action Language $\mathcal{C}$. Action languages use *fluents* to describe the states of a system and *actions* influence the values of fluents. In $\mathcal{C}$ (and $\mathcal{C}_{TAID}$), *static laws* describe properties between fluents that need to be satisfied in every state of the system. *Dynamic laws* describe the effects of actions, that is, how the system evolves when actions are executed.

More formally, we consider *action language* $\mathcal{C}$ (Gelfond and Lifschitz 1998) over a Boolean *action signature* $\langle B, F, A\rangle$, where B is the set $\{f, t\}$ of truth values, F is a set of *fluent names*, and A is a set of *action names*. In $\mathcal{C}$, an *action description* $D_\mathcal{C}$ over a signature $\langle B, F, A\rangle$ consists of *static laws*, such as

$$(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi) \tag{2}$$

and *dynamic laws* of the form

$$(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi\ \mathbf{after}\ \omega), \tag{3}$$

where φ and ψ are propositional combinations of fluent names and ω is a propositional combination of fluent and action names. Every action description $D_\mathcal{C}$ induces a unique transition system $\mathcal{T}_\mathcal{C}(D_\mathcal{C}) = \langle S, V, R\rangle$, where S is a set of *states*, V is a function determining fluents values in state s, and R is a relation containing all possible transitions between states. A *trajectory* $s_0, A_1, s_1, \ldots, s_{n-1}, A_n, s_n$ in a transition system $\langle S, V, R\rangle$ is a sequence of sets of actions $A_i \subseteq A$ and states $s_i \in S$ where $(s_{i-1}, A_i, s_i) \in R$ for $0 \leq i \leq n$. Intuitively, a trajectory represents one possible history (or simply path) within a transition system. In (Gelfond and Lifschitz 1998), several syntactic extensions are defined. For instance, the rule $(\omega\ \mathbf{may\ cause}\ \varphi\ \mathbf{if}\ \psi)$ is a shorthand for $(\mathbf{caused}\ \varphi\ \mathbf{if}\ \varphi\ \mathbf{after}\ \psi \wedge \omega)$. Similarly, $(\mathbf{inertial}\ \varphi)$ is a shorthand for $(\mathbf{caused}\ \varphi\ \mathbf{if}\ \varphi\ \mathbf{after}\ \varphi)$. We refer to (Gelfond and Lifschitz 1998) for more detailed definitions.

Besides an action description language, both $\mathcal{C}$ and $\mathcal{C}_{TAID}$ define a *query language*. We implemented $\mathcal{R}$ (Gelfond and Lifschitz 1998) as the query language for $\mathcal{C}$ and the query mechanisms described in (Dworschak et al. 2007) for $\mathcal{C}_{TAID}$. In this paper, we focus only on the transition systems and our toolbox realizing the different encodings, so we omit a detailed description of query languages and the different reasoning modes. In the biological setting, queries combined with reasoning modes like planning and explanation are used to answer biological questions. For example, one is able to determine whether certain states can be reached in molecular networks, queries about existence of paths can be answered and it is possible to do high-level experiment planning.

Encoding Action Language $\mathcal{C}$

For implementing action language $\mathcal{C}$, we build upon the translation to ASP described in (Lifschitz and Turner 1999). Let $D_\mathcal{C}$ be an action description over signature $\langle B, F, A\rangle$. We require $D_\mathcal{C}$ to be *definite*, that is, the heads φ of laws $(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi)$ and $(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi\ \mathbf{after}\ \omega)$ are fluent literals (or the constant $\bot$). Furthermore, ψ is a conjunction of fluent literals and ω is a conjunction of fluent and/or action literals. In what follows, we denote φ by f, ψ by $g_1 \wedge \ldots \wedge g_m$ and ω by $l_1 \wedge \ldots \wedge l_n$.

We define a logic program $lp_n(D_\mathcal{C})$ whose answer sets correspond to trajectories of length n in the transition system induced by $D_\mathcal{C}$. $lp_n(D_\mathcal{C})$ contains atoms $a(t)$ and $f(t)$ for each $a \in A$, $f \in F$ and $t = 0, \ldots, n$. For each static law $(\mathbf{caused}\ f\ \mathbf{if}\ g_1 \wedge \ldots \wedge g_m)$ in $D_\mathcal{C}$, $lp_n(D_\mathcal{C})$ contains for each $t = 0, \ldots, n$ a rule

$$f(t) \leftarrow not\ \overline{g_1(t)}, \ldots, not\ \overline{g_m(t)}\ .$$

Analogously, each dynamic law $(\mathbf{caused}\ f\ \mathbf{if}\ g_1 \wedge \ldots \wedge g_m\ \mathbf{after}\ l_{m+1} \wedge \ldots \wedge l_n)$ in $D_\mathcal{C}$, adds to $lp_n(D_\mathcal{C})$ for each $t = 0, \ldots, n-1$ a rule

$$f(t{+}1) \leftarrow not\ \overline{g_1(t+1)}, \ldots, not\ \overline{g_m(t+1)}, l_{m+1}(t), \ldots, l_n(t)\ .$$

Furthermore, $lp_n(D_\mathcal{C})$ contains

$$\begin{array}{rcl@{\qquad}rcl} \neg a(t) & \leftarrow & not\ a(t), & \neg e(0) & \leftarrow & not\ e(0), \\ a(t) & \leftarrow & not\ \neg a(t), & e(0) & \leftarrow & not\ \neg e(0) \end{array}$$

for each $a \in A$, $t = 0, \ldots, n$ and each $e \in F$.

Our implementation of the encoding allows to use variables when writing rules in $\mathcal{C}$. This is done by delegating the grounding of variables to the grounding process of the underlying logic program. To this end, we start by extending the syntax of $\mathcal{C}$ by a trailing keyword **where** followed by domain predicates for binding the variables occurring in the actual causal laws. To be precise, the causal laws in (2) and (3) are extended as follows:

$$(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi\ \mathbf{where}\ \delta) \tag{4}$$

$$(\mathbf{caused}\ \varphi\ \mathbf{if}\ \psi\ \mathbf{after}\ \omega\ \mathbf{where}\ \delta) \tag{5}$$

where φ, ψ, and ω are as defined in (2) and (3), except for containing variables, and δ is a combination of non-fluent and non-action atoms such that $var(\varphi) \cup var(\psi) \cup var(\omega) \subseteq var(\delta)$. Intuitively, δ captures static domain information used for binding the variables in φ, ψ, ω. The concept of a definite action description generalizes in the obvious way, restricting δ to conjunctions of non-fluent and non-action atoms. Now, given such a definite action description $D_\mathcal{C}$, the variable-tolerating extension of $lp_n(D_\mathcal{C})$ is obtained from $lp_n(D_\mathcal{C})$ by extending the body of each resulting logic programming rule by $d_1, \ldots, d_o$ whenever the causal law contains the condition $\mathbf{where}\ d_1 \wedge \cdots \wedge d_o$.

Let us illustrate the practical impact of this pragmatic extension by modeling the Biochemical Abstract Machine (BIOCHAM; (Fages, Sollman, and Chabrier-Rivier 2004; Chabrier-Rivier et al. 2004)), used to build biochemical systems. The biological background is indeed very easy. A modeled scenario consists of different chemical reactions that specify relations between different compounds. *Reactants* are compounds that need to be present that a reaction can take place and *products* are compounds that will be present after a reaction took place. One can model this scenario using $\mathcal{C}$ with the following rules. At first, our syntax requires to specify a preamble where actions and fluents are defined:

```
<action> occurs(R)  <where> reaction(R).
<fluent> present(P) <where> compound(P).
```

Strings enclosed in <> are keywords, variables start with uppercase letters and lines end with a dot. That is, for every term t belonging to the extension of the predicate reaction, we introduce the actions occurs(t). For every term t belonging to the extension of the predicate compound, we introduce the fluents present(t).

We now can define the dynamics of the system:

```
<caused> present(P)
  <after> occurs(R)
  <where> reaction(R), compound(P),
          product(P,R).

<caused> <false>
  <after> occurs(R),-present(P)
  <where> reaction(R),compound(P),
          reactant(P,R).

occurs(R) <may cause> -present(P)
  <where> reaction(R),compound(P),
          reactant(P,R).

<inertial> present(P) <where> compound(P).
<inertial> -present(P) <where> compound(P).
```

The first rule states that a compound P is present after a reaction R occurred producing P. The second rule is a constraint enforcing all compounds P to be present if a reaction occurs where P is a reactant of. Note that negation is denoted as - and <false> as well as <true> are keywords for the two Boolean constants. The third rule models a certain non-determinism: The semantics of BIOCHAM defines that after a reaction occurs, it remains unclear whether the reactants are still present or not. The reason is that the semantics abstract from concentrations of compounds. That is, we consider two cases: In one transition we assume that the compound P was fully consumed, modeled as -present(P). The other transition is that P remains to be present. The last two rules state that compounds that are not affected by reactions do not change their value [1].

Let us briefly detail how variables are passed through the encoding proposed in (Lifschitz and Turner 1999). For this, consider the first rule of the BIOCHAM example:

```
<caused> present(P)
  <after> occurs(R)
  <where> reaction(R), compound(P),
          product(P,R).
```

It is translated to the following logic rule:

```
present_fluent(P,T+1)
:- occurs_action(R,T),
   reaction(R),compound(P),product(P,R),
   time(T).
```

Apart from the time-parameter T, we attach variable P to the fluent present and R to the action occurs. The domain information given in the <where> statement is then passed as grounding information to the logic program rule.

[1] These rules represents the frame axiom: Compounds that are not consumed remain present, absent compounds that where not produced remain absent.

The last pending issue is to specify the domains:

```
compound(a). compound(b).
reaction(r1).
reactant(a,r1). product(b,r1).
```

The database is represented as a logic program. It can be seen as static knowledge attached to the modeled dynamic behavior of the system. In most cases, the database only contains facts. In the example, we are now able to reason about a scenario with two compounds and one reaction. An encoding of a simple version of the biological textbook example of the *Mitogen-activated protein kinase (MAPK)*[2] including 23 products and 30 reactions, yields a problem instance containing 147 facts. One of the advantages using variables is that the system can be easily enhanced by extending the database, that is, without touching the specification of the dynamics.

Mapping $\mathcal{C}_{TAID}$ to $\mathcal{C}$

As with $\mathcal{C}$, an *action description* in $\mathcal{C}_{TAID}$ is given relative to an action signature $\langle B, F, A\rangle$. The major conceptual difference between $\mathcal{C}_{TAID}$ and $\mathcal{C}$ is that the latter implicitly treats actions to be exogenous. That is, all actions might occur at every time-point as long as their effects do not lead to a contradiction. For biological purposes, this behavior is inappropriate. Unlike this, $\mathcal{C}_{TAID}$ allows for specifying explicit conditions when actions are executed or not. For example, using $\mathcal{C}_{TAID}$'s *triggering rule*, we can describe properties when (re)actions must be executed immediately. Furthermore, $\mathcal{C}_{TAID}$ offers the following constructs: *Inhibition rules* express when actions must not be executed and *allowance rules* express that actions might occur, but are not forced to. A *default* expresses that a fluent takes a certain value unless it is known otherwise. *No-concurrency constraints* allow to control the parallel execution of actions. In a more formal way, an action description in $\mathcal{C}_{TAID}$ contains expressions of the following form:

$$(a \textbf{ causes } \varphi \textbf{ if } \psi)$$
$$(\varphi \textbf{ if } \psi)$$
$$(\varphi \textbf{ triggers } a)$$
$$(\varphi \textbf{ allows } a)$$
$$(\varphi \textbf{ inhibits } a)$$
$$(\textbf{noconcurrency } \omega)$$
$$(\textbf{default } f),$$

where a is an action and ω is either an action or a conjunction of action literals, φ and ψ are conjunctions of fluent literals and f is a fluent literal. We refer to (Dworschak et al. 2007) for a more detailed description of $\mathcal{C}_{TAID}$.

We now describe our translation of $\mathcal{C}_{TAID}$ into $\mathcal{C}$. To this end, we need to extend the action signature to accommodate some control information. To be precise, we add the fluents $ih(a)$, $tr(a)$, $ex(a)$, $al(a)$ for each action name a. Intuitively, these fluents signal properties reflecting the behavior of inhibition, triggering, and allowance rules. With

[2] http://en.wikipedia.org/wiki/MAPK

them, we can define the mapping of rules in $\mathcal{C}_{TAID}$ to rules in $\mathcal{C}$ as follows.

Definition 1 *Let $D_{\mathcal{C}_{TAID}}$ be an action description in $\mathcal{C}_{TAID}$ over action signature $\langle B, F, A\rangle$. The corresponding action description $D_{\mathcal{C}}$ in $\mathcal{C}$ over action signature*

$$\langle B, F \cup \bigcup_{a\in A}\{ih(a), tr(a), ex(a), al(a)\}, A\rangle \quad (6)$$

is defined as follows:

1. *For each action name $a \in A$, action description $D_{\mathcal{C}}$ contains the static laws*

$$(\mathbf{caused}\ \neg ih(a)\ \mathbf{if}\ \neg ih(a)),$$
$$(\mathbf{caused}\ \neg tr(a)\ \mathbf{if}\ \neg tr(a)),$$
$$(\mathbf{caused}\ \neg ex(a)\ \mathbf{if}\ \neg ex(a)), \text{and}$$
$$(\mathbf{caused}\ \neg al(a)\ \mathbf{if}\ \neg al(a)).$$

2. *For each dynamic law $(a\ \mathbf{causes}\ \varphi\ \mathbf{if}\ \psi)$ in $D_{\mathcal{C}_{TAID}}$, where $\varphi = f_1 \wedge \ldots \wedge f_m$, $D_{\mathcal{C}}$ contains the laws*

$$(\mathbf{caused}\ f_i\ \mathbf{if}\ \top\ \mathbf{after}\ \psi \wedge a)$$

for each f_i where $1 \leq i \leq m$.

3. *For each static law $(\varphi\ \mathbf{if}\ \psi)$ in $D_{\mathcal{C}_{TAID}}$, where $\varphi = f_1 \wedge \ldots \wedge f_m$, $D_{\mathcal{C}}$ contains the laws*

$$(\mathbf{caused}\ f_i\ \mathbf{if}\ \psi)$$

for each f_i where $1 \leq i \leq m$.

4. *For each allowance rule $(\varphi\ \mathbf{allows}\ a)$ in $D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ al(a)\ \mathbf{if}\ \varphi)\ \text{and}$$
$$(\mathbf{caused}\ \bot\ \mathbf{if}\ \top\ \mathbf{after}\ \neg al(a) \wedge a).$$

5. *For each triggering rule $(\varphi\ \mathbf{triggers}\ a)$ in $D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ tr(a)\ \mathbf{if}\ \varphi),$$
$$(\mathbf{caused}\ ex(a)\ \mathbf{if}\ \top\ \mathbf{after}\ a),$$
$$(\mathbf{caused}\ \bot\ \mathbf{if}\ \neg ex(a)\ \mathbf{after}\ tr(a) \wedge \neg ih(a))\ \text{and}$$
$$(\mathbf{caused}\ \bot\ \mathbf{if}\ ex(a)\ \mathbf{after}\ \neg tr(a)).$$

6. *For each inhibition rule $(\varphi\ \mathbf{inhibits}\ a)$ in $D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ ih(a)\ \mathbf{if}\ \varphi)\ \text{and}$$
$$(\mathbf{caused}\ \bot\ \mathbf{if}\ \top\ \mathbf{after}\ ih(a) \wedge a).$$

7. *For each constraint $(\mathbf{noconcurrency}\ \omega)$ in $D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ \bot\ \mathbf{if}\ \top\ \mathbf{after}\ \omega).$$

8. *For each default rule $(\mathbf{default}\ f)$ in $D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ f\ \mathbf{if}\ f).$$

9. *For each $f \in F$, such that $(\mathbf{default}\ f) \notin D_{\mathcal{C}_{TAID}}$, $D_{\mathcal{C}}$ contains*

$$(\mathbf{caused}\ f\ \mathbf{if}\ f\ \mathbf{after}\ f)\ \text{and}$$
$$(\mathbf{caused}\ \neg f\ \mathbf{if}\ \neg f\ \mathbf{after}\ \neg f).$$

The symbols $\top$ and $\bot$ denote the Boolean constants for t and f in B.

The rules in 1. state that $ih(a)$, $tr(a)$, $ex(a)$, and $al(a)$ are set to be *false* by default. As described in 4.–6., they are only set *true* when certain properties hold. There is a direct correspondence between static and dynamic rules in $\mathcal{C}_{TAID}$ and $\mathcal{C}$ (cf. 2. and 3.) except the fact that conjunctions in heads are split in order to get a definite action description. An allowance rule is expressed using a static rule setting $al(a)$ and a dynamic rule that can be viewed as a constraint eliminating transitions where action a occurred while $al(a)$ was *false* (cf. 4.). Rules given in 5. express triggering rules: whenever a trigger is applicable, $tr(a)$ is set and every execution of an action a causes $ex(a)$ to be true. The second dynamic rule eliminates transitions where the conditions for a triggering rule were satisfied but a was not executed, that is, $ex(a)$ is *false*. Since $\mathcal{C}_{TAID}$ gives inhibition rules priority over triggering rules, the constraint is only applicable if $\neg ih(a)$ is satisfied. The third dynamic rule eliminates transitions where a is executed without having an applicable trigger. Inhibition rules are mapped in the same way as allowance rules (cf. 6.). No-concurrency constraints and defaults in $\mathcal{C}_{TAID}$ have a direct correspondence to rules in $\mathcal{C}$ (cf. 7. and 8.). Given that fluents are implicitly inertial[3] in $\mathcal{C}_{TAID}$ but not in $\mathcal{C}$, for each fluent there is a dynamic rule in $D_{\mathcal{C}}$ that expresses inertial behavior (cf. 9.).

We can show the following result:

Theorem 1 *Let $D_{\mathcal{C}_{TAID}}$ be an action description in $\mathcal{C}_{TAID}$ over action signature $\langle B, F, A\rangle$ and let $D_{\mathcal{C}}$ be the corresponding action description in $\mathcal{C}$ over the action signature in (6) generated from $D_{\mathcal{C}_{TAID}}$ using the mapping in Definition 1.*

Then, each trajectory in the transition system $\mathcal{T}_{\mathcal{C}}(D_{\mathcal{C}})$ (as defined in (Gelfond and Lifschitz 1998)), corresponds to a unique trajectory in the transition system induced by $D_{\mathcal{C}_{TAID}}$ (as defined in (Dworschak et al. 2007)) and vice versa.

Problem descriptions in $\mathcal{C}_{TAID}$ can now be dealt with the general-purpose language $\mathcal{C}$. That is, we do not need a rather complicated (re)definition of semantics in order to describe transition systems having a biological background using $\mathcal{C}_{TAID}$. It can now be seen as another layer of interface on top of the action description language $\mathcal{C}$.

In the following sections we describe how the different encodings can be used in our toolchain and how they perform compared to other implementations.

The `BioPlan` System

Our approach to representing and reasoning about biological models is as follows: at first, the biological model needs to be specified in the action description language of $\mathcal{C}$ or $\mathcal{C}_{TAID}$. This description is compiled into a logic program as described above and subsequently dealt with using an ASP system, usually composed of a grounder and a solver. Once the logic program is solved, the answers of the solver need to

[3] That is, fluents that are neither defaults nor affected directly or indirectly by dynamic rules do not change their value in a transition.

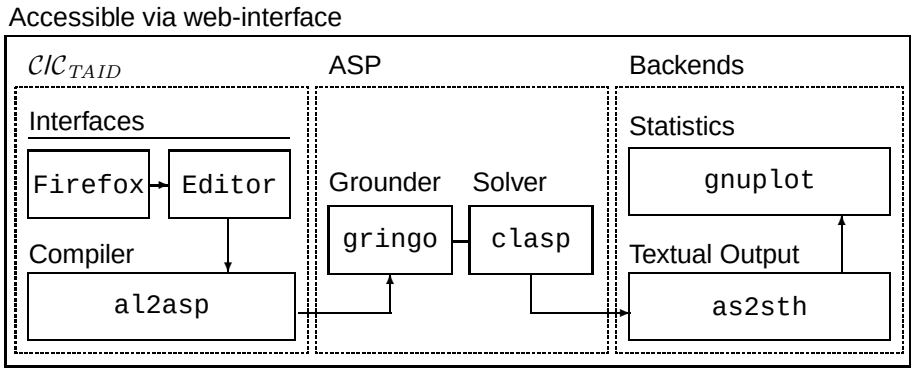


Figure 1: Overview of our system architecture

be put back in correspondence to the original problem specification. Finally, the obtained data needs to be interpreted in a biologically meaningful way by a human expert. An overview of our system is given in Figure 1.

Interfaces

To begin with, we have a closer look at the interfaces to our system. Our system is able to handle the discussed action descriptions in $\mathcal{C}$ and $\mathcal{C}_{TAID}$. For action descriptions in $\mathcal{C}$, one has to write down the rules in an editor, as shown in the BIOCHAM example. This is of course also possible using $\mathcal{C}_{TAID}$. Since $\mathcal{C}_{TAID}$ has a much more biological orientation than $\mathcal{C}$, we offer another interface for $\mathcal{C}_{TAID}$ that is more intuitive for users having a purely biological background: A graphical interface that was built as a Firefox browser extension. It allows for building rules as a graph whose nodes (fluents and actions) and edges (causal relationships) correspond to the underlying expressions of $\mathcal{C}_{TAID}$. An example is shown in Figure 2. Since this paper has more a technical orientation, we are not detailing a biological example using $\mathcal{C}_{TAID}$.

Compiler

Once the description is done, it is passed to our compiler `al2asp`. As mentioned before, this program is able to handle the described languages and their different encodings that need to be given via command line options:

`al2asp -l c`	direct $\mathcal{C}$ to ASP encoding
`al2asp -l c_taid`	direct $\mathcal{C}_{TAID}$ to ASP encoding
`al2asp -l c_taid2c`	$\mathcal{C}_{TAID}$ to $\mathcal{C}$ encoding

While the two first commands yield a logic program[4], the last one outputs rules in $\mathcal{C}$.[5]

`al2asp` is implemented in `C++` and freely available at (BioASP Tools). Notably, `al2asp` relies on scanner and parser generators `flex` and `bison++`, making it easily amenable to language extensions.

[4]The direct $\mathcal{C}_{TAID}$ to ASP encoding implements a slightly modified encoding according to the one given in (Dworschak et al. 2007) that is not discussed in this paper.

[5]One can just reuse the tool to complete the encoding: `al2asp -l c_taid2c <file.desc> | al2asp -l c`.

An `al2asp` generated logic program containing variables appears incomplete. The additional logic program providing the binding information must be concatenated to the output of `al2asp` in order to get the resulting logic program that can be grounded. This ground program expresses the transition system described by the original description in $\mathcal{C}$.

ASP Tools

Reconsidering Figure 1, the resulting logic program is dealt with by an ASP system, consisting of a grounder and a solver component. As discussed, the logical representation of an action description may contain object variables that are passed on to the grounder. Our grounder, `gringo` (Gebser, Schaub, and Thiele 2007)[6], systematically replaces all variables by ground terms, while aiming at producing a compact propositional program. The resulting program is then passed to the ASP solver, `clasp` (Gebser et al. 2007b; 2007a)[7], which computes the stable models (see (Baral 2003) for details) of the program. Each such model represents a valid trajectory in the transition system induced by the original action description.

Backends

The action description for the BIOCHAM system combined with the underlying domain induces the transition system given in Figure 3.

Given that fluent and action names are changed in the logical encoding (ie. an additional time parameter appears as additional argument) as well as the obtained solutions appear in an unsorted way, the output of an ASP system must be transformed in a more readable and problem-oriented format. To this end, we offer different possibilities to present the output using the program `as2sth`: One possibility is a textual representation of the trajectories that gives a detailed overview of actions and states involved in a given solution. To illustrate this, recall our BIOCHAM example and consider the answer sets representing all 8 trajectories of length 1. Our interface displays them as follows.

[6]`http://www.cs.uni-potsdam.de/gringo`

[7]`http://www.cs.uni-potsdam.de/clasp`

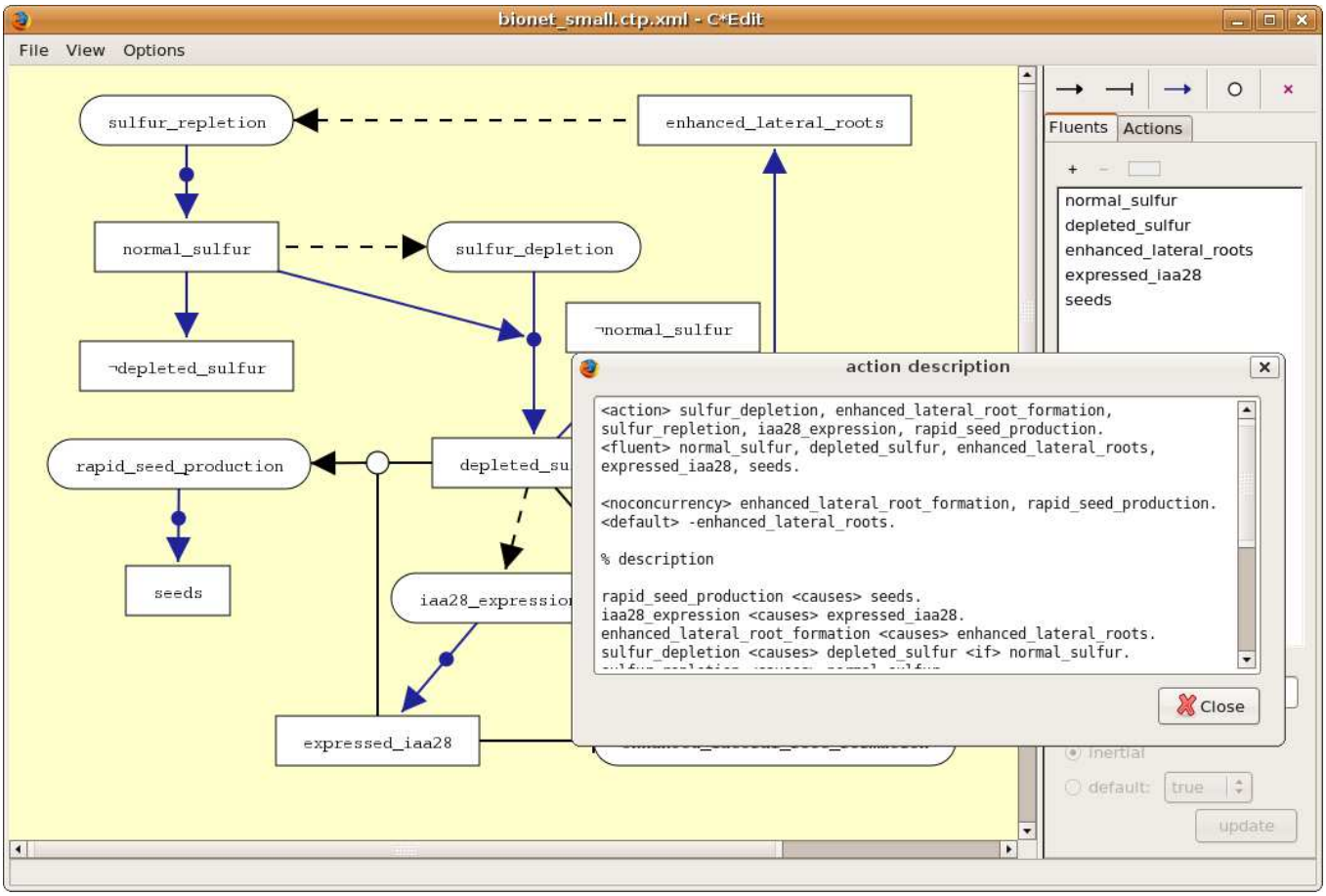


Figure 2: Screenshot of our graphical user interface for $\mathcal{C}_{TAID}$. Problem descriptions can be modeled in a graphical way by combining nodes with different arrows that correspond to rules in $\mathcal{C}_{TAID}$. The textual representation is generated by the program in order to process it with our compiler or to directly send the description to our web-based service.

```
## ANSWER 1 ################
0  A  +  occurs(r1)
0  F  +  present(a)
0  F  -  present(b)
1  F  -  present(a)
1  F  +  present(b)
## ANSWER 2 ################
...
## ANSWER 8 ################
0  F  +  present(a)
0  F  -  present(b)
1  F  +  present(a)
1  F  -  present(b)
## SUMMARY #################
models: 8
```

The first column denotes the timestep, the second one the type of the logic literal (action or fluent), the third one the value of the literal (true or false) and the last one the original name as used in the action description.

This method becomes inapplicable when the number of solutions increases, which is the case in most of the biological applications. To this end, another possibility is to generate `csv` output that can be processed with external programs like database systems, statistical tools, etc.

A third possibility is to use our built in `gnuplot` interface: We currently provide some statistical post-processing counting fluent values and actions at each time step in all trajectories. For example, let us assume that a fluent f appears to be true at a certain timestep t in all trajectories. When presenting all occurrences of fluents (or actions) in a graphical way, one can easily see that fluent f is essential for having solutions.[8] Although our simple BIOCHAM example focuses on the transition system (having no queries at all), the graphical representation can already be useful to get an idea. Reconsider the transition system given in Figure 3. Figure 4 is the graphical representation of all 128 trajectories having a length of 6. It is easy to see that there is a direct correspondence between the presence of compound `a` and `b`. While `a` tends to decrease, `b` tends to increase.[9] It is nearly

[8] This can also be seen as a cautious reasoning mode.

[9] Indeed, this outcome seems to be trivial since this is exactly the relation between the two compounds that was modeled before. But on larger scale examples it is possible to identify relations that

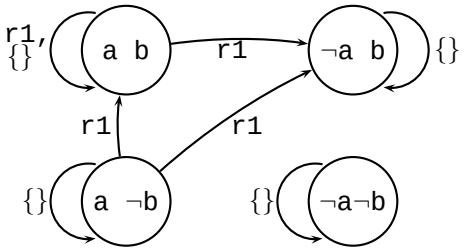


Figure 3: Transition system of the BIOCHAM example. `a` and `b` are shorthands for fluents `present(a)` and `present(b)`, `r1` is a shorthand for action `occurs(r1)`. $\{\}$ denotes the empty action, that is, no action is executed in a transition labeled like this. Note that the loop at node $\{\mathtt{a}, \mathtt{b}\}$ describes two transitions.

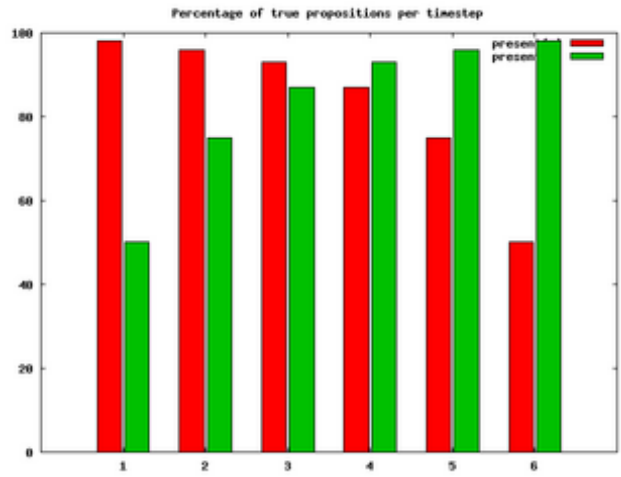


Figure 4: Graphical representation of all trajectories of the BIOCHAM example having length 6. Y-Axis denotes the percentage of true propositions (resp. the presence of compounds), and X-Axis denotes the timesteps. The two different bars represent the compounds `a` and `b`. For example, consider the second bar at timestep 1: it denotes that `present(b)` is true at timestep 1 in 50% out of all answer sets.

impossible to gain such information by only looking at the calculated trajectories.

Toolchain Access

The whole reasoning tool is accessible in two ways. The first possibility is to download the tools described in Figure 1 from (BioASP Tools) and to run them on a local machine. We are building up a graphical tool wrapping the underlying command-line execution of the described tools. By now, given that the tools are available on a Linux machine, a user may start the different programs via pipelining by hand. For example, if we have a our BIOCHAM description in $\mathcal{C}$ given in a file named `biocham.alc`, the domain specification given in a file named `biocham.stat` and want to display the chart as given in Figure 4 using gnuplot, you invoke on your local system the following commands:

```
$UNIX> al2asp -l c biocham.alc | \
       cat - biocham.stat | \
       gringo -c n=5 | clasp 0 | \
       as2sth --csv | \
       asplot present(a) present(b) \
       && gnuplot plot.plt
```

The second possibility is more user-friendly. To this end, we built up a web-based interface at (BioASP Tools), where the described tools are fully encapsulated as a server application. In this way, one can use the whole reasoning system without installing local applications. The mentioned Firefox-Plugin to describe $\mathcal{C}_{TAID}$ problems in a graphical way is able to access the web interface directly by sending the underlying action description to the web server. We added several examples on our web interface, where one can see how descriptions and queries to the system look like and how a user is able to access the different backends.

Benchmarks

In this section, our core tools (`al2asp`, `gringo` and `clasp`) will be empirically compared to the systems $CCalc$ (Giunchiglia et al. 2004) and dlv^k (Eiter et al. 2003a) since all of them use input languages based on $\mathcal{C}$. Unfortunately, the system $CPlan$ (Castellini, Giunchiglia, and Tacchella 2003) is no longer maintained and the authors provided a windows executable only which was not usable in our benchmark setting.

The benchmarks were carried out on an Intel Core2Duo 6400 with 2.13GHz and 2 GB RAM running a 32-bit version of Ubuntu GNU/Linux. For our tests, we used `al2asp` v0.4, `gringo` v1.0.0 and `clasp` v1.0.5 with default settings. $CCalc$ was used in version 2.0 and among the provided SAT solvers $grasp$ was used. Although $grasp$ does not provide the current state of the art SAT solving techniques, it was the only solver in our tests that produced all solutions. Regarding dlv^k, we used release 2007-10-11 with default settings.

Concerning pure planning problems, one is often interested in finding only the first solution. This issue is different in our approach, in most of the biological applications there is a need to consider all solutions. For example, recall Figure 4 where we need to process all answer sets in order to do statistical analysis. Biological queries to the system often lead to a large number of answers that need to be processed by biologists afterwards. Due to biologist's additional knowlegde, some of the answers might make no sense in the real biological background and sometimes they want to figure out subsets satisfying certain constraints they did not know before. To this end, we consider both cases when comparing the different systems, finding one, and finding all solutions.

Unfortunately, our current biological applications get solved too fast to make systems compareable. Being not

were not given explicitly in the action description.

No.	Instance	length	bioplan	$CCalc$	dlv^k
1	l1nc	10	0.10	0.14	17.81
2	l2nc	15	0.20	0.19	—
3	l3nc	20	2.12	0.26	—
4	l4nc	25	—	0.39	—
5	l1c	1	8.63	—	2.43
6	l2c	1	17.39	—	5.24
7	l3c	1	26.41	—	8.09
8	l4c	1	35.43	—	10.56
Average Time (Sum Timeouts)			12.90 (3)	0.24 (12)	8.82 (9)
Average Penalized Time			86.29	300.12	230.51

Table 2: Lights out experiments computing one solution

generic[10], a comparison of different systems using our biological problems is not yet feasible. We use crafted artificial problems instead to compare performance of systems.

The first problem is the well known *blocks world* which consists of a table and several blocks. Given an initial state of piled up blocks, the planning system's task is to find out how to rearrange the blocks such that they are piled up in a predefined order. We used the dlv^kencoding and problem instances from (Eiter et al. 2003b). Due to advances in computer hardware, these old instances are solved too fast to get reasonable runtimes. That is why we came up with five additional instances (p6 - p10, see Table 1) which are still demanding for the systems running on today's hardware.

Our second benchmark suite *lights out*[11] is very similar to the *bomb in the toilet* problem: All of a variable number of light bulbs has to be switched off. In every state, every light can either be switched on or off. The problem comes in two flavors, either with concurrent execution of actions allowed or with concurrency disabled. The optimal[12] plan length in the latter case is equal to the number of light bulbs. It is easy to see that this problem leads to $n!$ many optimal plans regarding n bulbs that only differ in the sequence of switching off bulbs. Due to this behavior, we omit computing all solutions as in the blocks world setting.

The results of the *blocks world* benchmarks are listed in Table 1 and the *lights out* results are in Table 2. For every problem instance, we measured the time in seconds of three separate solving processes and computed the average which is shown in each systems column. A dash indicates that a system was unable to compute a solution in less than 600 seconds. The column labeled *length* denotes the length of the shortest possible plan(s) for the problem instance which is passed as a parameter to the different systems. The last row in the tables lists penalized average times. In contrast to normal average times, the penalized ones take timeouts into account. Although the system might have taken much longer to find a solution, the penalized average is computed as if the system found a solution after 600 seconds.

[10]Unlike most artificial problems, we do not have parameters controlling the size of problem instances.

[11]Idea taken from General Gameplaying Competition 2008.

[12]Optimal means that there is at least one solution at bound t, but no solution can at bound $t-1$.

Results show that compared to the other systems our system performs quite well and appears to be robust. In the *blocks world* example, it was the only system that could enumerate all solutions in reasonable time. As mentioned, this issue is especially valuable because our biological applications often need all solutions to be computed. But also when only one solution has to be found, our system outperformed both $CCalc$ and dlv^k. $CCalc$'s performance was comparable to ours until the problems became too hard in benchmark number 9.

Regarding the *lights out* problem, $CCalc$ performs surprisingly well when concurrent execution of actions is not allowed. It computes a solution almost instantly, while dlv^k has difficulties even in the smallest instance. Although being quite fast with a few light bulbs, the runtime of our system rises rapidly as soon as more than twenty bulbs are involved. When allowing concurrency in this example, dlv^k is the fastest system. $CCalc$ seems to have great problems with the huge[13] number of light bulbs which was used in the problem instances and is unable to find a solution in any instance. Our system performs quite well in this benchmark, though not as well as dlv^k. In general, the benchmarks show that our system is more than competitive compared to other planning systems.

Discussion

Although we motivate (and apply) our approach in a biological setting, many features are readily applicable to representing and reasoning about dynamical systems in general. Centering our approach on $\mathcal{C}$ has several benefits. First, $\mathcal{C}$ is a rich and well-studied formalism. Second, it constitutes a mainstream implementation line for action languages. To this end, we provided a translation of the biologically motivated action language $\mathcal{C}_{TAID}$ to $\mathcal{C}$ and devise several tools for dealing with action descriptions in $\mathcal{C}$ (and $\mathcal{C}_{TAID}$). Among them, we implemented the compiler `al2asp` allowing for translating action descriptions in $\mathcal{C}$ (and $\mathcal{C}_{TAID}$) to logic programs under answer sets semantics. This approach is similar to the one taken by dlv^k for processing action language $\mathcal{K}$. Both approaches exploit the grounding and solving capacities of ASP, offering uniform (and thus instance independent) problem encodings and easy variable handling. Our approach is supported by a variety of pragmatic yet indispensable tools for addressing real world applications. Compared to other planning systems, we are able to compete with, and sometimes even outperform current systems. Finally, our tools (as well as their source code) and the benchmark problems are freely available at (BioASP Tools).

References

Baral, C.; Chancellor, K.; Tran, N.; Tran, N.; Joy, A.; and Berens, M. 2004. A knowledge based approach for representing and reasoning about signaling networks. In *Proceedings of the Twelfth International Conference on Intelli-*

[13]25.000 bulbs in instance *l1c* up until 100.000 bulbs in instance *l4c*.

No.	Instance	length	bioplan - one	$CCalc$ - one	dlv^k - one	bioplan - all	$CCalc$ - all	dlv^k - all
1	p01	05	0.31	0.51	0.06	0.32	0.51	0.06
2	p02	06	0.22	0.35	0.05	0.22	0.42	0.05
3	p03	08	1.19	1.21	1.21	1.21	8.23	4.57
4	p04	09	3.89	3.88	1.17	4.05	5.74	15.19
5	p05	11	5.04	5.39	2.92	4.92	11.24	22.98
6	p06	13	4.21	3.88	21.15	4.78	408.23	—
7	p07	14	8.76	7.08	42.04	11.81	364.22	—
8	p08	16	36.88	14.28	—	133.49	—	—
9	p09	16	39.39	129.15	—	41.75	—	—
10	p10	17	66.68	—	—	85.83	—	—
Average Time (Sum Timeouts)			17.49 (0)	19.47 (3)	10.54 (9)	20.61 (0)	122.87 (9)	9.58 (15)
Average Penalized Time			17.49	77.52	187.38	20.61	266.01	304.79

Table 1: Blocks world experiments computing one and all solutions

gent Systems for Molecular Biology/Third European Conference on Computational Biology (ISMB'04/ECCB'04), 15–22.

Baral, C.; Brewka, G.; and Schlipf, J., eds. 2007. *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

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

BioASP Tools. `http://www.cs.uni-potsdam.de/wv/bioasp`.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2003. Sat-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence* 147(1-2):85–117.

Chabrier-Rivier, N.; Chiaverini, M.; Danos, V.; Fages, F.; and Schächter, V. 2004. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.* 325(1):25–44.

Dworschak, S.; Grell, S.; Nikiforova, V.; Schaub, T.; and Selbig, J. 2007. Modeling biological networks by action languages via answer set programming. *Constraints Journal*. To appear.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003a. A logic programming approach to knowledge-state planning. *Artificial Intelligence* 144(1-2):157–211.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2003b. A logic programming approach to knowledge-state planning, ii: The dlv$^{\mathrm{k}}$ system. *Artificial Intelligence* 144(1-2):157–211.

Fages, F.; Sollman, S.; and Chabrier-Rivier, N. 2004. Modelling and querying interaction networks in the biochemical abstract machine biocham. *Journal of Biological Physics and Chemistry* 4:64–73.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007a. clasp: A conflict-driven answer set solver. In Baral et al. (2007), 260–265.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007b. Conflict-driven answer set solving. In Veloso, M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 386–392. AAAI Press/The MIT Press. Available at http://www.ijcai.org/papers07/contents.php.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A new grounder for answer set programming. In Baral et al. (2007), 266–271.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Transactions on Artificial Intelligence* 3(6):193–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 623–630.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1-2):49–104.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In Gelfond, M.; Leone, N.; and Pfeifer, G., eds., *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, 92–106. Springer-Verlag.

Tran, N., and Baral, C. 2004. Reasoning about triggered actions in AnsProlog and its application to molecular interactions in cells. In Dubois, D.; Welty, C.; and Williams, M., eds., *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, 554–564. AAAI Press.

Tran, N. 2006. *Reasoning and hypothesing about signaling networks*. Ph.D. Dissertation, Arizona State University.

Special Session on Argument, Dialogue and Decision

Since the work of John Pollock, Ronald Loui and others in the eighties, argumentation has proven to be successful in nonmonotonic logic. In the early nineties Dung and others showed that argumentation is also very suitable as a general framework for relating different nonmonotonic logics. Finally, in recent years argument-based logics have been used to facilitate reasoning and communication in multi-agent systems.

Argumentation can be studied on its own, but it also has interesting relations with other topics, such as dialogue and decision. For instance, argumentation is an essential component of such phenomena as fact finding investigations, computer supported collaborative work, negotiation, legal procedures, and online dispute mediation. However, only recently have researchers begun to explore the use of argumentation in these contexts.

Session Chairs

Yannis Dimopoulos, University of Cyprus, Cyprus
Gerard Vreeswijk, Universiteit Utrecht, The Netherlands

Program Committee

Leila Amgoud, Université Paul Sabatier, France
Jamal Bentahar, Concordia University, Canada
Gerhard Brewka, Universität Leipzig, Germany
Paul Dunne, University of Liverpool, UK
Michael Maher, National ICT, Australia
Pavlos Moraitis, Université Paris Descartes, France
Tim Norman, University of Aberdeen, UK
Simon Parsons, Brooklyn College CUNY, USA
Henry Prakken, Utrecht University and University of Groningen, The Netherlands
Iyad Rahwan, British University in Dubai, UAE; University of Edinburgh, UK
Guillermo Simari, Universidad Nacional del Sur, Argentina
Francesca Toni, Imperial College London, UK

Towards Enforcement of Confidentiality in Agent Interactions

Joachim Biskup and **Gabriele Kern-Isberner** and **Matthias Thimm**
Faculty of Computer Science
Technische Universität Dortmund, Germany

Abstract

A lot of work has been done on interactions and negotiations in multi agent systems. In this paper, we introduce new aspects of security into this scenario by dealing with confidentiality preservation. As we assume the agents to be capable of reasoning, we also have to take inferences of conveyed information into account. This problem has been addressed by *controlled query evaluation (CQE)* in the security community. We present a conceptual integration of CQE techniques into a BDI inspired agent model that allows agents to explicitly handle confidentiality preserving concerns when interacting with other agents in a multi agent system. We illustrate our ideas on an example of distributed meeting scheduling.

Introduction

Negotiation between agents involves exchange of information and persuasion in order to reach an agreement. In this paper we are concerned with security aspects and especially with confidentiality preservation aspects of negotiations between agents in multi agent systems which provide rich application scenarios for problems dealing with confidentiality and availability of information. There is much work on formalizing negotiation scenarios using multi agent systems, see for example (Kraus 2001; Karunatillake *et al.* 2005; Rahwan, Sonenberg, & Dignum 2003; Booth 2002). But very little work has been done in security aspects for multi agent systems, most of this work handling secure communication and authentication problems, e g. (Winslett 2003; Poslad, Charlton, & Calisti 2003; Sierra *et al.* 2003; Boulosa *et al.* 2006). We are addressing the problem of confidentiality preservation under the inference problem (Farkas & Jajodia 2002) in the sense of (Biskup & Bonatti 2004; Biskup & Weibert 2007a; 2007b) for multi agent systems. When agents interact with other agents in a multi agent system, pieces of information can be exchanged and by using inference techniques, more information can be obtained. An agent might not be fully aware of the conclusions a second agent can infer from the given information, and thus the second agent may be able to derive information he is not allowed to know.

Controlled query evaluation (Biskup & Bonatti 2004; Biskup & Weibert 2007a; 2007b) is a formal approach to determine the conclusions an agent can infer if provided with a specific information and to check whether the disclosure of said information violates confidentiality. We will present a declarative concept of confidentiality preserving negotiations in multi agent scenarios that is based on quite a general type of negotiation acts. Apart from defining which agent conveys to which other agent which piece of information, we also include interactions concerning justifications that often prove essential for reaching a negotiation goal. We link our ideas to standard agent models by extending the BDI model (Weiss 1999) by components for controlled query evaluation, thus taking first steps towards an operational framework. Our approach is illustrated by the problem of distributed meeting scheduling (Garrido, Brena, & Sycara 1996).

This paper is structured as follows: First, we give a brief overview on controlled query evaluation. Then we identify the problems of confidentiality preservation in agent interactions and introduce our running example. We continue with a formal description of our framework and sketch an agent model that summarizes the formal functionalities in an abstract manner afterwards. We conclude with comparisons to other works and an outlook on future work.

Controlled Query Evaluation

Controlled Query Evaluation (CQE) is an approach for preservation of confidential information in interactions between an information system and its users (Sichermann, de Jonge, & van de Riet 1983; Bonatti, Kraus, & Subrahmanian 1995). Each user of the system might have some restrictions on the information he is allowed to obtain from the system. Ordinary database systems restrict permissions using static access rights, but thus suffer from the inference problem (Farkas & Jajodia 2002). A malicious user can outsmart such a system by exploiting the inference problem, given he has some knowledge about the structure of the data. This is illustrated in the following example, which is taken from (Biskup *et al.* 2007).

Example 1. Suppose a database stores information about employees and their salary but must not disseminate information about the specific salary of a specific employee. If

using static access rights the two pieces of information "Alice is a manager" and "a manager's salary is $ 50,00" might appear harmless; but if one combines them they imply the information "Alice's salary is $ 50,000" which should be kept secret.

CQE is a dynamic approach to overcome the inference problem by dynamically checking the user's knowledge to ensure that he can not derive information he is not allowed to know. By distorting answers to queries by either lying or refusing to answer, it has been shown that confidential information can be kept secret in many different scenarios (Biskup & Bonatti 2004; Biskup & Weibert 2007b). What follows is a brief introduction to CQE terminology that will suffice for our needs.

Confidentiality Policies

Let $\mathcal{L}$ be a propositional language. Although $\mathcal{L}$ is propositional we will also use predicates and rule schemata and assume that $\mathcal{L}$ is properly grounded. When considering subsets of $\mathcal{L}$ we assume these subsets to be consistent when not mentioned otherwise. For a set S we denote with $\mathfrak{P}(S)$ the power set of S.

Let $\phi \subseteq \mathcal{L}$ be a finite set of sentences and $\alpha \in \mathcal{L}$ a sentence. The *evaluation* of α in ϕ is either `true` (iff α can be inferred: $\phi \vdash \alpha$), `false` (iff $\neg\alpha$ can be inferred: $\phi \vdash \neg\alpha$) or `unknown` (else). We consider queries addressing the information system ϕ, that consist only of a sentence $\alpha \in \mathcal{L}$ and awaits an answer of the form `true`, `false` or `unknown` with the obvious meaning.

At all times the system keeps a log of the user's knowledge, denoted `log`, that consists of the assumed a priori knowledge of the user and is updated with every answer the system gives to that particular user. The actual controlled query evaluation consists of two steps: first a *censor* checks whether the actual evaluation of a query, or a possible conclusion of it, together with the user's assumed knowledge violate confidentiality, and then, if necessary, a *modificator* distorts the answer somehow so that the distorted answer can be given to the user without violating confidentiality.

To represent a confidentiality policy we use the notion of *confidentiality targets*. A confidentiality target consists of the sentence to be protected and a set of truth values the user should not infer.

Definition 1 (Confidentiality target, confidentiality policy)**.** A *confidentiality target* is a pair $\langle \psi, V \rangle$ with $\psi \in \mathcal{L}$ and $V \subset \{\texttt{true}, \texttt{false}, \texttt{unknown}\}$ and $\emptyset \neq V \neq \{\texttt{true}, \texttt{false}, \texttt{unknown}\}$. A *confidentiality policy* is a finite set of confidentiality targets.

Example 2. The confidentiality target $\langle a, \{\texttt{true}, \texttt{false}\}\rangle$ defines that the user is not allowed to infer that a is either `true` or `false` (whether this coincides with the information system's actual evaluation of a or not).

The specification of CQE does not make sense, when the user already knows a confidential piece of information in his a priori knowledge $\texttt{log}_0$. So it is reasonable to assume that the user's a priori knowledge does not already violate confidentiality in the first place. This condition and all other possible restrictions on the use of the system are formalized as a precondition $precond$. In general, for a finite set of sentences $\phi \subseteq \mathcal{L}$, the user's assumed a priori knowledge $\texttt{log}_0$ and a confidentiality policy $policy$, the tuple $(\phi, \texttt{log}_0, policy)$ has to *satisfy* a given precondition $precond$ in order to use the system.

Preserving confidentiality

The approach of CQE is described via a CQE-function that basically maps a sequence of queries to a sequence of answers. As a side effect, the function updates the maintained user log appropriately with the newly acquired knowledge of the user. The formal definition of *confidentiality preserving* is expressed in terms of an indistinguishability property, roughly saying that for all reasonable situations, including all possible sequences of queries, the user cannot distinguish the actual information system instance from an alternative instance in which the evaluation of a target-sentence ψ is not in the corresponding target-set V.

A CQE-function can preserve confidentiality by either lying, i. e., providing a false answer to a query, or refusing to answer at all, i. e., returning a special answer value `refuse`. Nonetheless, such a CQE-function should also provide a maximal availability towards the user, i. e., it should distort as few answers as possible in order to preserve confidentiality.

Example 3. We continue Example 1. Suppose the user has the a priori knowledge $\texttt{log}_0 = \{m(X) \wedge mSalary(Y) \Rightarrow salary(X, Y)\}$ which says that "If X is a manager and a manager's salary is Y, then X's salary is Y". Furthermore the information system ϕ is given by $\phi = \{m(alice), mSalary(50000)\}$. The information about Alice's salary must be preserved in ϕ as given by the confidentiality policy $\texttt{conf} = \{\langle salary(alice, 50000), \{\texttt{true}\}\rangle\}$. Suppose the user asks ϕ for the evaluation of $m(alice)$. The information system can answer this query truthfully with `true` as confidentiality is still preserved when this information is disclosed. The CQE method then adds $m(alice)$ to the user's knowledge yielding $\texttt{log}_1 = \texttt{log}_0 \cup \{m(alice)\}$. When the user now asks ϕ about the evaluation of $mSalary(50000)$, the information system must not answer `true` because then the user can infer the confidential piece of information $salary(alice, 50000)$. Therefore ϕ must lie by answering either `false` or `unknown` or refuse to answer.

Operational frameworks for CQE have been investigated in many different scenarios, see e. g. (Biskup & Weibert 2007b).

Confidentiality in Multi Agent Systems

Although CQE is developed mostly for interactions between an information system and its users, preservation of confidentiality is needed in many different situations. Furthermore in the standard CQE scenarios confidentiality is only the concern of the information system. When considering multi agent systems, and especially multi agent systems performing negotiation, preservation of confidentiality is a bilateral concern of all agents. Although the agents are willing

to participate in a negotiation, they also might have secrets they do not want to disclose to other agents participating in that negotiation (Winslett 2003).

To illustrate our ideas of a confidentiality preserving multi agent system, we use the problem of meeting scheduling as an example (Garrido, Brena, & Sycara 1996). Although this problem has already been studied under privacy issues in, for example, (Wallace & Freuder 2002), we pursue a more sophisticated approach. While in those papers the agents attempt to preserve privacy and thus confidentiality by only disseminating as little information as possible to achieve a successful negotiation, we aim at implementing well-known methods for CQE directly into the agent.

The problem of meeting scheduling as it fits to our purposes is described as follows.[1] We consider a set of n different agents, each of them has its own calendar of the week, consisting of six days from Monday to Saturday, with eight time slots of one hour each day. Every agent may already have some time slots filled with appointments and some agents may share the same appointments. The "negotiation goal" of the n agents is to determine a specific time slot for a new appointment, such that every agent can attend to that appointment. To reach this goal the agents can exchange information by querying other agents about their calendars and other beliefs, give proposals for the new appointment, and agree as well as reject given proposals. Agents may change their beliefs over time and abandon other appointments in order to reach an agreement. As negotiation involves persuasion and argumentation (Parsons, Sierra, & Jennings 1998; Kraus, Nirkhe, & Sycara 1993; Karunatillake *et al.* 2005) we also enable the agents to ask for justifications for other agents' beliefs.

The confidentiality issues of an agent in this scenario can be of different kinds. Suppose the agents are in an employer/employee relationship, then the employee surely wants to hide information about spare time activities or the attendance to a strike commission. Furthermore the employer may also want to hide information about spare time activities but also about the existence of a job interview for a possible replacement for the said employee. In general we consider the following four different privacy issues as relevant for the problem of meeting scheduling: 1.) the date/time of a specific appointment, 2.) whether a specific agents attends a specific appointment or not, 3.) the existence of an appointment at a specific date/time and 4.) whether an agent is busy at a specific date/time or not.

The Declarative Concept

In this section we develop a declarative view of a multi agent system with negotiating and confidentiality preserving agents. Our approach is inspired by the work of Kraus in (Kraus 2001) but due to space restrictions we only give a short overview of our ideas. Therefore we do not provide a formal definition of the semantics, but give some examples to illustrate the integration of confidentiality preservation into an agent model.

Let $\mathfrak{A}$ be a set of agent identifiers $\mathfrak{A} = \{A_1, \ldots, A_n\}$. The agents negotiate on a given *negotiation goal* NG which is a subset of $\mathcal{L}$. NG specifies the search space for possible solutions.

Example 4. Suppose $\mathcal{L}$ contains grounded predicates of the form $\texttt{daytime}(AP, D, TS, TE)$ where AP denotes an appointment, D is the day and TS resp. TE is the start resp. end time. Then the meaning of the instance $\texttt{daytime}(staff\_meeting, monday, 12, 13)$ is "The staff meeting takes place on Monday between 12 and 13". Suppose a group of agents wants to negotiate about the day and time for a project meeting with a duration of one hour. Then the corresponding negotiation goal $\mathsf{NG} \subseteq \mathcal{L}$ is

$$\begin{aligned}\mathsf{NG} \;=\; \{&\texttt{daytime}(proj\_meeting, monday, 8, 9),\\ &\texttt{daytime}(proj\_meeting, monday, 9, 10), \ldots\}\end{aligned}$$

We abbreviate the above negotiation goal with $\texttt{daytime}(project\_meeting, X, Y, Y{+}1)$.

Given a negotiation goal NG the task of the agents is to determine a sentence $\mathsf{ng} \in \mathsf{NG}$ which can be mutually believed or adopted to be believed (if a revision of the agent's belief is necessary) by all agents.

We continue our development by defining the possible *negotiation acts*, i. e., the possible actions an agent can undertake in this system. As in the previous section we only consider simple yes/no/unknown-queries. But to enable the agents to use refusal as possible distortion method for modification of confidentiality violating information, let $\Theta = \{\texttt{true}, \texttt{false}, \texttt{unknown}, \texttt{refuse}\}$ be the set of possible answers to a query. To define the possible actions of an agent in our system, we define the set Ψ of negotiation (speech) acts as follows.

Definition 2 (Negotiation Acts)**.** The set of *negotiation acts* Ψ is the minimal set containing the following:

- For every $A \in \mathfrak{A}$, $\mathbf{B} \subseteq \mathfrak{A}$, $\alpha \in \mathcal{L}$ and $x \in \{\texttt{true}, \texttt{false}, \texttt{unknown}\}$ it is $\langle A : \mathsf{inform}\ \mathbf{B}\ \alpha\ x\rangle \in \Psi$ with the meaning: A tells $\mathbf{B}$ that his evaluation of α is x.
- For every $A, B \in \mathfrak{A}$, $\alpha \in \mathcal{L}$ it is $\langle A : \mathsf{query}\ B\ \alpha\rangle \in \Psi$ with the meaning: A asks B for his evaluation of α.
- For every $A, B \in \mathfrak{A}$, $\alpha \in \mathcal{L}$, $x \in \Theta$ it is $\langle A : \mathsf{answer}\ B\ \alpha\ x\rangle \in \Psi$ with the meaning: the answer of A to B regarding α is x.
- For every $A \in \mathfrak{A}$ it is $\langle A : \mathsf{abandon}\rangle \in \Psi$ with the meaning: A abandons the current negotiation.
- For every $A \in \mathfrak{A}$, $\alpha \in \mathcal{L}$ it is $\langle A : \mathsf{propose}\ \alpha\rangle \in \Psi$ with the meaning: A proposes α as a solution for the negotiation.
- For every $A, B \in \mathfrak{A}$, $\alpha \in \mathcal{L}$ and $x \in \{\texttt{true}, \texttt{false}\}$ it is $\langle A : \mathsf{justify}\ B\ \alpha\ x\rangle \in \Psi$ with the meaning: A asks B to justify that α has the evaluation x.
- For every $A, B \in \mathfrak{A}$, $\alpha \in \mathcal{L}, \Phi \subseteq \mathcal{L}$ and $x \in \{\texttt{true}, \texttt{false}\}$ it is $\langle A : \mathsf{justification}\ B\ \alpha\ x\ \Phi\rangle \in \Psi$ with the meaning: A justifies that α has the evaluation x towards B with Φ.

[1] We simplify the problem of meeting scheduling in comparison to (Wallace & Freuder 2002) by omitting the locations of the meetings.

- It is $\circ \in \Psi$ which denotes the "empty" action.

We restrain our presentation of these negotiation acts on the syntactic representation above and omit definitions of semantics.

Let Σ denote the set of all *negotiation sequences*, i. e., all ordered tuples $(\tau_1, \ldots, \tau_l)$ with $\tau_1, \ldots, \tau_l \in \Psi$, $l \in \mathbb{N}$ and $\diamond$ denoting the empty negotiation sequence. We use the element operator $\in$ with the usual meaning also on negotiation sequences, i. e., it is $\tau \in (\tau_1, \ldots, \tau_l)$ iff $\tau \in \{\tau_1, \ldots, \tau_l\}$.

Negotiating agents

We assume a suitable BDI architecture (Weiss 1999) as a basis for an agent. In this section we focus on modeling beliefs and confidentiality issues before proposing a complete suitable BDI-inspired agent model in the next section.

An agent's beliefs will comprise beliefs about himself and the current state of the world as well as his view of the beliefs of other agents (Kraus, Nirkhe, & Sycara 1993). Furthermore we explicitly represent an agent's confidentiality policy as a separate piece of the agent's belief. As information may be confidential differently with respect to the different agents, we extend the definition of confidentiality target appropriately.

Definition 3 (Personalized confidentiality target, personalized confidentiality policy)**.** A *personalized confidentiality target* is a triple $\langle B, \psi, V\rangle$ with $B \in \mathfrak{A}$, $\psi \in \mathcal{L}$ and $V \subset \{\texttt{true}, \texttt{false}, \texttt{unknown}\}$ and $\emptyset \neq V \neq \{\texttt{true}, \texttt{false}, \texttt{unknown}\}$. A *personalized confidentiality policy* is a finite set of personalized confidentiality targets.

The first component of a personalized confidentiality target is the subject for this target, i. e., the agent from whom the piece of information, that ψ has a truth-value in V, should be kept secret.

Example 5. Let e_1 and b_1 be agents. Suppose e_1 wants to hide from b_1 the information that he attends the appointment scm (*strike committee meeting*). This can be represented as the personalized confidentiality target

$$\langle b_1, \texttt{attends}(e_1, scm), \{\texttt{true}\}\rangle$$

being part of e_1's confidentiality policy.

Example 6. Let a_1 and b_1 be agents. Suppose agent a_1 wants agent b_1 to know nothing definite about the date and time of an appointment m_1. This can be represented as (for all possible X, Y, Z)

$$\langle b_1, \texttt{daytime}(m_1, X, Y, Z), \{\texttt{true}, \texttt{false}\}\rangle$$

being part of a_1's confidentiality policy. , which is an abbreviation for

$$\langle b_1, \texttt{daytime}(m_1, monday, 8, 9), \{\texttt{true}, \texttt{false}\}\rangle,$$
$$\langle b_1, \texttt{daytime}(m_1, monday, 9, 10), \{\texttt{true}, \texttt{false}\}\rangle$$
$$\ldots$$

Observe that the above personalized confidentiality target also prohibits the disclosure of information that might not be true in the current belief of an agent. Given adequate background constraints as "One appointment can not take place at two different times" one of the above (sub-)targets necessarily has to be false in the agent's belief. But the above target states that b_1 must know nothing definite about the day and time of m_1 and so he should also not believe a wrong date for it.

When agents gather new information about other agents by observing a negotiation act, they have to incorporate this new information into their individual belief and their beliefs about other agents, using belief operations as revision or update (Krümpelmann *et al.* 2008). Thus the beliefs of an agent have to be represented with respect to a given sequence σ of hitherto executed negotiation acts.

Furthermore, we improve our underlying framework of propositional logic in two ways.

First, as agents may have different beliefs about the world and especially about the beliefs of other agents, the propositional language $\mathcal{L}$ is not expressive enough to capture these needs. We therefore extend the propositional language $\mathcal{L}$ by introducing a family of modal operators B_X that read "Agent X believes…" as in epistemic logic with a standard Kripke-style semantics, yielding an extended language $\mathcal{L}^B$ with $\mathcal{L} \subseteq \mathcal{L}^B$. We assume the standard properties for B_X and refer to (Fagin *et al.* 2003) for a full axiomatization. With the use of these modal operators, we can represent agents' beliefs about other agents and the agents can reason about other agents' beliefs. Therefore, let $\models^B$ denote an appropriate inference relation for $\mathcal{L}^B$.

Definition 4 (Beliefs)**.** The *beliefs* $\texttt{bel}_A^\sigma$ of an agent A after the negotiation sequence σ is a tuple

$$\texttt{bel}_A^\sigma = (\texttt{is}_A^\sigma, \texttt{conf}_A^\sigma, \{\texttt{view}_{A,B_1}^\sigma, \ldots, \texttt{view}_{A,B_m}^\sigma\})$$

with individual belief $\texttt{is}_A^\sigma \subseteq \mathcal{L}^B$, a personalized confidentiality policy $\texttt{conf}_A^\sigma$, and views $\texttt{view}_{A,B_1}^\sigma, \ldots, \texttt{view}_{A,B_m}^\sigma \subseteq \mathcal{L}^B$ about agents $B_1, \ldots, B_m$. The a priori belief of the agent is denoted $\texttt{bel}_A^\diamond$.

Second, given the current beliefs $\texttt{bel}_A^\sigma$ and observing a negotiation act τ, or even participating in it, an agent A needs to process the new information in order to derive the new beliefs $\texttt{bel}_A^{\sigma+\tau}$, where $+$ denotes concatenation. Such derivations might just apply propositional or modal logic inferences, or suitable combinations of these with more sophisticated and generally non-monotonic techniques like belief operations, which are not the topic of this discussion, see e. g. (Krümpelmann *et al.* 2008; Kern-Isberner 2001) for more information.

As a simple example of deriving a new belief, let agent A perform the negotiation act $\langle A : \textsf{inform}\ \{C\}\ \alpha\ \{\texttt{true}\}\rangle$ after the negotiation sequence σ resulting in a negotiation sequence σ'. If C now derives by means of some appropriate belief operations, that A truly believes in α to be true, then this negotiation act results in $B_A\alpha \in \texttt{view}_{C,A}^{\sigma'}$.

The above definition also offers the option to change the personalized confidentiality policy $\texttt{conf}_A^\sigma$ during a negotiation process. In fact, to guarantee a successful negotiation it might be necessary for the agent to abandon some of his personalized confidentiality targets in order to reach an agreement as the abandonment of beliefs in general is a crucial

issue in non-trivial negotiations (Zhang *et al.* 2004). Nevertheless the change of confidentiality policies is an open research problem for ordinary CQE as well, so assume that $\mathtt{conf}^{\sigma}_{A} = \mathtt{conf}^{\diamond}_{A}$ for all σ and A.

To initiate a negotiation between several agents, the agents have to accept a previously determined negotiation goal as a precondition. Although the process of determining an acceptable negotiation goal can itself be seen as a negotiation we do not formalize this process but require the agents to be willing to participate in a negotiation for a given negotiation goal.

Definition 5 (Acceptance function)**.** An *acceptance function* accept_A for an agent A is a function $\mathsf{accept}_A : \mathfrak{P}(\mathcal{L}) \to \{\mathtt{true}, \mathtt{false}\}$.

An agent A accepts a subset NG of $\mathcal{L}$ as a negotiation goal, if $\mathsf{accept}_A(\mathsf{NG}) = \mathtt{true}$. We call a set of agents $\mathfrak{A}$ *unwilling* if there is no negotiation goal NG such that for all $A \in \mathfrak{A}$ it holds $\mathsf{accept}_A(NG) = \mathtt{true}$. We only consider sets of agents $\mathfrak{A}$ that are willing to participate in a negotiation.

Once the negotiation goal is determined, the agents start exchanging information using the possible negotiation acts in Ψ. We introduce a simple function that determines the best next action in the current situation (possibly the empty action which is denoted by $\circ$).

Definition 6 (Action function)**.** An *action function* action_A for agent A is a function $\mathsf{action}_A : \Sigma \to \Psi$.

After incorporating new information into their beliefs, the agents' attitudes towards proposals of other agents may change. A negotiation ends when all agents agree to a given proposal ng in the solution space of the given negotiation goal NG.

Definition 7 (Agreement function)**.** An *agreement function* agree_A for an agent A is a function $\mathsf{agree}_A : \Sigma \times \mathcal{L} \to \{\mathtt{true}, \mathtt{false}\}$.

An agent A *agrees* to a proposal $\mathsf{ng} \in \mathsf{NG} \subseteq \mathcal{L}$ after exchanging some information during a sequence σ, if it holds $\mathsf{agree}_A(\sigma, \mathsf{ng}) = \mathtt{true}$. If $\mathsf{agree}_A(\sigma, \mathsf{ng}) = \mathtt{false}$, A *rejects* the proposal.

With the use of the above functions we can describe the final product of a negotiation in our approach. We call a negotiation sequence σ *semi-complete*, if it is intuitively well-formed. That means for example, that every query and every call for justification is answered, that there are no answers without a query, and so on.

Definition 8 (Negotiation Product)**.** A sentence ng is a *negotiation product* with respect to $\mathfrak{A}$, iff there exists a negotiation goal NG, such that

1. $\mathsf{ng} \in \mathsf{NG}$,
2. for all $A \in \mathfrak{A}$ it holds $\mathsf{accept}_A(NG) = \mathtt{true}$ and
3. there exists a semi-complete negotiation sequence σ for $\mathfrak{A}$ and NG, such that
   (a) $\langle A : \mathsf{propose\ ng} \rangle \in \sigma$ for some $A \in \mathfrak{A}$ and
   (b) for all $A \in \mathfrak{A}$ it holds $\mathsf{agree}_A(\sigma, \mathsf{ng}) = \mathtt{true}$.

We say that a negotiation sequence σ *failed*, if $\langle A : \mathsf{abandon} \rangle \in \sigma$ for a $A \in \mathfrak{A}$.

Preserving confidentiality in agent interactions

The confidentiality features are given on a declarative and an operational layer.

On a declarative layer, a formal definition for "confidentiality preserving" basically requires the following: If an agent is not allowed to learn some piece of information, then the agent's particular view on the behaviour of the overall system, applying for all possible initializations and all possible action sequences of the system, should never leave the agent with a belief that the said piece of information holds.

On an operational layer a control component censors each planned individual action for potential harmful consequences and possibly modifies the plan appropriately. We give some ideas on the operationalizing of the following declarative concept in the next section.

Clearly, the basic challenges are to design a policy control, i. e., censors and modificators necessarily operating action-wise, such that these mechanisms provably achieve the confidentiality goal declaratively expressed referring to all possible initializations and action sequences. These challenges are well-known to be highly demanding, see, e. g., the rich work on noninterference (Goquen & Mesequer 1982; Mantel 2001) or on cryptographic protocols (Goldreich 2004; 2001). Accordingly, in this paper we can only sketch our general approach to provide a tentative solution for negotiation in multi agent systems.

Given a suitable definition of "indistinguishability of situations", we propose the following (rough) generic outline for the declarative layer (as above let $precond$ expresses that confidentiality is not violated in the first place):

Definition 9 (Mutual confidentiality preservation)**.** The agents $A_1, \ldots, A_n$ *mutually preserve confidentiality* iff it holds

for all negotiation goals NG,
for all "actual situations", i. e.,
  for all initial a priori beliefs $\mathtt{bel}^{\diamond}_{A_1}, \ldots, \mathtt{bel}^{\diamond}_{A_n}$,
  for all negotiation sequences σ,
for all personalized confidentiality targets
$\langle A_j, \psi, V \rangle \in \mathtt{conf}^{\sigma}_{A_i}$ for some i, j

where $(\mathsf{NG}, \{\mathtt{bel}^{\diamond}_{A_1}, \ldots, \mathtt{bel}^{\diamond}_{A_n}\})$ satisfies $precond$

there exists an "alternative situation", i. e.,
  there exist alternative a priori beliefs $\Delta^{\diamond}_{A_1}, \ldots, \Delta^{\diamond}_{A_n}$
    such that $(\mathsf{NG}, \{\Delta^{\diamond}_{A_1}, \ldots, \Delta^{\diamond}_{A_n}\})$
    satisfies $precond$, and
  there exists an alternative negotiation sequence σ',

such that the following two conditions are met:

1. The actual situation as given above and the alternative situation as postulated are indistinguishable from the point of view of agent A_j.
2. From the point of view of agent A_j in the alternative situation, the evaluation of ψ is not in V.

Towards an Operational Framework

We now give some ideas on how to operationalize the declarative layer from Definition 9 on confidentiality preservation

in agent interactions, i. e., we propose a method that an agent can use to satisfy the above given security requirements for some exemplary cases.

Censoring and modification

Whenever an agent is about to execute an action (which is equivalent to the disclosure of information), he has to check whether confidentiality will be preserved after having executed said action. He does so by simulating the derivation methods that would (presumably) be applied by the other agents, when observing said action. Here we assume, that an agent has complete knowledge about the deriving methods of other agents and about their background knowledge. So the agent is capable of checking whether an action will violate confidentiality at any time. This check is accomplished by the censor of the agent which prevents the agent to disclose confidential information.

Definition 10 (Censor)**.** The *censor function* $\mathsf{violates}_A$ for an agent A is a function $\mathsf{violates}_A : \Sigma \times \Psi \rightarrow \{\mathtt{true}, \mathtt{false}\}$.

The censor evaluates the action under consideration regarding the agent's confidentiality policy. The definition of $\mathsf{violates}_A$ depends on the type of action. Due to lack of space we do not give a full definition of $\mathsf{violates}_A$ for all types of actions but only some examples for necessary conditions for $\mathsf{violates}_A$ to be true in order meet the declarative definition of confidentiality preservation above (Definition 9). For an action $\tau = \langle A : \mathsf{inform}\ \{B\}\ \alpha\ \mathtt{true}\rangle$ and a sequence σ it is $\mathsf{violates}_A(\sigma, \tau) = \mathtt{true}$ if

$$\begin{aligned} &\exists\, \langle B, \psi, \{\mathtt{true}\}\rangle \in \mathtt{conf}_A^\sigma : \mathtt{view}_{A,B}^\sigma \cup \{\alpha\} \models^B \psi \\ \vee\ &\exists\, \langle B, \psi, \{\mathtt{false}\}\rangle \in \mathtt{conf}_A^\sigma : \mathtt{view}_{A,B}^\sigma \cup \{\alpha\} \models^B \neg\psi \\ \vee\ &\ldots \end{aligned}$$

The definition of $\mathsf{violates}_A$ is the same as above for the action type answer. Interestingly, even a query can violate confidentiality.

Example 7. The question "Are you busy on Wednesday at 12?" provides several pieces of information about the questioner. First the respondent can infer, that the questioner does not know what the respondent does on Wednesday at 12[2] and second, that the questioner himself is assumably not busy on Wednesday at 12.

Thus for an action $\tau = \langle A : \mathsf{query}\ C\ \alpha\rangle$ and a sequence σ it is $\mathsf{violates}_A(\sigma, \tau) = \mathtt{true}$ if

$$\begin{aligned} &\exists\, \langle C, \psi, \{\mathtt{true}\}\rangle \in \mathtt{conf}_A^\sigma : \\ &\qquad \mathtt{view}_{A,C}^\sigma \cup \{\neg B_A\alpha, \neg B_A\neg\alpha\} \models^B \psi \\ \vee\ &\exists\, \langle C, \psi, \{\mathtt{false}\}\rangle \in \mathtt{conf}_A^\sigma : \\ &\qquad \mathtt{view}_{A,C}^\sigma \cup \{\neg B_A\alpha, \neg B_A\neg\alpha\} \models^B \neg\psi \\ \vee\ &\ldots \end{aligned}$$

Remember that B_A is the modal operator that reads "Agent A believes…".

[2]We assume that agents do only perform these queries if they do not know the answer.

As mentioned before, agents can either use lying, refusal or a combination of both to distort information, such that confidentiality is preserved. When defining the censor function $\mathsf{violates}_A$, one has to consider the actual distortion method to be used in order to actually preserve confidentiality. If the actual distortion method is lying, then the censor must not only prohibit, that any individual confidential piece of information is preserved, but the disjunction of all confidential pieces of information (Bonatti, Kraus, & Subrahmanian 1995).

Example 8. Suppose that $\mathtt{conf}_A^\sigma$ only consists of confidentiality targets regarding agent B with the evaluation "`true`" being the only confidential evaluation, i. e. $\mathtt{conf}_A^\sigma = \{\langle B, \psi_1, \{\mathtt{true}\}\rangle, \ldots, \langle B, \psi_l, \{\mathtt{true}\}\rangle\}$. If B already knows, that $\psi_1 \vee \ldots \vee \psi_l$ must be true, then the sequence of queries for ψ_1 to ψ_l results in an inconsistent view of the agent B's beliefs, because the query for every ψ_i must be answered with `false`.

Thus, for the confidentiality policy given in Example 8, an action $\tau = \langle A : \mathsf{inform}\ \{B\}\ \alpha\ \mathtt{true}\rangle$ and a sequence σ it is not sufficient to define $\mathsf{violates}_A(\sigma, \tau) = \mathtt{true}$ if

$$\exists\, i \in \{1, \ldots, l\} : \mathtt{view}_{A,B}^\sigma \cup \{\alpha\} \models^B \psi_i$$

but necessary to define $\mathsf{violates}_A(\sigma, \tau) = \mathtt{true}$ if

$$\mathtt{view}_{A,B}^\sigma \cup \{\alpha\} \models^B \psi_1 \vee \ldots \vee \psi_l$$

Furthermore, if the actual distortion method is refusal, then the censor must also take the possibility for meta inference into account.

Example 9. Let "The evaluation of α is `true`" be a confidential piece of information. Suppose Agent B believes, that α must be either `true` or `false`, and that agent B is fully aware of how agent A distorts answers to preserve confidentiality. Assume α is actually `false` for agent A and agent B asks A about the truth-value of α. Then A truthfully returns the answer "The evaluation of α is `false`" as it does not violate confidentiality. But suppose now, that α is actually `true` for agent A and B asks the same question. Now A must refuse to answer, in order to preserve confidentiality. But now B can infer, that α must be `true` for A, because if α would have been `false` for A, then A had not refused to answer. To overcome this problem A must refuse to answer the query about α in any case, so that B can not distinguish these two cases.

Suppose now that $\mathsf{violates}_A$ is properly defined and handles the above mentioned security problems accordingly. Then $\mathsf{violates}_A$ restrains the action function action_A of an agent A by ensuring the following constraint:

$$\textbf{IF}\ \mathsf{action}_A(\sigma) = \tau \quad \textbf{THEN} \quad \mathsf{violates}_A(\sigma, \tau) = \mathtt{false}$$

As agreeing and rejecting a proposal ng is equivalent to informing all agents about ng or $\neg$ng, the censor $\mathsf{violates}_A$ restrains the agreement function agree_A of an agent A by ensuring the following constraint:

$$\begin{aligned} &\textbf{IF}\ \mathsf{agree}_A(\sigma, \alpha) = x\ \textbf{THEN} \\ &\quad \mathsf{violates}_A(\sigma, \langle A : \mathsf{inform}\ \mathfrak{A}\ \alpha\ x\rangle) = \mathtt{false} \end{aligned}$$

If an action endangers confidentiality, the agent has to choose another action to execute. In the case of actions of the type inform, abandon, query, justify this can be realized by executing no action at all, as no other agent expects a particularly action from the first agent. But if one or more agents expect an action, either an answer to a query or a justification for a belief, the agent must produce an alternative answer that preserves confidentiality. The same is true for the agreement function of an agent, but as there are only two values possible, preservation can only be achieved by setting $\mathsf{agree}_A(\sigma, \alpha) = \mathtt{false}$ if $\mathsf{agree}_A(\sigma, \alpha) = \mathtt{true}$ violates confidentiality and vice versa[3]. In the case of answers to queries, the agent has the additional options to answer with `unknown` or to refuse to answer at all. Here standard CQE methods can be used to determine the best alternative answer (Biskup & Bonatti 2004; Biskup & Weibert 2007b).

Confidentiality preserving issues regarding disclosure of justifications have not been investigated in CQE so far. When the true justification for a belief violates confidentiality, many solutions to modify the answer are possible. The agent can make up a new justification, present another one that does not violate confidentiality or refuse to justify at all. But as we only want to formalize a general framework for enforcement of confidentiality between agents in this paper, we do not discuss the matter here and leave it open for future research. We conclude this section with an example that illustrates the above definitions.

Example 10. We continue our example of meeting scheduling. Suppose agent e_1 features the personalized confidentiality policy $\mathtt{conf}^{\sigma}_{e_1}$ after a negotiation sequence σ with

$$\mathtt{conf}^{\sigma}_{e_1} = \{\langle b_1, \mathtt{attends}(e_1, scm), \{\mathtt{true}\}\rangle\}$$

and does truly attend the strike committee meeting: $\mathtt{attends}(e_1, scm) \in \mathtt{is}^{\sigma}_{A}$. Furthermore e_1 has a pretty good clue that b_1 knows that there is a strike committee meeting being held on Wednesday at 12 to 13 and that if someone is busy at that time, he assumably attends said meeting. So e_1's view of b_1's beliefs includes

$$\begin{aligned}\mathtt{view}^{\sigma}_{e_1,b_1} \supseteq \{&\mathtt{daytime}(scm, wednesday, 12, 13),\\ &\mathtt{daytime}(scm, X, Y, Z) \wedge \mathtt{busy}(E, X, Y, Z)\\ &\Rightarrow \mathtt{attends}(E, scm))\}\end{aligned}$$

Let $\langle b_1 : \mathtt{query}\ e_1\ \mathtt{busy}(e_1, wednesday, 12, 13)\rangle$ be the last action of the sequence σ, i. e. b_1 asks e_1 whether he is busy on Wednesday at 12 to 13. Then e_1 must not answer this query truthfully, because we have $\mathsf{violates}_A(\sigma, \langle e_1 : \mathsf{answer}\ b_1\ \mathtt{busy}(e_1, wednesday, 12, 13)\ \mathtt{true}\rangle) = \mathtt{true}$ due to

$$\mathtt{view}^{\sigma}_{e_1,b_1} \cup \{\mathtt{busy}(e_1, wednesday, 12, 13)\} \models^{B} \mathtt{attends}(e_1, scm).$$

Therefore confidentiality is at risk and e_1 must alter his answer in order to preserve confidentiality. It is reasonable to assume that an agent knows whether he is busy at a given time or not. Especially e_1 must assume that b_1 thinks so. So it is for every X, Y, Z:

$$B_{e_1} busy(e_1, X, Y, Z) \vee B_{e_1} \neg busy(e_1, X, Y, Z) \in \mathtt{view}^{\sigma}_{e_1,b_1}$$

Due to this constraint e_1 can not undertake $\chi = \langle e_1 : \mathsf{answer}\ b_1\ \mathtt{busy}(e_1, wednesday, 12, 13)\ \mathtt{unknown}\rangle$ as next action, because this would result in the sentence

$$\neg B_{e_1} busy(e_1, X, Y, Z) \wedge \neg B_{e_1} \neg busy(e_1, X, Y, Z)$$

to be incorporated into $\mathtt{view}^{\sigma}_{e_1,b_1}$ and therefore leads to an inconsistency. So we have $\mathsf{violates}_A(\sigma, \chi) = \mathtt{true}$, because every confidential piece of information can be inferred from $\mathtt{view}^{\sigma}_{e_1,b_1}$ and the above piece of information. It follows that e_1 can only answer `false` or refuse to answer at all. Given that e_1 and b_1 are in an employee/employer relationship, refusal does not seem appropriate, so the answer `false` is the best choice for agent e_1.

Confidentiality preservation for BDI agents

The above developed framework summarizes the essential aspects of a negotiation in a formal but nonetheless mostly declarative manner. A fully featured model of multi agent negotiation needs among other things also to comprise decision making processes (Kraus 2001) and belief operations (Alchourrón, Gärdenfors, & Makinson 1985; Kern-Isberner 2001; Booth 2002; Krümpelmann *et al.* 2008). In this section we only give a brief overview about the model of a negotiating and confidentiality preserving agent in an abstract manner. Our agent model incorporates standard BDI architecture in order to represent a rational and autonomous agent (Rao & Georgeff 1995; Weiss 1999). BDI stands for Beliefs, Desires and Intentions and a BDI architecture separates the logical model of an agent into these three areas.

Figure 1 shows an abstract view of our negotiating and confidentiality preserving agent which incorporates both BDI as well as CQE techniques. However, in our model the beliefs component is explicitly divided into the beliefs of the agent about the world and himself (`is`), the beliefs about other agents (`view`) and a confidentiality policy (`conf`) as developed in our formal framework in the previous section. Furthermore the agent itself is divided into an active part (upper half) and a reactive part (lower half) which cooperate in a parallel mode; information flow is depicted with dashed lines, while action flow with solid lines. As in the BDI model developed in (Weiss 1999) the agents continuously evaluate the current state of the world, generate possible options for the next actions, and filter the best options using their beliefs, some underlying preferences (not depicted in the figure), their desires (`des`) and their intentions (`int`). Thereupon the best options are furthermore evaluated in the sense of confidentiality preservation by an agent's `policy control`. If an intention can be selected to be performed, the necessary actions are executed as depicted in Figure 1. Newly acquired information must be incorporated into the beliefs of the agent using revision and update techniques (Krümpelmann *et al.* 2008). As in the active part, also in the reactive part the preservation of confidentiality

[3] We disregard the case that both values violate confidentiality.

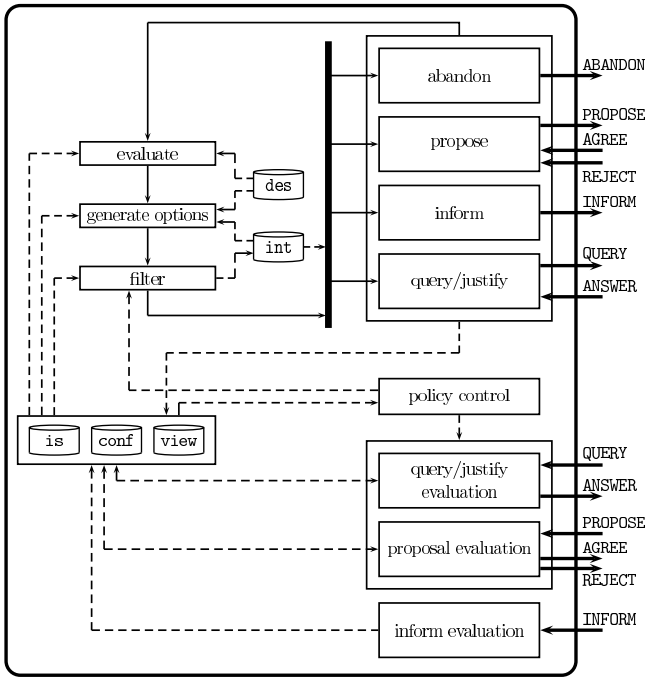


Figure 1: A negotiating agent

is handled by the component `policy control`, which prohibits the dissemination of confidential information. Whenever the agent needs to reply to a general query, a query for justification or a proposal, the `policy control` checks whether confidentiality is preserved and eventually alters the intended reply.

Related Work

Frameworks for distributed negotiation have been investigated very broadly so far, see (Rueda, Garcia, & Simari 2002; Kraus 2001; Karunatillake *et al.* 2005; Rahwan, Sonenberg, & Dignum 2003) for some examples. In this paper we do not intend to neglect this huge body of work but to add the new feature of confidentiality preservation. In fact it should be investigated if confidentiality preservation can be modularized and integrated into existing frameworks and implementations for negotiation.

The example of meeting scheduling as a negotiation problem has been elaborated before (Garrido, Brena, & Sycara 1996; Wallace & Freuder 2002). In (Wallace & Freuder 2002) also security issues are raised. However, in contrast to our approach, the preservation of confidentiality there is handled in a very simple manner as the agents only aim at disseminating as little information about themselves as necessary but do not consider confidentiality targets. The dissemination of as less information as possible can also be achieved in our approach with a suitable representation of the BDI core of the agent. When restraining the agent to act only passively in the environment so that he only reacts on queries, he does not disclose any other information. However, in a negotiation scenario this is not a desirable feature for all agents as then no negotiation will succeed as no action takes place; so a compromise between these two requirements can be made by adjusting the preferences of the agents accordingly. However, with the use of methods for CQE the agents do have a better formal representation of how to preserve their privacy. We therefore consider in our framework a lot more confidentiality problems as in (Wallace & Freuder 2002).

When talking about negotiation, another important aspect is argumentation. Argumentation theory has become a very active field of research and many proposals exist for introducing argumentative capabilities into negotiation systems (Amgoud, Dimopolous, & Moraitis 2007; Bench-Capon 2003; Rueda, Garcia, & Simari 2002; Karunatillake *et al.* 2005; Thimm & Kern-Isberner 2008). In our framework we provide a declarative support for handling argumentation in a multi agent system with the action types justify and justification. However, future research includes the evaluation of argumentation formalism for our framework and especially regarding our perspective of privacy and confidentiality issues.

Conclusion and future work

In this paper we presented a formal approach to adapt methods for CQE for the use in multi agent systems. CQE has some history in scientific research but has not been adapted for the use in multi agent systems so far. We are currently developing a full framework for handling negotiation, belief and confidentiality issues that bases on the approach proposed in this paper. Although the work presented here is just preliminary, the developed framework provides a solid basis for future work. As mentioned above this includes the exploration of argumentation formalism as well as an adaption of techniques for CQE for more sophisticated approaches of knowledge representation. More precisely, the fitness of CQE techniques for non-monotonic representation formalism has not yet been investigated. As default logics are common representation formalisms, an adaption of CQE techniques for these is mandatory. Furthermore, as persuasion (Bench-Capon 2003) is a fundamental aspect of negotiation, agents must have the ability to abandon specific confidentiality targets in order to reach agreements (Biskup *et al.* 2007). Also the adaption of other agents' confidentiality targets must be taken into account in order to ensure effective confidentiality handling.

Another main concern in CQE is the warranty of availability. This means that, although an agent must preserve confidentiality, he is also committed to provide as much useful information as possible. An agent can be equipped with an availability policy as well as a confidentiality policy. In the employer/employee example the employee might be committed to provide the employer with any information concerning a specific project, even if this violates confidentiality. The discrepancy between these two requirements has to be handled by the agent appropriately.

References

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

Amgoud, L.; Dimopolous, Y.; and Moraitis, P. 2007. A unified and general framework for argumentation-based negotiation. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems, AAMAS'2007*.

Bench-Capon, T. 2003. Persuasion in practical argument using value based argumentation frameworks. *Journal of Logic and Computation* 13(3):429–448.

Biskup, J., and Bonatti, P. 2004. Controlled query evaluation for enforcing confidentiality in complete information systems. *Int. Journal of Information Security* 3(1):14–27.

Biskup, J., and Weibert, T. 2007a. Confidentiality policies for controlled query evaluation. In *Proceedings of the 21th IFIP WG11.3 Working Conference on Data and Applications Security, LNCS 4602*, 1–13. Springer.

Biskup, J., and Weibert, T. 2007b. Keeping secrets in incomplete databases. *Int. Journal of Information Security* online first.

Biskup, J.; Burgard, D. M.; Weibert, T.; and Wiese, L. 2007. Inference control in logic databases as a constraint satisfaction problem. In *Proc. of the Third International Conference on Information Systems Security*, 128–142.

Bonatti, P. A.; Kraus, S.; and Subrahmanian, V. S. 1995. Foundations of secure deductive databases. *IEEE Transactions on Knowledge and Data Engineering* 7:406–422.

Booth, R. 2002. Social contraction and belief negotiation. In *Proc. of the 8th Conference on Principles of Knowledge Representation and Reasoning*, 375–384.

Boulosa, M.; Caib, Q.; Padgetc, J. A.; and Rushton, G. 2006. Using software agents to preserve individual health data confidentiality in micro-scale geographical analyses. *Journal of Biomedical Informatics* 39(2):160–170.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 2003. *Reasoning about Knowledge*. MIT Press.

Farkas, C., and Jajodia, S. 2002. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter* 4:6–11.

Garrido, L.; Brena, R.; and Sycara, K. 1996. Cognitive modeling and group adaptation in intelligent multi-agent meeting scheduling. In *First Iberoamerican Workshop on DAI and MAS*, 55–72.

Goldreich, O. 2001. *Foundations of Cryptography I – Basic Tools*. Cambridge University Press.

Goldreich, O. 2004. *Foundations of Cryptography II – Basic Applications*. Cambridge University Press.

Goquen, J. A., and Mesequer, J. 1982. Security policies and security models. In *Proc. IEEE Symposium on Security and Privacy*, 11–22.

Karunatillake, N. C.; Jennings, N. R.; Rahwan, I.; and Norman, T. J. 2005. Argument-based negotiation in a social context. In *Proc. of the 4th Int. joint conference on Autonomous agents and multiagent systems*, 1331–1332.

Kern-Isberner, G. 2001. *Conditionals in nonmonotonic reasoning and belief revision*. Number 2087 in Lecture Notes in Computer Science. Springer.

Kraus, S.; Nirkhe, M.; and Sycara, K. P. 1993. Reaching agreements through argumentation: a logical model and implementation. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, 233–247.

Kraus, S. 2001. Automated negotiation and decision making in multiagent environments. In *Selected Tutorial Papers from the 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School on Multi-Agent Systems and Applications*, 150–172.

Krümpelmann, P.; Thimm, M.; Ritterskamp, M.; and Kern-Isberner, G. 2008. Belief operations for motivated BDI agents. In *Proceedings of AAMAS'08*.

Mantel, H. 2001. Preserving information flow properties under refinement. In *Proc. IEEE Symposium on Security and Privacy*, 78–91.

Parsons, S.; Sierra, C.; and Jennings, N. 1998. Agents that reason and negotiate by arguing. *Journal of Logic and Computation* 8(3):261–292.

Poslad, S.; Charlton, P.; and Calisti, M. 2003. Specifying standard security mechanisms in multi-agent systems. In *Trust, Reputation, and Security: Theories and Practice*, volume 2631 of *LNCS*. Springer. 227–237.

Rahwan, I.; Sonenberg, L.; and Dignum, F. 2003. Towards interest-based negotiation. In *Proc. of the 2nd Int. Conf on Autonomous Agents and Multi-Agent Systems*, 773–780.

Rao, A. S., and Georgeff, M. P. 1995. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, 312–319.

Rueda, S. V.; Garcia, A.; and Simari, G. R. 2002. Argument-based negotiation among bdi agents. *Journal of Computer Science and Technology* 2(7):1–8.

Sichermann, G. L.; de Jonge, W.; and van de Riet, R. P. 1983. Answering queries without revealing secrets. *ACM Transactions on Database Systems* 8:41–59.

Sierra, J. M.; Hernández, J. C.; Ponce, E.; and Ribagorda, A. 2003. Protection of multiagent systems. In *Computational Science and its Applications*, 984–990.

Thimm, M., and Kern-Isberner, G. 2008. A distributed argumentation framework using defeasible logic programming. In *Proc. of the 2nd International Conference on Computational Models of Argument (COMMA'08)*.

Wallace, R., and Freuder, E. 2002. Constraint-based multiagent meeting scheduling: effects of agent heterogeneity on performance and privacy loss. In *Proc. Workshop on DCR*, 176–182.

Weiss, G., ed. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Winslett, M. 2003. An introduction to trust negotiation. In *iTrust 2003. Volume 2692 of Lecture Notes in Computer Science*. Springer-Verlag. 275–283.

Zhang, D.; Foo, N.; Meyer, T.; and Kwok, R. 2004. Negotiation as mutual belief revision. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 317–322.

Application of Possibilistic Stable Models to Decision Making

Jeremy Forth
Department of Computing
Imperial College London
London, UK
jforth@iweng.org

Abstract

This paper investigates the applicability of Possibilistic Stable Models (PSM) for Logic Programming to decision making through the use of several extensions to the PSM formalism that are chosen to enhance the realism in the resulting decision making model. The selected extensions to PSMs are unified through their incorporation into an argumentation framework designed to represent the combined formalism.

A PSM for a logic program is initially mapped to a Possibilistic preferred extension of the argumentation framework. This is first used as the basis for a particular type of strength comparison between arguments where the rule weights are interpreted as utilities. The framework is then extended to a value-based system incorporating context dependent utilities, and uncertainty in its domain knowledge. The strengths of arguments are assessed using an aggregation function combining all the significant magnitude information present in the argument. The concept of *value projection rules* are introduced that serve to provide domain context sensitivity for the mapping of circumstances onto values. Together with the aggregation function, this constitutes a method for realizing dynamic *decision framing*: the determination of the basis or *grounds* for the decision.

An example of some realism is identified from Psychology literature based on an ethical dilemma familiar to contemporary moral philosophers. This example is presented to the system using a standard axiomatization for knowledge representation. Through the choice of a very general aggregation function, the dilemma exemplifies how the defined system capabilities may act together to realize a meaningful decision function, which is later independently characterized.

Introduction

Preference handing frameworks based on priorities between rules have been defined previously in non-monotonic reasoning systems such as Default Logic e.g. (Reiter 1980; Brewka 1994; Delgrande & Schaub 2000) and also Answer Set Programming (ASP) and Extended Logic Programming e.g.(Gelfond & Lifschitz 1991; Zhang & Foo 1997; Prakken & Sartor 1997) amongst others. These formalisms have proven their value in capturing various representation problems in the study of commonsense reasoning, and more specifically in qualitative decision making. A related approach for representing weighting between rules are formalisms based on Possibility Theory (Zadeh 1978). Typically this represents the degrees of a modality, usually uncertainty or preference. In contrast with the qualitative preference representations mentioned above, Possibility Theory usually adopts a numerical weighting measure that when combined with a logical language yields a formalism with sufficient expressive power to represent decision problems. Possibilistic Logic (Dubois, Lang, & Prade 1994), an extension of classical logic, contains a measure representing possibility (consistency) or necessity (certainty) information occurring as part of a total ordering. The measure's degree is represented by weights attached to the formulas. In the necessity case, weights specifically represent the lower bounds of necessity measures.

In order to augment the representation and reasoning power of the logical language ASP to include numerical weighting, (Nicolas, Garcia, & Stéphan 2005) extended the notion of Stable Model Semantics (Gelfond & Lifschitz 1988) to Possibilistic Stable Models (PSM) for Logic Programming. Benefits of this approach include good computational characteristics and appropriate semantics for representing conditional rules.[1] Each stable model represents a coherent possible world view, and is expressed as a minimal set of atoms.)

The expressive power of PSMs (with some extensions) provides an opportunity to reason about more complex decision domains than are possible using qualitative preferences. In this paper we present and apply several such extensions to decision domains, and extract the resulting decision model.

To form the extensions in a disciplined way, we make use of an argumentation framework with the power to express all the extensions we consider. Argumentation frameworks normally find application in the construction of plausible reasoners. The definition of plausibility can vary, but the one used here is based on the selection of a set of mutually consistent assumptions (normally facts or rules). The abstract

[1] Stable Model Semantics interprets Normal Logic Programs (which possess default negation, and do not possess classical negation), including cases where the program's clauses are not stratifiable.

argumentation system of (Dung 1995) may be used to capture various forms of non-monotonic reasoning by using a layered abstraction design technique. We begin by defining PSMs and showing that the PSMs are captured by a particular instantiation of the abstract argument framework.

Using the argumentation framework's defined notion of *conflicting atoms* and an argument being *stronger* than another, the first decision mechanism is defined, given an illustrative example, and characterized in terms of the decision model realized. A PSM contains information not fully exploited in the first model, therefore the system is generalized through an assessment of arguments based on an aggregated measure of argument strength. Through a further enhancement to bipolar value based assessment, and context dependent projection of consequences the framework is shown to exhibit a powerful decision model, able to represent a multi-criteria decision making domain selected for realism from the Psychology decision literature.

Possibilistic Stable Models

We begin with this section where both Possibilistic Definite Logic Programs (PDLP) and Possibilistic Normal Logic Programs (PNLP) are defined along with Possibilistic Stable Models for both PDLPs and PNLPs.

Let $\mathcal{B}$ be a countable set of atoms α, let $\mathcal{R}$ be a countable set of rules $\alpha \leftarrow \beta_1, \ldots, \beta_n$ where $n \geq 0$, $\alpha, \beta_i \in \mathcal{B}$ for $0 \leq i \leq n$, and let $\mathcal{N}$ be a set of necessity measures w.

A possibilistic atom is then an ordered pair $a = (\alpha, w)$ in the set $\mathcal{B}_\mathcal{N} = \mathcal{B} \times \mathcal{N}$. A possibilistic rule is an ordered pair $r \in \{(\alpha \leftarrow \beta_1, \ldots, \beta_n, w) \mid \alpha, \beta_i \in \mathcal{B}, 0 \leq i \leq n, w \in \mathcal{N}\}$.

We will adopt the notation of r^* and a^* to represent a rule and atom respectively with the weight removed, while r° and a° represent the weight with the rule and atom removed respectively.

Let $\mathcal{B}(\mathcal{P}) = \mathcal{B}$ be the Herbrand base of the program $\mathcal{P}$, $\mathcal{I}$ be a set of atoms forming an interpretation, and let $\mathbb{P}$ denote the powerset operator.

Measure $\Pi_\mathcal{P}(\alpha)$ describes how possible (compatible) atom α is with $\mathcal{P}$, while $N_\mathcal{P}(\alpha)$ describes how strongly α is proved by $\mathcal{P}$.

A possibility distribution places an ordering on the interpretations of the theory. Definition 1 defines a distribution of a PDLP through a function from interpretations to an interval $[0, 1]$. Let $ps(\mathcal{P}^*, B) \subseteq \mathcal{P}^*$ represent the set of rules whose preconditions are satisfied by B, and the function $head$ represent all head atoms in a given set of rules.

Definition 1 (**Distribution of a PDLP**). Given a Possibilistic Definite Logic Program (PDLP) $\mathcal{P}$ and its induced possibility distribution $\pi_\mathcal{P}$ with respect to a set $B \in \mathcal{B}$, then

$$\pi_\mathcal{P}(B) = \begin{cases} 0 : \text{if } B \nsubseteq head(ps(\mathcal{P}^*, B)), \text{ or} \\ \quad ps(\mathcal{P}^*, B) \text{ is not grounded,} \\ 1 : \text{if } \forall r \in \mathcal{P}, B \vDash r^*, \\ 1 - \max_{r \in \mathcal{P}}\{r^\circ \mid B \nvDash r^*\} : \text{otherwise} \end{cases}$$

Definitions 2 and 3 below, capture a Possibilistic Stable Model for a PDLP. Possibility and Necessity measures are defined (induced) by a PDLP $\mathcal{P}$ to represent consistency and certainty information respectively.

Definition 2 (**Measures of a PDLP**). Given a Possibilistic Definite Logic Program (PDLP) $\mathcal{P}$ and its induced possibility distribution $\pi_\mathcal{P}$, the possibility measure $\Pi_\mathcal{P}(\alpha)$ and necessity measure $N_\mathcal{P}(\alpha)$ are given by:

- $\Pi_\mathcal{P}(\alpha) = \max_{\mathcal{I} \in \mathbb{P}(\mathcal{B}(\mathcal{P}))}\{\pi_\mathcal{P}(\mathcal{I}) \mid \alpha \in \mathcal{I}\}$,
- $N_\mathcal{P}(\alpha) = 1 - \max_{\mathcal{I} \in \mathbb{P}(\mathcal{B}(\mathcal{P}))}\{\pi_\mathcal{P}(\mathcal{I}) \mid \alpha \notin \mathcal{I}\}$,

From the definition of a Necessity measure, the Possibilistic Stable Models of a Possibilistic Definite Logic Program can be given.

Definition 3 (**Possibilistic Stable Model of a PDLP**). Given a PDLP $\mathcal{P}$ and its possibility measure $\Pi_\mathcal{P}(\alpha)$ and necessity measure $N_\mathcal{P}(\alpha)$, then its Possibilistic Stable Model is given by $\Pi\mathcal{M}(\mathcal{P}) = \{(\alpha, N_\mathcal{P}(\alpha)) \mid \alpha \in \mathcal{B}(\mathcal{P}), N_\mathcal{P}(\alpha) > 0\}$.

It follows that $\Pi\mathcal{M}(\mathcal{P})^* =$ least model of $\mathcal{P}^*$.

A possibilistic Normal Logic Program (PNLP) may now be defined as an incremental extension from the PDLP.

A possibilistic rule is represented as an ordered pair $r = (\alpha \leftarrow \beta_1, \ldots, \beta_n, not\, \nu_1, \ldots, not\, \nu_m, w)$ for $\nu \in \mathcal{B}$, in the set $\{(\alpha' \leftarrow \beta'_1, \ldots, \beta'_n, not\, \nu'_1, \ldots, not\, \nu'_m, w') \mid \alpha', \beta'_n \in B, n, m \geq 0, w' \in \mathcal{N}\}$.

Let r^+ be the positive-only component of rule r (rule r with negative literals removed). Let $tail^-(r)$ to refer to the negative atoms in the tail (body) of the rule, and $\mathcal{I}$ be a set of possibilistic atoms of the form $a = (\alpha, w)$.

A definition for a Possibilistic stable model (Definition 4) now follows as a straightforward extension of the PDLP case.

Definition 4 (**Possibilistic Stable Model of a PNLP**). Given a PNLP $\mathcal{P}$, and an atom set $B \in \mathcal{B}$, the possibilistic reduction of $\mathcal{P}$ wrt B is the PDLP

$$\mathcal{P}^B = \{(r^{*+}, r^\circ) \mid r \in \mathcal{P}, tail^-(r) \cap B = \emptyset\}.$$

Then $\mathcal{I}$ is a PSM of the PNLP $\mathcal{P}$ iff $\mathcal{I} = \Pi\mathcal{M}(\mathcal{P}^{\mathcal{I}^*})$.

Argumentation Framework

An abstract argumentation framework was defined in (Dung 1995) with the potential to be instantiated in many different forms, each having distinct properties that could also be understood within an overarching framework. (Bondarenko *et al.* 1997) used this technique to show that many types of non-monotonic logics could be expressed equivalently as an argumentation system.

The clarity offered by such a modular method makes it possible to detail an argumentation framework suitable for capturing the Possibilistic Stable Models of a PNLP.

Throughout the discussion on argumentation frameworks we will make use of the following notation. Let $\mathcal{L}_\mathcal{B}$ be a countable set of atoms, and $\mathcal{L}_\mathcal{R}$ be a countable set of rules of the form $\alpha \leftarrow \beta_1, \ldots, \beta_n$ where $n \geq 0$ and $\alpha, \beta_i \in \mathcal{L}_\mathcal{B}$ where $0 \leq i \leq n$. Then $\mathcal{L}$ is a formal language of countably many sentences $\mathcal{L} = \mathcal{L}_\mathcal{B} \cup \mathcal{L}_\mathcal{R}$.

Also let $\mathcal{L}_{\mathcal{BN}}$ be a countable set of possibilistic atoms $a = (\alpha, w)$ where $\alpha \in \mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{\mathcal{RN}}$ be a countable set of possibilistic rules of the form $r = (\alpha \leftarrow \beta_1, ..., \beta_n, w)$ where $r^* \in \mathcal{L}_{\mathcal{R}}$. Then $\mathcal{L}_{\mathcal{N}}$ is a formal language of countably many sentences $\mathcal{L}_{\mathcal{N}} = \mathcal{L}_{\mathcal{BN}} \cup \mathcal{L}_{\mathcal{RN}}$.

The following first two definitions define an underlying monotonic logic, starting with Definition 5 for a propositional weighted deductive system. The deductive system is defined to be aligned with Possibilistic Definite Logic Programming, and also is a counterpart to Possibilistic Logic.

From the deductive system, two types of inference are defined: an inference relying on sufficient support, and a minimal inference requiring the least (with respect to rule subsets) support adequate for the derivation of a given conclusion.

Definition 5 (Monotonic Deductive System). A monotonic deductive system is defined by the pair $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$ where $\vdash$ is defined by the inference rules

- $[\alpha \leftarrow \beta_1, ..., \beta_n, w_0], [(\beta_1, w_1), ..., (\beta_n, w_n)]$ $\vdash [\alpha, min(w_0, ..., w_n)]$
- $(\alpha, w) \vdash (\alpha, w'),\ w' < w$.

Definition 6 defines two types of provability relations. The first is a variant of modus ponens with inferred atom weights equal to the minimum contained in the set of support. The second inference enforces a minimality condition with respect to the set of support: only those premises necessary are permitted.

Definition 6 (Deduction). Given a deductive system $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, a set T of rules of type r, a sentence of possibilistic atoms $\sigma = a_1, ..., a_n$ can be concluded from a premise set T,

- $T \vdash \sigma$ iff σ can be derived through an iterative application of the inference rules to possibilistic propositional formulae in T,
- $T \vdash_{min} \sigma$ iff $T \vdash \sigma$ and there does not exist $T' \subset T$ such that $T' \vdash \sigma$.

An argumentation framework may now be defined in Definition 7, parametric on a background theory, set of assumptions, and a notion of Attack between arguments. Definition 8 then captures the idea that an argument for a conclusion sentence is an inference from a set of assumptions and a background theory. An argument is thus considered to be based on a set of assumptions.

Definition 7 (Argumentation framework). Given a deductive system $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, then an *argumentation framework* wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$ is a tuple $\langle \mathcal{T}, \mathcal{A}, attack \rangle$, where *theory* $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{N}}$, *assumptions* $\mathcal{A} \subseteq \mathcal{L}_{\mathcal{N}}$, and $attack$ is a relation between sets of assumptions.

Definition 8 (Argument). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, an *argument* for a sentence $\sigma \in \mathcal{L}_{\mathcal{N}}$ supported by a set of assumptions $A \in \mathcal{A}$ is a deduction $\mathcal{T} \cup A \vdash \sigma$.

Definition 9 states how two arguments are regarded to be in conflict with one another, and Definition 10 defines the conditions under which one argument attacks another. This definition is set up to be parametric on *stronger*, which will be varied throughout the paper below according to varying requirements. At this point stronger should be considered atomic in nature. A collection $\mathcal{CA}$ of sets of atoms is also introduced with the purpose of defining mutually conflicting (contradictory) atoms. Frequently, this may consist of an atom and its negation $\{\alpha, \neg\alpha\}$, but in general may be any arbitrary set of atoms in the language.

Definition 9 (Conflict). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, and sets of assumptions $A, A' \subseteq \mathcal{A}$, and $a_1, a_2 \in \mathcal{L}_{\mathcal{BN}}$, then A *conflicts with* A' over a, a' iff

- there exists $CA \in \mathcal{CA}$ s.t. $a^*, a'^* \in CA$, and
- $(A \cup \mathcal{T}) \vdash a$, and
- $(A' \cup \mathcal{T}) \vdash a'$.

Definition 10 (Attack). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, and sets of assumptions $A, A' \subseteq \mathcal{A}$, then A *attacks* A' iff there exists $a_1, a_2 \in \mathcal{L}_{\mathcal{BN}}$ such that,

- A *conflicts with* A' over a, a', and
- A' is not *stronger than* A wrt a', a.

A notion of non-monotonic consequence can now be defined by the type of *attack* to which the framework is instantiated. Stable sets, and their associated *extensions* in Definition 11 are a particular instance of credulous consequence, while Definitions 12 and 13 together form the Preferred Extensions.

Definition 11 (Stable set). A *set of assumptions* $A \subseteq \mathcal{A}$ is *stable* iff A attacks every assumption $a \notin A$, and A does not attack itself.

Definition 12 (Admissible set). A *set of assumptions* $A \subseteq \mathcal{A}$ is *admissible* iff A attacks every set of assumptions that attacks A, and A does not attack itself.

Definition 13 (Preferred set). A *set of assumptions* $A \subseteq \mathcal{A}$ is *preferred* iff A is maximal (with respect to set membership) admissible.

Then, E is a stable extension iff $E = Th(\mathcal{P} \cup A)$, where A is a stable set. The consequence set contains all permitted derived weightings for atoms, and may (for example) include $(a, 0.5)$ and its weaker variant $(a, 0.3)$. This characteristic preserves the property that the skeptical consequence set is the intersection of credulous consequence sets[2].

Argumentation System representing a Possibilistic Logic Program

Having now defined both a PSM and an argumentation system, we are in a position to parameterize the latter to capture (represent) the former. An instantiation of the argumentation framework is given below such that its stable extensions are identical to the stable models of a Possibilistic NLP.

[2]This is solely a semantic notion, and for practical implementation, explicit enumeration of such weights would not be necessary.

Conceptually, negation by failure is viewed as default negation, and accordingly the set of assumptions contains all default negation literals. Then, the negation of an atom is assumed unless it is possible to prove the contrary.

For a PNLP $\mathcal{P}$, the instantiation of parameters is as follows. Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, then, a Normal Logic Program $\mathcal{P}$ is represented by its corresponding assumption framework where $attack$ is defined in Definitions 9 and 10.

We introduce the notion of a *contrary atom* through the set $\mathcal{L}_{naf} = \{not\text{-}\alpha \,|\, \alpha \in \mathcal{B}\}$ of special atoms taking the form "not-α" that occur as a counterpart to every normal atom in set $\mathcal{B}$, the Herbrand base of $\mathcal{P}^*$. An associated transformation is made to the program's rules creating a new program $\mathcal{P}' = Tr(\mathcal{P})$ by replacing atoms negated by default in $\mathcal{P}$ with the new corresponding atoms from set $\mathcal{L}_{naf}$.

$$\mathcal{P}' = \{\, (\alpha \leftarrow \beta_1, \ldots, \beta_n, not\text{-}\nu_1, \ldots, not\text{-}\nu_m, w) \;\mid\; (\alpha \leftarrow \beta_1, \ldots, \beta_n, not\, \nu_1, \ldots, not\, \nu_m, w) \in \mathcal{P},\; m, n \geq 0 \,\}$$

The parameters of the argumentation framework may now be assigned straightforwardly:

$\mathcal{L}_{\mathcal{B}} = \mathcal{B} \cup \mathcal{L}_{naf}$,
$\mathcal{L}_{\mathcal{R}} = (\mathcal{P}'^*)$,
$\mathcal{L}_{\mathcal{RN}} = (\mathcal{P}')$,
$\mathcal{L}_{\mathcal{BN}} = \mathcal{B}_{\mathcal{N}}$,
$\mathcal{A} = \mathcal{L}_{naf}$,
$\mathcal{T} = (\mathcal{P}')$,
$\mathcal{CA} = \{\, \{\alpha, not\text{-}\alpha\} \,|\, \alpha \in \mathcal{B} \,\}$.

The notion of attack between two sets of arguments is defined in terms of a potential conflict. It may be noted here that weighting information is not used in PSMs to participate in conflict or attacks between arguments, therefore no argument is stronger than another, and the condition $not\ stronger$ in Definition 10 always holds.

Using the definition of Stable Sets (Definition 11), and the associated Stable Extension, it is possible to draw a correspondence between Possibilistic Stable Models as defined in Definitions 2-4, and Possibilistic Stable Extensions (PSE) of the argumentation framework. In the following theorem let $E^+ = E - \mathcal{L}_{naf}$.

Theorem 1 (**PSM = PSE**). E is a stable extension of the argumentation system corresponding to a program $\mathcal{P}'$ iff $M = E^+$ is a possibilistic stable model of PNLP P.

Proof (sketch)
E is a stable extension of the argumentation system,
$\Leftrightarrow E = \{a \mid P' \cup A_E \vdash a\}$,
(where $A_E = E - \mathcal{L}_{\mathcal{BN}}$),
$\Leftrightarrow \{a \mid P' \cup A_E \vdash a\} - \mathcal{L}_{naf} = \{a \mid P \vdash a\}$,
$\Leftrightarrow \{a \mid P \vdash a\} = M$,
$\Leftrightarrow M = E^+$ is a possibilistic stable model of P.

Necessity Assessment of Argument Strength

One particular difference that exists between qualitative priority frameworks and Possibilistic frameworks is the use to which preference information is put. Priority frameworks make use of preference information to remove particular credulous extensions. Due to the totally ordered nature of weighted rules in Possibilistic Stable Models, if all priority information were used to resolve conflicts between rules, then all stable models except one would be eliminated. For the present section however, this is acceptable because we will control which atoms are conflicting atoms using a declaration.

Having established an argumentation framework for PNLP, we will now substitute preferred extension semantics for stable extension semantics. Also, in order to simplify the language, we will consider just the case of Definite Logic Programs. The resulting system yields a method of assessing arguments based on the weighting of rules.

In this section, let $\mathcal{CA}$ be a collection of sets containing two conflicting atoms: $\mathcal{CA} = \{\{\alpha_1, \beta_1\}, \ldots, \{\alpha_n, \beta_n\}\}$, $\mathcal{D}$ be a set of weighted defeasible rules, and $\mathcal{S}$ be a set of strict (non-defeasible) weighted rules. Finally, we must define *stronger*. One argument is stronger than another when the conflicting rule weights from their respective arguments are arranged so the stronger argument has the greater rule weight, Definition 14. It may be instructive to note that variables a and a' are bound in the definition for conflict.

Definition 14 (**Stronger**). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, sets of assumptions $A, A' \subseteq \mathcal{A}$, then A is *stronger than* A' wrt $a, a' \in \mathcal{L}_{\mathcal{BN}}$ iff $(a^\circ > a'^\circ)$.

In order to identify the relevant properties of this type of possibilistic preferred extension, we will now look at a decision making example.

Application to Decision Making: the resulting Decision Model

Decision making can be seen as one particular type of plausible reasoning. The example below will demonstrate unipolar[3] decision making similar in style to (Fox & Das 2000) by utilizing necessity weights present in the formulae to arbitrate conflicts between arguments induced from the program's rules.

A simple propositional language is employed to represent a robot warrior's decision making over whether it will attempt to traverse one of three pathways: one that is rocky, one that is sandy, and one that contains water. Deciding to traverse the rocky path is represented by gor. A reason to adopt the rocky path choice is $reason1$, and a consequence of adopting the rocky path is $consqr$. Corresponding atoms for the cases of sand and water follow in a similar way. A decision making example is shown below where rules defining $consqr$ represent a substitute for a ramification theory used to derive a comprehensive set of consequences.

$$\mathcal{CA} = \{\, \{reason1, reason2, reason3\} \,\}$$

[3]Unipolar decisions take into consideration only positive information about potential choices, as opposed to positive and negative information as is the case with bipolar decision making.

$\mathcal{D} = \{\ gor \leftarrow, \ldots,$
$gos \leftarrow, \ldots,$
$gow \leftarrow, \ldots,\ \}$

$\mathcal{S} = \{\ consqr \leftarrow gor, \ldots,$
$consqs \leftarrow gos, \ldots,$
$reason1 \leftarrow consqr, \ldots, 0.4$
$reason2 \leftarrow consqs, \ldots, 0.5$
$reason3 \leftarrow consqw, \ldots, 0.5\ \}$

A decision theory can then be represented by the argumentation framework where $\mathcal{A} = \mathcal{D}$ and $\mathcal{T} = \mathcal{S}$. The preferred extensions (with maximal weighting) are then:

$\{(gos), (reason2, 0.5), (consqs)\}$ and
$\{(gow), (reason3, 0.5), (consqw)\}$.

It may be helpful to note that the extension $\{(gor), (reason1, 0.4), (consqr)\}$ was defeated because it conflicted with the other defaults, while having lesser certainty. The system can be seen to have eliminated potential decision choices, but still not come down to a single answer even though there seems sufficient information to do so.

The decision model induced by the system may be characterized by considering a set AC of pairs $(alternative, consequence)$ describing the consequences following from each alternative, and a set CV of triples $(consequence, reason, utility)$ describing projections from consequences to reasons weighted with necessity measures. Then, the decision model realized is the selection of all alternatives corresponding to the maximum utility in the reasons projected.

Let u_{max} be the maximum derived utility of the $reason$ decision atoms, then

$$u_{max} = max\{u \mid \exists c, v.(c, v, u) \in CV\}.$$

The set of consequences corresponding to the maximum utility is given by

$$C_{max} = \{c \mid \exists v.(c, v, u_{max}) \in CV\}.$$

The set of alternatives corresponding to the selected consequences is given by

$$A_D = \{a \mid (a, c), c \in C_{max}\}.$$

The decision choice(s) of the system is thus represented by A_D. We may say that the decision model chooses the alternative(s) corresponding to the 'best' reason. There may be multiple chosen alternatives. This model is very similar to the decision models of (Fox & Das 2000).

Upon examination, a number of shortcomings become evident in this approach. Firstly, the system is unipolar, in the sense that reasons in support of an alternative can only be positive. This makes it difficult to represent risk and danger for example. Reasons in support of an alternative are also assessed in isolation, with no extra weight accruing from a combination of reasons. Related to this was a sensitivity to small changes occurring in utility designations. Lastly, there is no uncertainty representation in the formalism.

Each of these shortcomings will be addressed in the following sections where an alternative extension to PSMs is given.

Aggregated assessment of Argument Strength

It is possible to utilize information already contained in a Possibilistic Logic Program more completely than was achieved in the section above. This involves accumulating, or *aggregating* reasons (justifications) in support of one decision choice over another.

In this framework, an argument is assessed based on the underlying values that it embodies. This makes it possible to dynamically alter the assessment of arguments based on the *values* a particular audience subscribes to. One particular audience could be oneself. In this case, argument advocacy can be viewed as rendering a judgment, or equivalently as decision making: the argument advocates a scenario that one will find to be a convincing rational case given the circumstances. An argument is convincing if it forms the basis of a rational decision.

Let $\mathcal{D}$ be a set of alternatives expressed as unconditional rules (i.e. rules with a head and no body). Let $\mathcal{S}$ be a set of rules optionally weighted with a necessity measure. $\mathcal{S}$ contains a consequence theory defining the conditions under which consequences $c_1, ..., c_n$ occur. From the consequences, a set of rules termed *value projection rules* define the distinguished predicate $value$-p for each relevant value contained within $\mathcal{S}$. Each value projection rule makes use of a unique identifier to ensure it contributes only once, assuring unique reasons are the sole basis for a decision alternative's favorability. The rule takes the form below (with the weight being optional).

$$value\text{-}p(v, vp, ui) \leftarrow c_n, a_1, ..., a_n\ , w$$

Finally, $\mathcal{S}$ contains a set of rules defining another distinguished predicate $value$-w. Values may be positive or negative in nature, however value projections are always positive. In the case of a negative value, a negative value utility, argument w is employed.

$$value\text{-}w(v, vu) \leftarrow a_1, ..., a_n\ , w$$

The set $\mathcal{CA}$ is a collection of sets of mutually conflicting atoms. In the simplest case, it may be a mutually exclusive set of alternatives e.g. $\{\{alt1, alt2, alt3\}\}$ where only one alternative may appear in a decision. This may also be used to create sets of alternatives that may be deployed together, e.g. $\{\{alt1, alt2\}, \{alt2, alt3\}\}$ indicates that any alternative may be used individually, but the pair $alt1, alt3$ (among others) may appear as part of the same decision. This capability is used in the realization of composite decisions.

A decision theory can then be represented by an argumentation framework where $\mathcal{A} = \mathcal{D}$ and $\mathcal{T} = \mathcal{S}$. Let set CS be an argument characteristic set with elements $((vu, w_{vu}), (vp, w_{vp}), vn, pn)$ where vu, vp, vn and pi represent the value utility, value projection, value name and projection unique identifier respectively, and w_{vu} and w_{vp} represent the necessity uncertainty weights of vu, vp respectively.

The definitions for *attack* (Definition 10) and *conflict* (Definition 9) come from the previous section, however a new definition is needed for *stronger*, updated so as to be aware of aggregation (Definition 15). Definition 15 captures the essential intuition of argument evaluation, and makes use of the notion of an $aggregation$ strength. The definition makes

use of an s-aggregation which should be considered atomic until instantiation later.

Definition 15 (**Stronger**). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, sets of assumptions $A, A' \subseteq \mathcal{A}$, then A is *stronger than* A' wrt $a, a' \in \mathcal{L}_{\mathcal{BN}}$ iff

- (A forms an s_1-*aggregation*), (A' forms an s_2-*aggregation*), and $s_1 > s_2$ where $s_1, s_2 \in \mathbb{R}$.

The aggregation strength definition makes use of an aggregation function agg, a function from an argument characteristic set (of weights) CS to a strength measure in $\mathbb{R}$. Definition 16 identifies an argument's characteristic set and aggregates the weights together using a method defined by the aggregation function.

Definition 16 (s-**aggregation**). Given an argumentation framework $\langle \mathcal{T}, \mathcal{A}, attack \rangle$ wrt $\langle \mathcal{L}_{\mathcal{N}}, \vdash \rangle$, sets of assumptions $A \subseteq \mathcal{A}$, a characteristic function agg, then A forms an s-*aggregation* for $s \in \mathbb{R}$ iff $s = agg(CS)$ where

- $CS = \{ ((vu, w_{vu}), (vp, w_{vp}), vn, pi) \mid (A \cup \mathcal{T}) \vdash value\text{-}w(vn, vu), w_{vu} \ , (A \cup \mathcal{T}) \vdash value\text{-}p(vn, vp, pi), w_{vp} \ \}.$

The properties of the argumentation system will clearly be determined in part by the choice of aggregation function. One instantiation will be given here, an analog of a decision function of $|CS|$ reasons. Let p be an individual projection, and let vu^p be the vu component of projection p.

$$agg(CS) = \frac{1}{\sum_{p \in CS} |vu^p.vp^p|} \left(\sum_{p \in CS} (vu^p.w^p_{vu}.vp^p.w^p_{vp}) \right)$$

This particular aggregation function is of a weighted arithmetic mean type. Each projected value is weighted by a real number according to the value weighting in effect, then a weighted mean is taken of a set of necessary utility measures. The weighted mean has several desirable properties such as being continuous, idempotent, linear, additive and self-dual aggregation function. A limitation of this aggregation is that it is not designed to address interactions between utility values. Its use therefore makes an implicit assumption that all values are mutually independent.

Application to Decision Making: the resulting Decision Model

The aggregation based mechanism in the above section makes possible a far more sophisticated decision model than was possible previously. Decision making settings frequently involve the consideration of a *set* of conflicting *objectives*. A candidate *alternative* may then be assessed against several criteria which in aggregate measure the desirability of the alternative's inherent consequences. This approach is known as *multi-criteria decision making*.

The method identifies a set of distinct alternatives (a composite notion containing consequences) and objectives ranging from needs to wants (necessary requirements to desirable properties). A set of criteria is then selected that serves as a performance measure for the objectives. Every alternative is evaluated using criteria chosen to be appropriate for the decision context. The alternative (possibly non-unique) emerging with the highest rank is selected as the decision.

Some authors e.g. (Keeney & Raiffa 1976) advocate taking a more abstract approach to objectives (than is conventional in decision making) through the incorporation of *values*. These are more abstract objectives (or percepts) through which to judge the desirability of an alternative. The preceding section adopted a mechanism to realize the values approach to decision making through the use of value projection rules to provide context dependency in the assessment of alternatives. This serves to identify the (differing) dominant factors underpinning each particular decision. The projection rules work in conjunction with a (static) aggregation function that conducts a cumulative (dynamic) weighting of the projections into their respective criteria.

A set of decision criteria corresponds closely to a set of values in the argumentation framework. Since criteria can be inherently positive or negative in desirability, there needs to be a bipolar evaluation method chosen. Criteria may also have weights which change according to the context of decision making. Context-dependent value weighting therefore appears valuable.

The decision domain is also subject to uncertainty, applicable to both domain knowledge and aspects of the decision structure itself. In this framework, rule weights will be interpreted in their conventional way as necessity measures indicating uncertainty in a rule that may represent the domain, or part of the decision function.

Decision framing determines the basis or *grounds* for the decision. It yields a way of stating a problem formally so that analytical methods may be applied. To frame a decision is to recognize context-specific aspects of its structure; framing is thus dynamic in nature. The choice of criteria relevant to the circumstances and the mapping of domain conditions to those criteria are the most significant aspects of framing a decision. As a secondary factor, the relative weighting applied to the chosen criteria also contributes to a *balance* in the judgment. The importance weighting applied to values induces a total preference order over those values deemed relevant to the decision.

Decision framing is made possible in the argumentation framework using context sensitive, value (criteria) projection rules which map some (relevant) aspect of the situational circumstances to a value being satisfied by some degree. Context dependent value weightings are also possible. Context sensitive value projection rules can be understood as providing relevance descriptions and decision-framing for values we care about. The entire theory will be evaluated using the Possibilistic preferred model entailment. Each rule will use a necessity measure, omitted if it is 1.

The following example was selected from the literature for its known realism and discriminating properties in human decision making (Greene *et al.* 2001).

"A runaway maintenance railway trolley is about to hit and kill five people. Suppose there is a lever that will divert the trolley onto a different track where it will kill only one person instead of five. Suppose also there is a large person (larger than you) who may be pushed in front of the trolley, killing him but saving the others? Is either option suitable?"

Test subjects generally say 'yes' to the first case and 'no' to the second in spite of the fact the cases are so similar. We will now axiomatize the example using the methodology of the preceding section. Axiom block (1) in the program below is the consequence theory, block (2) the projected value theory, and (3) the context-dependent value utilities. (Rules without a weight are assumed to have weight 1.)

$$
\begin{aligned}
\mathcal{CA} &= \{\ \{\textit{no-action},\ \textit{button-pressed},\ \textit{push}(1)\}\ \} \\
\mathcal{D} &= \{\ \ \textit{no-action},\ \textit{button-pressed},\ \textit{push}(1)\ \ \} \\
\mathcal{S} &= \{\ \ \textit{no-involvement} \leftarrow \textit{no-action} \qquad (1)\\
&\quad\ \textit{edited-result} \leftarrow \textit{button-pressed} \\
&\quad\ \textit{approaching}(\textit{train}, \textit{trackA}) \leftarrow \\
&\qquad\qquad \textit{no-action},\ 0.7 \\
&\quad\ \textit{approaching}(\textit{train}, \textit{trackC}) \\
&\quad\ \textit{approaching}(\textit{train}, \textit{trackB}) \leftarrow \\
&\qquad\qquad \textit{button-pressed},\ 0.9 \\
&\quad\ \textit{authored-result} \leftarrow \textit{push}(X) \\
&\quad\ \textit{on}(\textit{trackA}, \textit{people}, 5) \\
&\quad\ \textit{on}(\textit{trackB}, \textit{people}, 1) \\
&\quad\ \textit{on}(\textit{trackC}, \textit{people}, 1) \leftarrow \textit{push}(1) \\
&\quad\ \textit{killed}(N) \leftarrow \textit{approaching}(\textit{train}, T), \\
&\qquad\qquad \textit{on}(T, \textit{people}, N), 0.8 \\
&\quad\ \textit{value-p}(\textit{humanity-loss}, N, \textit{pr}1) \leftarrow \qquad (2)\\
&\qquad\qquad \textit{killed}(N) \\
&\quad\ \textit{value-p}(\textit{individual-harm}, N, \textit{pr}3) \leftarrow \\
&\qquad\qquad \textit{killed}(N), \\
&\qquad\qquad \textit{authored-result} \\
&\quad\ \textit{value-w}(\textit{humanity-loss}, \text{-1}) \qquad (3)\\
&\quad\ \textit{value-w}(\textit{individual-harm}, \text{-1}) \leftarrow \\
&\qquad\qquad \textit{no-involvement} \\
&\quad\ \textit{value-w}(\textit{individual-harm}, \text{-3}) \leftarrow \\
&\qquad\qquad \textit{edited-result} \\
&\quad\ \textit{value-w}(\textit{individual-harm}, \text{-50}) \leftarrow \\
&\qquad\qquad \textit{authored-result}
\end{aligned}
$$

The Possibilistic Preferred semantics instantiated with the aggregation function of section entails just the sets of atoms below. Each set represents the conclusion atoms of a full decision argument evaluated for desirability by the aggregation function. In this example, the decision model reduces the three alternative choices all the way down to a single best case. It is very unlikely for this not to be the case because it would require two competing decision arguments to have the same aggregation strength, something very unlikely for anything but trivial examples.

1. $\{\textit{no-action}, \textit{no-involvement}, \textit{approaching}(\textit{train}, \textit{trackC}), \textit{approaching}(\textit{train}, \textit{trackA}), 0.7, \textit{on}(\textit{trackA}, \textit{people}, 5), \textit{killed}(5), 0.7, \textit{value-p}(\textit{humanity-loss}, 5, \textit{pr}1), 0.7, \textit{value-w}(\textit{humanity-loss}, \text{-1}), \textit{value-w}(\textit{individual-harm}, \text{-1})\}$
2. $\{\textit{button-pressed}, \textit{edited-result}, \textit{approaching}(\textit{train}, \textit{trackC}), \textit{approaching}(\textit{train}, \textit{trackB}), 0.9, \textit{on}(\textit{trackA}, \textit{people}, 5), \textit{on}(\textit{trackB}, \textit{people}, 1), \textit{killed}(1), 0.8, \textit{value-p}(\textit{humanity-loss}, 1, \textit{pr}1), 0.8, \textit{value-w}(\textit{humanity-loss}, \text{-3})\}$
3. $\{\textit{push}(1), \textit{authored-result}, \textit{killed}(1), 0.8, \textit{approaching}(\textit{train}, \textit{trackC}), \textit{on}(\textit{trackA}, \textit{people}, 5), \textit{on}(\textit{trackB}, \textit{people}, 1), \textit{on}(\textit{trackC}, \textit{people}, 1), \textit{value-p}(\textit{humanity-loss}, 1, \textit{pr}1), 0.8, \textit{value-p}(\textit{individual-harm}, 1, \textit{pr}3), 0.8, \textit{value-w}(\textit{humanity-loss}, \text{-1}), \textit{value-w}(\textit{individual-harm}, \text{-50})\}$.

Set one has an aggregation value of -3.5, set two -0.8 and set three -0.816. In this framework, the atoms in set two are the only atoms entailed by the system either skeptically, or credulously. Arguments one and three are subjected to attacks while lacking sufficient power to defend themselves. The alternative of pressing the button is thus the selected decision of the system.

The result may be understood more clearly if we characterize the decision model. For an argument A, given a set of derived value weightings VW with elements $(\textit{value}, \textit{weighting}, \textit{uncertainly})$ and a set of derived value projections VP with elements $(\textit{value}, \textit{utility}, \textit{uncertainty}, \textit{projection-identifier})$ then the argument characterization set for argument A denoted by CS_A may be defined as

$$
\begin{aligned}
CS_A = \{\ &((vu, w_{vu}), (vp, w_{vp}), vn, pi)\ \mid \\
&(vn, vu, w_{vu}, rn, pi) \in VP, \\
&(vn, w_{vu}, w_{vp}) \in VW\ \}
\end{aligned}
$$

Maximum argument desirability evaluated for all non-self-conflicting arguments may then be defined as

$$m = max\{a \mid \forall \mathrm{A} \in \mathit{Args}\ ,\ a = agg(CS_A)\}.$$

The set of alternatives is then selected as those corresponding to maximum argument desirability as defined by the aggregation function.

Although this particular case represents just one decision, an argument may contain more than one decision: a composite decision comprised of many decisions. The aggregation function is able to assess multiple decisions concurrently, rendering a result globally optimal in terms of maximum necessary utility for the composite decision. In this system however, each successful sub-decision does have the same criteria weighting context. This may be justified on the grounds of consistency in rational decision making, but it does require more investigation to assess whether this is universally applicable for real-world multiple decision making.

Discussion

Decision making has been addressed previously in an argumentation logic programming setting by (Kakas & Moraitis 2003) where qualitative preferences between rules are used to resolve conflicts between argument conclusions. The semantics for this system are somewhat close to, but do not

quite coincide with, the weakest-link principle for the determination of the strength of a conclusion. Thus, the decisions rendered by the system differ somewhat from a qualitative preference version of the first decision system in the present paper.

Possibilistic argumentation in the language of logic programming has also been addressed in (Chesñevar *et al.* 2004), however the focus of the work was not specifically on decision making as it is in this paper, and therefore most of the features employed here to realize decision models have no direct counterpart in the work. However, it would be interesting from a general reasoning standpoint to make a comparison with the design choices made here.

Some comparison may be made between the use of values in the decision model and those introduced into argumentation systems by (Bench-Capon 2002). However in the work, the definition of values were static (not context dependent), and there was no notion of aggregation or multi-criteria evaluation.

In the works of (Amgoud & Prade 2004) and (Amgoud & Prade) a system for decision making in argumentation was introduced. The focus of the work appears to be on uncertainty management, rather than on the authenticity of modeling of the structure of decision making. Accordingly, there is no counterpart to context sensitivity of values or weights, and neither is there a method of mapping consequences onto values, making dynamic value allocation difficult. The framework also does not account for the variations in how values are applied, for instance some domains are value-additive while some not.

When applying the aggregation framework presented in earlier sections, it is important to recognize that the system accrues projected reasons rather than domain facts. This yields considerably greater flexibility for handling what otherwise would be pathological examples involving interaction between accrued domain facts. The use of projection rules for this purpose is exemplified in the axiomatization of the railway decision domain selected for its practical realism and displaying many of the system's extended capabilities.

Conclusion

Due to the non-involvement of rule weights in the outcome of rule conflicts in Possibilistic Stable Models for logic programming, it is difficult to realize decision making functionality possessing an adequate degree of realism. In order to make extensions to the system in a uniform way, we began by devising an argumentation framework capable of capturing PSMs. A correspondence was then demonstrated between the argumentation system's Possibilistic Stable Extension and the Possibilistic Stable Model. The resulting generality and expressive power of the argumentation system was used to make several extensions to PSMs suitable for decision making.

Through the incorporation of sets of declared conflict atoms, projection axioms, and a context-aware aggregation function, it has been possible to realize multi-criteria bipolar decision making with automatic framing (structuring) of the decision. Decision framing has the effect of selecting the dominant factors for each particular decision. To our knowledge, these capabilities are novel.

The system presented realizes automatic decision framing by a value projection theory. Value (criteria) projection rules map domain circumstances to values being satisfied by some degree, solving the value management problem, akin to a "frame problem" for values. This also yields a clear demarcation between consequences and values. Projection rules allow a clear distinction to be drawn between primary alternative choices and their secondary consequences. Context dependent value weightings where value weightings are derived from within the theory may be applied to the modeling of domains possessing values whose relative weighting varies depending on the type of decision undertaken. Context sensitivity of value projections where value projections are derived from within the theory are useful for domains possessing decision criteria whose applicability varies depending on the type of decision undertaken. The use of a strength aggregation based on the projected information yields more flexibility in domain representation over prior direct decision models. We believe individually and in combination, these approaches represent a novel enhancement to the methods presented previously in the literature.

An important part of the structural construction of the decision context concerns the ability of a decision system to predict a full set of consequences arising from a given candidate alterative. Depending on the type of implementation, it may be appropriate to incorporate a modern ramification theory for this purpose.

In future work for an extended version of this paper, particular classes of decision examples from the literature will be applied to the system to discover more fully how its capabilities may be applied and developed.

References

Amgoud, L., and Prade, H. Using arguments for making decisions: A possibilistic logic approach.

Amgoud, L., and Prade, H. 2004. Towards argumentation-based decision making: A possibilistic logic approach. In *Proc. of the International Conference on Fuzzy Systems, Budapest, 25-29/07/04*, 1531–1536. IEEE.

Bench-Capon, T. J. M. 2002. Value-based argumentation frameworks. In Benferhat, S., and Giunchiglia, E., eds., *9th International Workshop on Non-Monotonic Reasoning (NMR 2002), April 19-21, Toulouse, France, Proceedings*, 443–454.

Bondarenko, A.; Dung, P. M.; Kowalski, R. A.; and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence* 93(1–2):63–101.

Brewka, G. 1994. Adding priorities and specificity to default logic. *Lecture Notes in Computer Science* 838:247–260.

Chesñevar, C. I.; Simari, G. R.; Alsinet, T.; and Godo, L. 2004. A logic programming framework for possibilistic argumentation with vague knowledge. In Chickering, D. M., and Halpern, J. Y., eds., *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, July 7-11 2004, Banff, Canada*, 76–84. AUAI Press.

Delgrande, J. P., and Schaub, T. 2000. Expressing preferences in default logic. *Artificial Intelligence* 123(1–2):41–87.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic. In Gabbay, D.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford: Oxford University Press. 439–513.

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):321–257.

Fox, J., and Das, S. 2000. *Safe and Sound: Artificial Intelligence in Hazardous Applications*. AAAI Press.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, 1070–1080.

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

Greene, J.; Sommerville, R.; L.E.Nystrom; J.M.Darley; and J.D.Cohen. 2001. An fMRI investigation of emotional engagement in moral judgment. *Science* 293:2105–2108.

Kakas, A., and Moraitis, P. 2003. Argumentation based decision making for autonomous agents. In Rovatsos, M., and Rahwan, I., eds., *Second International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 17, 883–890.

Keeney, R. L., and Raiffa, H. 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. New York: Wiley and Sons.

Nicolas, P.; Garcia, L.; and Stéphan, I. 2005. Possibilistic stable models. In Brewka, G.; Niemelä, I.; Schaub, T.; and Truszczynski, M., eds., *Nonmonotonic Reasoning, Answer Set Programming and Constraints*, volume 05171 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics* 7:25–75.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Zadeh, L. 1978. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems* 1:3–28.

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

Formalizing Accrual in Defeasible Logic Programming

Mauro J. Gómez Lucero and **Carlos I. Chesñevar** and **Guillermo R. Simari**
National Council of Scientific and Technical Research (CONICET)
Artificial Intelligence Research & Development Laboratory
Department of Computer Science and Engineering, Universidad Nacional del Sur
Av. Alem 1253, (B8000CPB) Bahía Blanca, Argentina
Tel: +54-291-459-5135 / Fax: +54-291-459-5136
Email: {mjg, cic, grs}@cs.uns.edu.ar

Abstract

Argumentation has evolved as a powerful paradigm to formalize commonsense qualitative reasoning. Several argumentation frameworks have been developed, notably Defeasible Logic Programming (DeLP), a logic programming approach to argumentation which has proven to be successful for many real-world applications. Recently the notion of accrual of arguments has received some attention from the argumentation community. Three principles for argument accrual have been identified as necessary to hold in argumentation frameworks. In this paper we propose an approach to model the accrual of arguments based on the language and notion of argument of DeLP. We will analyze the above principles in the context of our proposal, studying as well other interesting properties.

Introduction

In the last years, argumentation has evolved as a powerful paradigm to formalize commonsense qualitative reasoning. Different argument-based frameworks have been developed, e.g. (Dung 1995; García and Simari 2004; Besnard and Hunter 2001). In order to determine whether a given conclusion is finally accepted, most argumentation systems perform a dialectical process in which arguments in favor and against the conclusion are taken into account. Among all existing approaches there exists a whole family of argumentative frameworks (e.g. (Prakken and Sartor 1997; García and Simari 2004; Alsinet et al. 2008)) that resulted of integrating extensions of logic programming with argumentation, providing a natural way of combining knowledge representation and reasoning. In particular, Defeasible Logic Programming (DeLP) (García and Simari 2004) has proven to be a successful representative of this family, finding application in the context of different real-world problems (e.g. (Chesñevar, Maguitman, and Simari 2006)).

The notion of *accrual of arguments* has received some attention from the argumentation community (Verheij 1996; Prakken 2005). This notion is based on the intuitive idea that having more reasons or arguments for a given conclusion makes such a conclusion more credible. Modelling accrual of arguments is not a simple issue, and previous research (Prakken 2005) has identified different principles that should hold for performing accrual of arguments in a sound way.

In this paper we propose an approach to model accrual of arguments in the context of DeLP. We show that accrued arguments can be conceptualized as structures which can be subject to a dialectical analysis similar to the one applied in conventional argumentation systems. We also analyze Prakken's principles in the context of our proposal, and describe some valuable features of our approach.

The rest of this paper is structured as follows. The next section briefly describes DeLP. Next we present the notion of *accrued structure*, which plays a central role in our proposal. Based on this notion, we then formalize the notions of attack and defeat among accrued structures. We show then how to perform a dialectical analysis on accrued structures, formalizing the notion of justified accrued structure. Next, we discuss related work and describe some significant features of our approach. Finally, we present the main conclusions that have been obtained.

The DeLP system: a brief overview

Next we will briefly introduce DeLP (for more details see (García and Simari 2004)). As we will see in the next section, DeLP will provide a natural context for modeling the accrual of arguments. We begin by introducing its language.

Definition 1 (DeLP Language)**.** *The DeLP language is defined in terms of three disjoint sets: a set of facts which are literals, a set of strict rules of the form* $L_0 \leftarrow L_1, ..., L_k$*, and a set of defeasible rules of the form* $L_0 \prec L_1, ..., L_k$*, where* L_0*,* L_1*, ...,* L_k*, with* $k > 0$*, are literals. In the language of DeLP, a literal "*L*" is a ground atom "*A*" or a negated ground atom "*$\sim A$*", where "*$\sim$*" represents the strong negation.*

Pragmatically, facts and strict rules will be used to represent strict (non defeasible) information (e.g. $mammal \leftarrow dog$) whereas defeasible rules will be used to represent tentative or weak information (e.g. $flies \prec bird$).

Definition 2 (DeLP program)**.** *A DeLP program* $\mathcal{P}$ *is a finite set of facts, strict rules and defeasible rules. In a program* $\mathcal{P}$ *we will distinguish the subset* Π *of facts and strict rules, and the subset* Δ *of defeasible rules. When required, we will denote* $\mathcal{P}$ *as* (Π, Δ).

[1]Partially supported by CONICET, Universidad Nacional del Sur and Agencia Nacional de Promoción Científica y Tecnológica.

Example 1. *The following constitutes a DeLP program:*

$$\mathcal{P} = \left\{ \begin{array}{llll} a \prec b, c & b \prec d & c & f \\ a \prec b, f & b \prec e & d & h \\ a \prec g & g \leftarrow h & e & \end{array} \right\}$$

Definition 3 (Defeasible derivation)**.** *Let $\mathcal{P}$ be a DeLP program and L a ground literal. A defeasible derivation of L from $\mathcal{P}$, consists of a finite sequence $L_1, \ldots, L_n = L$ of ground literals, such that for each i, $1 \leq i \leq n$, L_i is a fact or there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body $B_1, \ldots, B_m$, such that each literal on the body of the rule is an element L_j of the sequence appearing before L_i ($j \leq i$).*

We say that a given set of DeLP wffs is contradictory if and only if there exists a defeasible derivation for a pair of complementary literals (w.r.t. strong negation) from this set.

Definition 4 (Argument)**.** *Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP program. We will say that $\langle A, h\rangle$ is an argument for a literal h from $\mathcal{P}$, if A is the minimal set of defeasible rules ($A \subseteq \Delta$), such that: (1) there exists a defeasible derivation for h from $\Pi \cup A$, and (2) the set $\Pi \cup A$ is non-contradictory.*

Definition 5 (Subargument)**.** *An argument $\langle B, q\rangle$ is a subargument of an argument $\langle A, h\rangle$ if $B \subseteq A$.*

Example 2. *Consider the DeLP program in Ex. 1. Then h, g, a and d, b, c, a are defeasible derivations for a, $\langle A_1, a\rangle = \langle\{(a \prec b, c), (b \prec d)\}, a\rangle$ and $\langle A_2, a\rangle = \langle\{a \prec g\}, a\rangle$ are arguments, and $\langle\{b \prec d\}, b\rangle$ is a subargument of $\langle A_1, a\rangle$.*

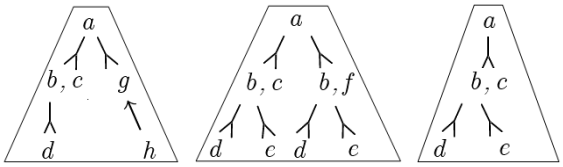
The attack among arguments in DeLP is defined in terms of the notion of *disagreement* of literals. Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$, two literals h_1 and h_2 are in disagreement (or just *disagree*) iff the set $\Pi \cup \{h_1, h_2\}$ is contradictory. Then, given two arguments $\langle A, h\rangle$ and $\langle B, k\rangle$ in $\mathcal{P}$, $\langle B, k\rangle$ attacks $\langle A, h\rangle$ at literal h' iff there exist a subargument $\langle A', h'\rangle$ of $\langle A, h\rangle$ such that k and h' disagree. The subargument $\langle A', h'\rangle$ is called the disagreement subargument.

Modeling the Accrual of Arguments

We will introduce next the notion of *accrued structure* in order to model the accrual of different arguments for the same conclusion.

Definition 6 (Accrued Structure)**.** *Let $\mathcal{P}$ be a DeLP program, and let Ω be a set of arguments in $\mathcal{P}$ supporting the same conclusion h, i.e., $\Omega = \{\langle A_1, h\rangle, \langle A_2, h\rangle, ..., \langle A_n, h\rangle\}$. We define the accrued structure for h (or just a-structure) from the set Ω (denoted $Accrual(\Omega)$) as $[\Phi, h]$, where $\Phi = A_1 \cup A_2 \cup ... \cup A_n$. When $\Omega = \emptyset$ we get the special accrued structure $[\emptyset, \epsilon]$, representing the accrual of no argument.*

Example 3. *Consider the DeLP program $\mathcal{P}$ in Ex. 1. Let*
$\langle A_1, a\rangle = \langle\{(a \prec b, c), (b \prec d)\}, a\rangle$,
$\langle A_2, a\rangle = \langle\{(a \prec b, c), (b \prec e)\}, a\rangle$,
$\langle A_3, a\rangle = \langle\{(a \prec b, f), (b \prec e)\}, a\rangle$ and
$\langle A_4, a\rangle = \langle\{a \prec g\}, a\rangle$ be arguments in $\mathcal{P}$. Then
$Accrual(\{\langle A_1, a\rangle, \langle A_4, a\rangle\}) = [\Phi_1, a]$ where
$\Phi_1 = \{(a \prec b, c), (b \prec d), (a \prec g)\}$ (Fig. 1a)
$Accrual(\{\langle A_1, a\rangle, \langle A_3, a\rangle\}) = [\Phi_2, a]$ where
$\Phi_2 = \{(a \prec b, c), (a \prec b, f), (b \prec d), (b \prec e)\}$ (Fig. 1b)
$Accrual(\{\langle A_1, a\rangle, \langle A_2, a\rangle\}) = [\Phi_3, a]$ where
$\Phi_3 = \{(a \prec b, c), (b \prec d), (b \prec e)\}$ (Fig. 1c)



$[\Phi_1, a]$ (a) $[\Phi_2, a]$ (b) $[\Phi_3, a]$ (c)

Figure 1: Accrued Structures

An a-structure for a conclusion h can be seen as a special kind of argument which subsumes different chains of reasoning which provide support for h. For instance, the a-structure $[\Phi_1, a]$ (see Fig. 1a) provides two alternative chains of reasoning supporting a, both coming from each of the arguments accrued. On the one hand, a because of b and c, and b because of d. On the other hand, a because of g. Note that the graphical representation of a-structures shows both strict and defeasible rules of the subsumed chains of reasoning (although the a-structure itself has only defeasible rules).

The case of $[\Phi_2, a]$ in Ex. 3 illustrates an important feature of our notion of accrual. If two arguments for the same conclusion share some intermediate conclusion but support it in different ways, then by accruing them the reasons for the intermediate conclusion also accrue. Fig. 1b shows this situation for two different reasons for the intermediate conclusion b. The case of $[\Phi_2, a]$ in Ex. 3 highlights another feature of our characterization of accrual. Although each of the arguments accrued stands for one chain of reasoning supporting a conclusion a, the resulting a-structure $[\Phi_2, a]$ stands for four chains of reasoning for a (two of them are not explicitly present in the individual arguments accrued).

The case of $[\Phi_3, a]$ in Ex. 3 illustrates a situation similar to the previous one, where the arguments involved share not only the intermediate conclusion b but also their topmost parts (more precisely the rule $a \prec b, c$), first differing in the reasons supporting b. As shown in the the graphical representation of $[\Phi_3, a]$ in Fig. 1c, we consider that a-structures factor out as many common steps of reasoning (of the arguments accrued) as possible. In this way we can distinguish that there are two different reasons for the intermediate conclusion b, and consequently that there are two reasons for the final conclusion a. Note that in the graphical representation of $[\Phi_3, a]$ there are two (defeasible) arrows pointing to b and only one pointing to a.

An important question that naturally emerges when considering the way we accrue arguments is what happens when accruing two arguments that are in conflict (for instance because they have contradictory intermediate conclusions). We will come back to this issue later.

Definition 7. *Let $[\Phi, h]$ be an a-structure. Then the set of arguments in $[\Phi, h]$, denoted as $Args([\Phi, h])$, is the set of all arguments $\langle A, h\rangle$ such that $A \subseteq \Phi$. Note that $Args([\emptyset, \epsilon]) = \emptyset$.*

Example 4. *Consider the a-structures $[\Phi_3, a]$ and $[\Phi_2, a]$ in Ex. 3. Then $\langle\{(a \prec b, c), (b \prec d)\}, a\rangle$ and $\langle\{(a \prec b, c), (b \prec e)\}, a\rangle$ are all the arguments in $Args([\Phi_3, a])$ and $\langle\{(a \prec b, c), (b \prec d)\}, a\rangle$, $\langle\{(a \prec b, c), (b \prec e)\}, a\rangle$, $\langle\{(a \prec b, f), (b \prec d)\}, a\rangle$ and $\langle\{(a \prec b, f), (b \prec e)\}, a\rangle$ are*

all the arguments in $Args([\Phi_2, a])$.

Although $Accrual$ and $Args$ are not reverse operations (as illustrated by the case of $[\Phi_2, a]$ in Exs. 3 and 4), we can ensure that the arguments accrued will always be among the arguments in the resulting a-structure. We can also ensure that by accruing the arguments in a given a-structure $[\Psi, k]$ we always get $[\Psi, k]$ as a result.

Property 1. *Let* Ω *be a set of arguments for a given conclusion* h. *Then* $Args(Accrual(\Omega)) \supseteq \Omega$. *Besides, for any a-structure* $[\Psi, k]$ *it holds that* $Accrual(Args([\Psi, k])) = [\Psi, k]$ [2].

Definition 8 (Maximal a-structure)**.** *Let* $\mathcal{P}$ *be a DeLP program. We say that an a-structure* $[\Phi, h]$ *is maximal iff* $Args([\Phi, h])$ *contains all arguments in* $\mathcal{P}$ *with conclusion* h.

Example 5. *Consider the DeLP program* $\mathcal{P}$ *in Ex. 1. Then* $[\{(b \prec d), (b \prec e)\}, b]$ *is a maximal a-structure in* $\mathcal{P}$, *whereas* $[\{(b \prec d)\}, b]$ *is not.*

Next we will formally define binary operations for the *addition* and *substraction* of a-structures with the same conclusion.

Definition 9. *Let* $[\Phi, h]$ *and* $[\Psi, h]$ *be two a-structures. Then we define the addition (+) and subtraction (-) of a-structures as follows:*

- $[\Phi, h] + [\Psi, h] =_{def} Accrual(Args([\Phi, h]) \cup Args([\Psi, h]))$
- $[\Phi, h] - [\Psi, h] =_{def} Accrual(Args([\Phi, h]) \setminus Args([\Psi, h]))$

Note that $[\emptyset, \epsilon]$ is the identity element for '+', *i.e.*, $[\Phi, h] + [\emptyset, \epsilon] = [\Phi, h]$. We have also that $[\Phi, h] - [\Phi, h] = [\emptyset, \epsilon]$.

Example 6. *Consider the a-structures* $[\Phi_2, a]$ *and* $[\Phi_3, a]$ *in Ex. 3, and let* $[\Phi_4, a] = [\{(a \prec b, f), (b \prec d), (b \prec e)\}, a]$. *Then* $[\Phi_3, a] + [\Phi_4, a] = [\Phi_2, a]$ *and* $[\Phi_2, a] - [\Phi_3, a] = [\Phi_4, a]$.

Next we will introduce the notion of *narrowing* of an a-structure, which is analogous to the notion of narrowing in (Verheij 1996). Intuitively, a narrowing of an a-structure $[\Phi, h]$ is an a-structure $[\Theta, h]$ accounting for a subset of $Args([\Phi, h])$.

Definition 10 (Narrowing of an a-structure)**.** *Let* $[\Phi, h]$ *and* $[\Theta, h]$ *be two a-structures. We say that* $[\Theta, h]$ *is a narrowing of* $[\Phi, h]$ *iff there exists an a-structure* $[\Lambda, h]$ *such that* $[\Phi, h] = [\Theta, h] + [\Lambda, h]$.

Example 7. *Consider the a-structures* $[\Phi_2, a]$ *and* $[\Phi_3, a]$ *in Ex. 3. Then* $[\Phi_3, a]$ *is a narrowing of* $[\Phi_2, a]$, *because* $[\Phi_2, a] = [\Phi_3, a] + [\{(a \prec b, f), (b \prec d), (b \prec e)\}, a]$.

Definition 11 (a-substructure)**.** *Let* $[\Phi, h]$ *and* $[\Theta, k]$ *be two a-structures such that every argument* $\langle B_i, k\rangle \in Args([\Theta, k])$ *is a subargument of some argument* $\langle A, h\rangle \in Args([\Phi, h])$. *Then we say that* $[\Theta, k]$ *is an accrued substructure (o just a-substructure) of* $[\Phi, h]$.

Intuitively, $[\Theta, k]$ is an a-substructure of $[\Phi, h]$ whenever $[\Theta, k]$ accounts for the accrual of different subarguments (all for the same conclusion) of arguments in $Args([\Phi, h])$.

Example 8. *Consider the a-structure* $[\Phi_3, a]$ *in Ex. 3. Then* $[\{b \prec d\}, b]$, $[\{(b \prec d), (b \prec e)\}, b]$ *and* $[\Phi_3, a]$ *itself are a-substructures of* $[\Phi_3, a]$.

[2] For space reasons, proofs for properties are not included.

Intuitively, an a-substructure $[\Theta, k]$ of $[\Phi, h]$ is complete if it is the largest a-substructure of $[\Phi, h]$ for the conclusion k.

Definition 12 (Complete a-substructure)**.** *Let* $[\Phi, h]$ *be an a-structure. Let* $[\Theta, k]$ *be an a-substructure of* $[\Phi, h]$ *such that* $Args([\Theta, k])$ *is the set of all subarguments* $\langle B_i, k\rangle$ *of arguments in* $Args([\Phi, h])$. *Then we will say that* $[\Theta, k]$ *is a complete a-substructure.*

Example 9. *Consider the a-structure* $[\Phi_3, a]$ *in Ex. 3. Then* $[\{(b \prec d), (b \prec e)\}, b]$ *is a complete a-substructure of* $[\Phi_3, a]$.

Conflict and Defeat among Accrued Structures

Next we will formalize the notion of attack between a-structures, which differs from the notion of attack in argumentation frameworks in several respects. First, an a-structure $[\Phi, h]$ generally stands for more than one chain of reasoning (argument) supporting the conclusion h. Besides, some intermediate conclusions in $[\Phi, h]$ could be shared by some, but not necessarily all the arguments in $Args([\Phi, h])$. Thus, given two a-structures $[\Phi, h]$ and $[\Psi, k]$, if the conclusion k of $[\Psi, k]$ contradicts some intermediate conclusion r in $[\Phi, h]$, then only those arguments in $Args([\Phi, h])$ involving r will be affected by the conflict.

Next we will define the notion of *partial attack*, where the attacking structure generally affects only a narrowing of the attacked one (that one containing exactly the arguments in the attacked a-structure affected by the conflict).

Definition 13 (Partial Attack)**.** *Let* $\mathcal{P}$ *be a DeLP program. Let* $[\Phi, h]$ *and* $[\Psi, k]$ *be two a-structures in* $\mathcal{P}$. *We say that* $[\Psi, k]$ *partially attacks* $[\Phi, h]$ *at literal* h', *iff there exists a complete a-substructure* $[\Phi', h']$ *of* $[\Phi, h]$ *such that* k *and* h' *disagree. The a-substructure* $[\Phi', h']$ *will be called the disagreement a-substructure.*

Example 10. *Consider a DeLP program* $\mathcal{P}$ *where:*

$$\mathcal{P} = \left\{ \begin{array}{llll} x \prec z & \sim z \prec w & \sim x \prec q & u \\ x \longleftarrow y & \sim z \prec s & s \prec p & v \\ z \prec t & y \prec u & y & w \\ z \prec v & \sim y \prec p & t & p \end{array} \right\}$$

Consider the a-structures $[\Phi, x]$ *and* $[\Psi_1, \sim z]$ *in Fig. 2, where* $\Phi = \{(x \prec z), (z \prec t), (z \prec v), (y \prec u)\}$ *and* $\Psi_1 = \{(\sim z \prec w), (\sim z \prec s), (s \prec p)\}$. *Then* $[\Psi_1, \sim z]$ *partially attacks* $[\Phi, x]$ *with disagreement a-substructure* $[\Phi', z] = [\{(z \prec t), (z \prec v)\}, z]$. *Graphically, this attack relation will be depicted with a dotted arrow (see Fig. 2).*

Given an attack between two a-structures, the narrowing of the attacked a-structure affected will be referred to as its *attacked narrowing*, and it is formally defined as follows.

Definition 14 (Attacked Narrowing)**.** *Let* $[\Phi, h]$ *and* $[\Psi, k]$ *be two a-structures such that* $[\Psi, k]$ *partially attacks* $[\Phi, h]$ *with disagreement a-substructure* $[\Phi', h']$. *Then we say that* $[\Lambda, h]$ *is the attacked narrowing of* $[\Phi, h]$ *iff* $[\Lambda, h]$ *is the minimal narrowing of* $[\Phi, h]$ *that has* $[\Phi', h']$ *as an a-substructure.*

Example 11. *Consider the attack from* $[\Psi_1, \sim z]$ *to* $[\Phi, x]$ *in Ex. 10. The attacked narrowing of* $[\Phi, x]$ *is* $[\{(x \prec z), (z \prec t), (z \prec v)\}, x]$ *(see Fig. 2).*

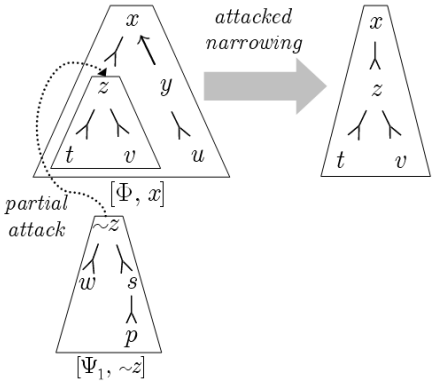


Figure 2: Partial Attack

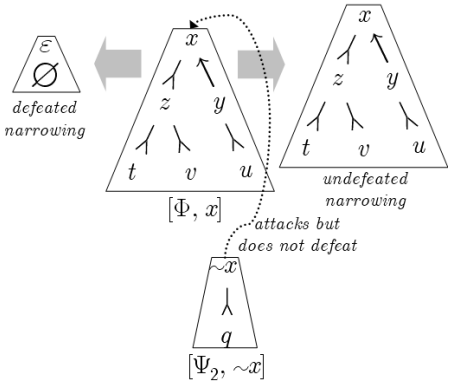


Figure 4: Defeated and Undefeated Narrowings

Accrued Structures: Evaluation and Defeat

In order to decide if a partial attack really succeeds and constitutes a defeat we need a criterion to determine the relative strength (or conclusive force) of those a-structures in conflict. In general, such comparison criterion must be defined according to the application domain. In what follows, we will abstract from that criterion assuming the existence of a binary preference relation '$\gg$' between a-structures.

Definition 15 (Partial Defeater)**.** *Let $[\Phi, h]$ and $[\Psi, k]$ be two a-structures. Then we say that $[\Psi, k]$ is a partial defeater of $[\Phi, h]$ (or equivalently that $[\Psi, k]$ is a successful attack on $[\Phi, h]$) iff*

- *$[\Psi, k]$ attacks $[\Phi, h]$ at literal h', where $[\Phi', h']$ is the disagreement a-substructure and*
- *it is not the case that $[\Phi', h'] \gg [\Psi, k]$.*

Example 12. *Consider the attack from $[\Psi_1, \sim z]$ to $[\Phi, x]$ with disagreement a-substructure $[\Phi', z]$ in Ex. 10 (Fig. 2), and suppose that $[\Psi_1, \sim z] \gg [\Phi', z]$. Then the attack succeeds, constituting a defeat. Graphically, this defeat relation will be depicted with a continuous arrow (see Fig. 3).*

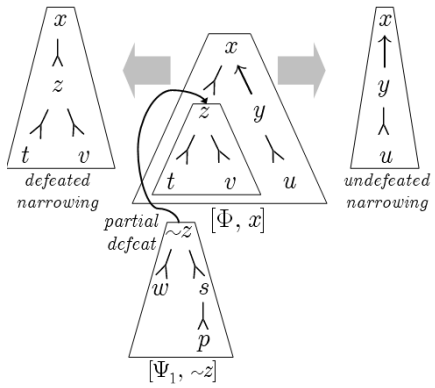


Figure 3: Defeated and Undefeated Narrowings

Given an attack relation, we will identify two complementary narrowings associated with the attacked a-structure: the narrowing that becomes defeated as a consequence of the attack, and the narrowing that remains undefeated.

Definition 16 (Undefeated and Defeated narrowings)**.** *Let $[\Phi, h]$ and $[\Psi, k]$ be two a-structures such that $[\Psi, k]$ attacks $[\Phi, h]$. Let $[\Lambda, h]$ be the attacked narrowing of $[\Phi, h]$. Then the defeated narrowing of $[\Phi, h]$ associated with the attack, denoted as $\mathsf{N}^{\mathsf{D}}_{\mathsf{w}}([\Phi, h], [\Psi, k])$, is defined as follows:*

$$\mathsf{N}^{\mathsf{D}}_{\mathsf{w}}([\Phi, h], [\Psi, k]) =_{def} \begin{cases} [\Lambda, h] & \text{if } [\Psi, k] \text{ is a partial defeater of } [\Phi, h]; \\ [\emptyset, \epsilon] & \text{otherwise.} \end{cases}$$

The undefeated narrowing of $[\Phi, h]$ associated with the attack, denoted as $\mathsf{N}^{\mathsf{U}}_{\mathsf{w}}([\Phi, h], [\Psi, k])$, is the a-structure $[\Phi, h] - \mathsf{N}^{\mathsf{D}}_{\mathsf{w}}([\Phi, h], [\Psi, k])$.

Example 13. *Fig. 3 illustrates a successful attack from $[\Psi_1, \sim z]$ to $[\Phi, x]$, as well as the defeated and undefeated narrowings of $[\Phi, x]$ associated with the attack. Fig. 4 illustrates an unsuccessful attack from $[\Psi_2, \sim x] = [\{\sim x \prec q\}, \sim x]$ to $[\Phi, x]$ with $[\Phi, x]$ itself as disagreement a-substructure, assuming that $[\Phi, x] \gg [\Psi_2, \sim x]$.*

Combined Attack

Until now we have considered only *single* attacks. When a single attack succeeds, a nonempty narrowing of the attacked a-structure becomes defeated. But two or more a-structures could simultaneously attack another, possibly affecting different narrowings of the target a-structure, and thus causing a bigger narrowing to become defeated (compared with the defeated narrowings associated with the individual attacks). Fig. 5 illustrates a *combined* attack from the a-structures $[\Psi_1, \sim z]$ and $[\Psi_3, \sim y]$ against $[\Phi, x]$. Note that the associated defeated narrowing is the whole $[\Phi, x]$, even though each attacking a-structure defeats only a proper narrowing of $[\Phi, x]$.

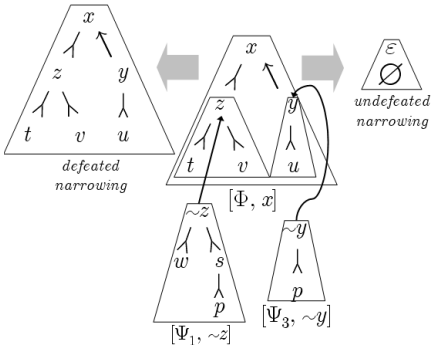


Figure 5: Combined Defeat

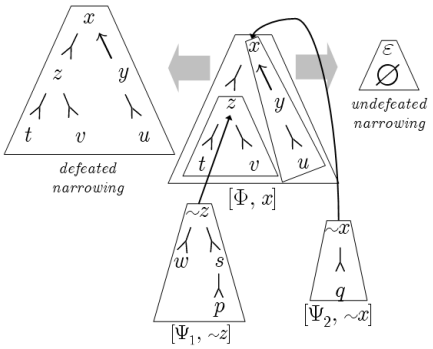


Figure 6: Combined Defeat

Consider now the combined attack against $[\Phi, x]$ shown in Fig. 6. One of the attacking a-structures ($[\Psi_1, \sim z]$) defeats a narrowing of $[\Phi, x]$ on its own, whereas the other ($[\Psi_2, \sim x]$) only attacks $[\Phi, x]$, as shown in Figs. 3 and 4, respectively (remember that we assumed that $[\Phi, x] \gg [\Psi_2, \sim x]$). But suppose that, although $[\Phi, x]$ is stronger than $[\Psi_2, \sim x]$ according to our criterion, $[\Psi_2, \sim x]$ is stronger than $[\Phi', x] = [\{y \prec u\}, x]$, a proper narrowing of $[\Phi, x]$. That is possible as, in general, a narrowing of an a-structure is weaker than the a-structure itself. Then, as shown in Fig. 6, when the a-structures $[\Psi_1, \sim z]$ and $[\Psi_2, \sim x]$ combine their attacks, they cause the whole $[\Phi, x]$ to become defeated. The reason is that the successful attack of $[\Psi_1, \sim z]$ weakens the target a-structure, allowing the attack of $[\Psi_2, \sim x]$ to succeed.

As illustrated by the previous case, in order to determine the defeated narrowing (and then the undefeated narrowing) associated with a combined attack we cannot just consider the attacks independently. Indeed, a given attacking a-structure could be successful in attacking another target a-structure only if the latter was previously "weakened" by some other attack. The purpose of the following definitions is to formally capture the notions of defeated and undefeated narrowings associated with a combined attack.

Definition 17 (Sequential Degradation)**.** *Let $[\Phi, h]$ be an a-structure and let Σ be a set of a-structures attacking $[\Phi, h]$. A Sequential Degradation of $[\Phi, h]$, associated with the combined attack of the a-structures in Σ, consists of a finite sequence of narrowings of $[\Phi, h]$:*

$$[\Phi_1, h],\ [\Phi_2, h],\ \ldots, [\Phi_{m+1}, h]$$

provided there exists a finite sequence of a-structures in Σ:

$$[\Psi_1, k_1],\ [\Psi_2, k_2],\ \ldots, [\Psi_m, k_m]$$

where $[\Phi_1, h] = [\Phi, h]$, for each i, $1 \leq i \leq m$, $[\Psi_i, k_i]$ partially defeats $[\Phi_i, h]$ with associated undefeated narrowing $[\Phi_{i+1}, h]$, and $[\Phi_{m+1}, h]$ has not defeaters in Σ.

Example 14. *Fig. 7 shows two sequential degradations for the combined attack of $[\Psi_1, \sim z]$, $[\Psi_2, \sim x]$ and $[\Psi_3, \sim y]$ against $[\Phi, x]$, namely $\mathcal{S}_1 = [\Phi, x]$, $[\Phi_2, x]$, $[\emptyset, \epsilon]$ and $\mathcal{S}_2 = [\Phi, x]$, $[\Phi_3, x]$, $[\emptyset, \epsilon]$. Fig. 8 shows the sequential degradation $\mathcal{S}_3 = [\Phi, x]$, $[\Phi_3, x]$ for the combined attack of $[\Psi_2, \sim x]$ and $[\Psi_3, \sim y]$ against $[\Phi, x]$, assuming that $[\Phi_3, x] \gg [\Psi_2, \sim x]$.*

As shown in Fig. 7, there could exist more than one sequential degradation associated with the same combined attack. However, not all sequential degradations associated with a given attack will necessary end with the same a-structure. This situation could arise when, according to the

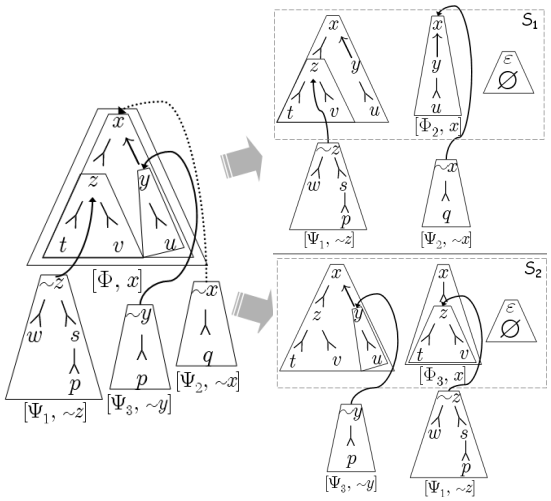


Figure 7: Sequential Degradations

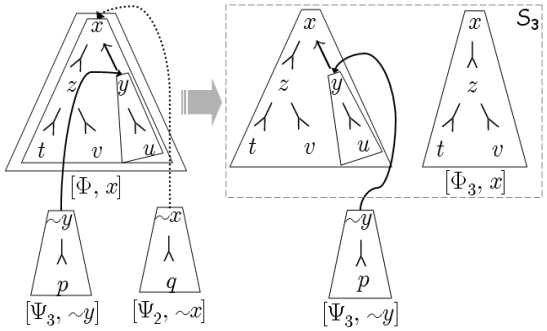


Figure 8: Sequential Degradation

preference relation, a given accrual is weaker than some of its narrowings. Consider again the sequential degradation in Fig. 8. Suppose that $[\Psi_2, \sim x] \gg [\Phi, x]$, but $[\{(x \prec z), (z \prec t), (z \prec v)\}, x] \gg [\Psi_2, \sim x]$. Then the sequence in Fig. 8 is indeed a sequential degradation associated with the attack, but there exists other sequential degradation that corresponds to applying first the defeat of $[\Psi_2, \sim x]$, immediately ending in $[\emptyset, \epsilon]$. Thus, if a particular a-structure turns out to be weaker than some of its narrowings, the order in which defeaters are introduced does matter. The solution is to restrict the order of defeater application so that the "deeper" defeats are applied first (and for the case recently discussed, the application order of Fig. 8 will be the right one). For the following definition, we will use the term *a-substructure defeater* to refer to a defeater which attacks the target a-structure at an intermediate conclusion, *i.e.*, that attacks an a-substructure of the target a-structure.

Definition 18 (Bottom-up sequential degradation)**.** *Let $[\Phi, h]$ be an a-structure and let Σ be a set of a-structures attacking $[\Phi, h]$. Let $\mathcal{S} = [\Phi_1, h],\ \ldots, [\Phi_{m+1}, h]$ be a sequential degradation of $[\Phi, h]$ associated with the combined attack of the a-structures in Σ, and let $[\Psi_1, k_1],\ \ldots, [\Psi_m, k_m]$ be the associated sequence of a-structures in Σ. Let $[\Lambda_i, k_i]$ be the disagreement a-substructure associated with the attack of $[\Psi_i, k_i]$ against $[\Phi_i, h]$, $1 \leq i \leq m$. Then we say that the sequential degradation $\mathcal{S}$ is bottom-up iff $[\Lambda_i, k_i]$ has no a-substructure defeater in Σ, $1 \leq i \leq m$.*

Thus, according to definition 18, the top most sequential degradation in Fig. 7 is not bottom-up. Note that the disagreement a-substructure associated with the attack of $[\Psi_2, \sim x]$ against $[\Phi_2, x]$, which is $[\Phi_2, x]$ itself, has $[\Psi_3, \sim y]$ as a-substructure defeater. The other sequential degradation in Fig. 7 is indeed bottom-up. Interestingly, it can be shown that all bottom-up sequential degradations associated with a given combined attack converge to the same a-structure.

Property 2. *Let $[\Phi, h]$ be an a-structure and let Σ be a set of a-structures attacking $[\Phi, h]$. Let $[\Phi_1, h], [\Phi_2, h], \ldots, [\Phi_m, h]$ and $[\Phi'_1, h], [\Phi'_2, h], \ldots, [\Phi'_n, h]$ be two bottom-up sequential degradations of $[\Phi, h]$ associated with the combined attack of the a-structures in Σ. Then $[\Phi_m, h] = [\Phi'_n, h]$.*

Definition 19 (Defeated and Undefeated Narrowings associated with a Combined Attack)**.** *Let $[\Phi, h]$ be an a-structure and let Σ be a set of a-structures attacking $[\Phi, h]$. Let $[\Phi_1, h]$, $[\Phi_2, h]$, ..., $[\Phi_{m+1}, h]$ be a bottom-up sequential degradation of $[\Phi, h]$ associated with the combined attack of the a-structures in Σ. Then $[\Phi_{m+1}, h]$ is the undefeated narrowing of $[\Phi, h]$ associated with the combined attack, and $[\Phi, h] - [\Phi_{m+1}, h]$ is its defeated narrowing.*

Example 15. *Consider the combined attack of $[\Psi_1, \sim z]$, $[\Psi_2, \sim x]$ and $[\Psi_3, \sim y]$ against $[\Phi, x]$. The associated undefeated narrowing of $[\Phi, x]$ is $[\emptyset, \epsilon]$, i.e., the whole $[\Phi, x]$ results defeated. On the other hand, when only $[\Psi_2, \sim y]$ and $[\Psi_3, \sim x]$ attack $[\Phi, x]$, its associated undefeated narrowing is $[\Phi_3, x] = [\{(x \prec z), (z \prec t), (z \prec v)\}, x]$.*

Dialectical Analysis for Accrued Structures

Given a DeLP program $\mathcal{P}$ and a maximal a-structure $[\Phi, h]$ we are interested in determining which is the final undefeated narrowing of $[\Phi, h]$ after considering all possible partial defeaters for $[\Phi, h]$. As every partial defeater is on its turn an a-structure, they can also have partial defeaters associated with them. This prompts a recursive dialectical analysis formalized as follows.

Definition 20 (Accrued Dialectical Tree)**.** *Let $[\Phi, h]$ be a maximal a-structure. The accrued dialectical tree for $[\Phi, h]$, denoted $\mathcal{T}_{[\Phi, h]}$, is defined as follows:*

1. *The root of the tree is labeled with $[\Phi, h]$.*
2. *Let N be an internal node labelled with $[\Theta, k]$. Let Σ be the set of all disagreement a-substructures associated with the attacks in the path from the root to N. Let $[\Theta_i, k_i]$ be an a-structure attacking $[\Theta, k]$ s.t. $[\Theta_i, k_i]$ has no a-substructures in Σ. Then the node N has a child node N_i labelled with $[\Theta_i, k_i]$. If there is no a-structure attacking $[\Theta, k]$ satisfying the above condition, then N is a leaf.*

The condition involving the set Σ avoids the introduction of a new a-structure as a child of a node N if it is already present in the path from the root to N (resulting in a circularity). This requirement is needed in order to avoid *fallacious* argumentative reasoning, as discussed in (García and Simari 2004).

Once the dialectical tree has been constructed, each combined attack is analyzed, from the deepest ones to the one against the root, in order to determine the undefeated narrowing of each node in the tree.

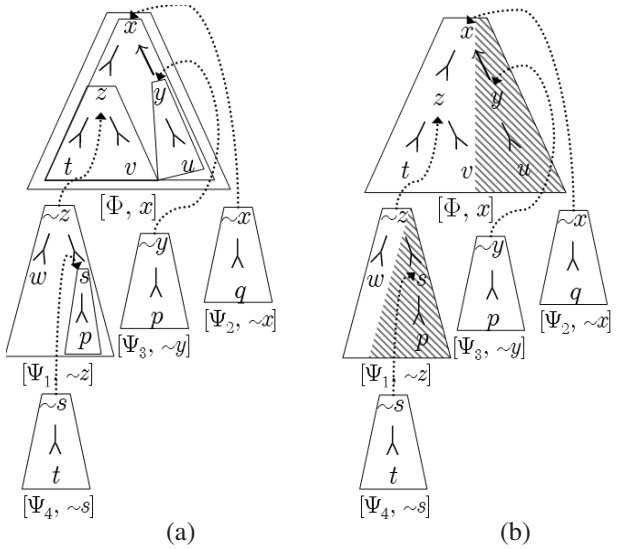


Figure 9: Dialectical Tree and Justification analysis.

Definition 21 (Undefeated narrowing of a Node)**.** *Let $\mathcal{T}_{[\Phi, h]}$ be an accrued dialectical tree for $[\Phi, h]$. Let N be a node of $\mathcal{T}_{[\Phi, h]}$ labelled with $[\Theta, k]$. Then the undefeated narrowing of N is defined as follows:*

1. *If N is a leaf node, then the undefeated narrowing of N is its own label $[\Theta, k]$.*
2. *Otherwise (i.e., if N is an internal node), let M_1, ..., M_n be the child nodes of N and let $[\Lambda_i, k]$ be the undefeated narrowing of the a-structure labelling the child node M_i, $1 \leq i \leq n$. Then the undefeated narrowing of N is the undefeated narrowing of $[\Theta, k]$ associated with the combined attack involving all the $[\Lambda_i, k]$, $1 \leq i \leq n$.*

Example 16. *Fig. 9a shows the dialectical tree for $[\Phi, x]$. Fig. 9b shows the dialectical tree for $[\Phi, x]$, where the undefeated narrowings of each node are highlighted. The preference relation is assumed the same as for sequential degradations in Figs. 7 and 8. Additionally, we assume that $[\Psi_4, \sim s] \gg [\{s \prec p\}, s]$, and thus $[\Psi_4, \sim s]$ defeats a narrowing of $[\Psi_1, \sim z]$. We also assume that although $[\Psi_1, \sim z]$ is preferred over $[\{(z \prec t), (z \prec v)\}, z]$, the undefeated narrowing of $[\Psi_1, \sim z]$ ($[\{\sim z \prec w\}, \sim z]$) is not, and thus its attack against $[\Phi, x]$ does not succeed.*

Definition 22 (Justified a-structure)**.** *Let $\mathcal{P}$ be a DeLP program and let h be a literal. Let $[\Phi, h]$ be a maximal a-structure for h such that the undefeated narrowing of $[\Phi, h]$ in $\mathcal{T}_{[\Phi, h]}$ is a non empty a-structure $[\Phi', h]$. Then we say that $[\Phi', h]$ is a justified a-structure for its conclusion h.*

According to the dialectical tree in Fig. 9b, $[\{(x \prec z), (z \prec t), (z \prec v), \}, x]$ is a justified a-structure for x.

The following property establishes that the a-structure emerging as a result of the above dialectical process cannot involve contradictory literals.

Property 3. *Let $\mathcal{P}$ be a DeLP program, and let $[\Phi, h]$ be a justified a-structure w.r.t. $\mathcal{P}$. Then there exist no intermediate conclusions k and r in $[\Phi, h]$ which are in disagreement.*

Related Work

There has been some previous research in argumentation concerning the treatment of accrual of reasons. In (Prakken 2005), Prakken enunciates three desirable principles that "any formal treatment of accrual should satisfy". The first principle says that *"accruals are sometimes weaker than their elements"* due to the possibility of accruing reasons are not independent. The second principle states that *"any 'larger' accrual that applies, makes all its 'lesser' versions inapplicable"*. Intuitively, that means that we should always accrue as many arguments as possible, even if in the end the accrual is outweighed by a conflicting accrual. The third principle states that *"flawed reasons or arguments may not accrue"*. That means that when an individual argument turns out to be flawed, it should not take part in the accrual.

Indeed, our framework satisfies all these principles. The first principle is satisfied since no assumption on the preference relation is made. Indeed, we introduced the notion of bottom-up sequential degradation in order to obtain a sound result when calculating the undefeated narrowing (associated with a given combined attack) of an accrual that is weaker than its elements. The second principle is trivially verified since the dialectical acceptance analysis only considers maximal a-structures. Finally, although the third principle is not verified in a strict sense, its underlying purpose is. That is, we first allow that all arguments accrue (in a maximal a-structure), and then we let the dialectical analysis (based on the notion of partial defeat) to rule out the flawed parts of the accrual. In the end, no flawed argument will be present in a justified a-structure.

In (Prakken 2005), Prakken also presents a formalization of accrual associated with the principles enunciated, that adapts the way of modeling accrual of reasons in Reason-Based Logic (Hage 1996) to an argument-based setting. This formalization is based on a combination of two widely recognized argument-based logics: Dung's abstract approach to argumentation (Dung 1995) instantiated with Pollock's approach to the structure of arguments (Pollock 1994). Prakken defines accrued arguments (or just accruals) as a special kind of defeasible derivations involving labels. Given a knowledge base (KB), consisting of a set of defeasible rules, defeasible derivations can be structured as described next. Defeasible rules in the knowledge base apply only to unlabeled premises, and when applied "produce" their conclusion labeled with such premises. Also, there exists a special accruing inference rule that from any set of labeled versions of a certain formula produces an unlabeled version. Then, the latter can in turn be used as a premise of another defeasible rule in the knowledge base, and so on. Consider the following KB:

$r_1 : b \Rightarrow a$ $\quad r_3 : c \Rightarrow b$ $\quad c$ $\quad f$
$r_2 : d \Rightarrow a$ $\quad r_4 : f \Rightarrow \sim b$ $\quad d$

Fig. 10 shows two derivations for a. The one on the left represents an accrual of two reasons for a, whereas the other represents the accrual of only one reason for a.

The attack relation between arguments is triggered by a particular notion of conflict between formulas. Two formulas are in conflict only if they are complementary and they are both unlabeled or have the same label. In this way, with

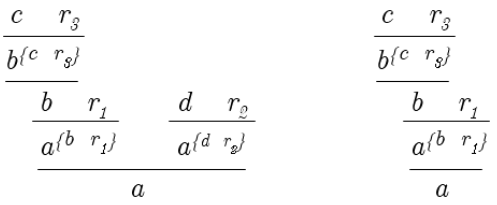


Figure 10: Prakken's accruals as labeled derivations

the labeling-unlabeling derivation and this particular notion of conflict, a reasoning process as in Reason-Based Logic is achieved, where each time a defeasible conclusion is produced, first all reasons for and against it are combined into two conflicting accruals, then the conflict between the two accruals is adjudicated, and only then the winning defeasible conclusion can be used in the rest of the reasoning process. Finally, a graph of attacks representing all the accruals and the attacks among them is constructed, which is then analyzed under the selected Dung's semantics in order to determine the status of accruals.

Unlike our formalization, Prakken's approach considers not only the maximal accruals in the status analysis, but all possible accruals for each conclusion. In other words, the graph of attacks will contain one node for each possible accrual derivable from the KB. Then, in order to satisfy the second principle of accrual a special 'construction' (Accrual undercutter) is introduced. The instances of this construction can also be represented as nodes in the graph of attacks, and they are used to state that when a given set of reasons for the same conclusion accrues, no proper subset accrues. Fig. 11 shows two derivations, A_1 and A_2, representing two different (individual) reasons for a, and a third derivation B representing a reason for $\sim b$. It also shows a part of the graph of attacks generated by the system from the KB (the part relevant to the conclusion a). In the graph there are three accruals for a, the one accruing only A_1, the one accruing only A_2 and the one accruing A_1 and A_2. There is also an instance of an accrual undercutter (depicted as an accrual but with conclusion '$*$') stating that if A_1 and A_2 successfully accrue (*i.e.*, neither A_1 nor A_2 is flawed), then all the 'lesser' accruals are undercut (note the arrows between the accrual undercutter and the lesser accruals). Finally, there is also an accrual for $\sim b$, involving only the derivation B. As the derivation A_1 contains b as intermediate conclusion, the accrual for $\sim b$ attacks each accrual involving A_1. For this graph, Dung's grounded extension coincides with the unique preferred extension, and is the set containing the accrual for $\sim b$ and the accrual for a involving only A_1. Note that the latter is the 'larger' accrual for a not containing flawed reasons.

In Verheij's Cumula system (Verheij 1996; 1995), accrual of reasons is carried out in two ways: through the notion of argument (where the conclusion or intermediate conclusions can be supported by more than one reason) and through the notion of compound defeat (which in its more general version allows to state that a certain set of arguments for the same conclusion defeats another set of arguments also for the same conclusion). In Cumula, the status of arguments is defined using a Dung-like semantics. According to this semantics, an argument is defeated if one of its subargu-

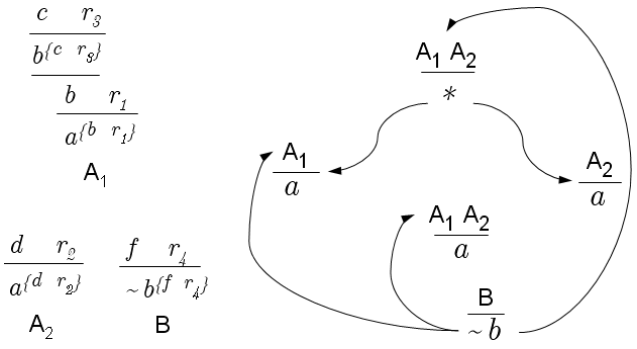


Figure 11: Graph of attacks

ments is defeated, and if a given argument is defeated, all its narrowings are also defeated. As analyzed by Prakken in (Prakken 2005), although Verheij's approach satisfies the three principles of accrual described above, the second one is satisfied in a way that is too strong. Because of the condition imposed by Verheij's semantics that if an accrual is defeated then all its narrowings are also defeated, it cannot be modeled a situation where an accrual is defeated because of subargument defeat so that some of its lesser versions (narrowings) can be undefeated.

Finally, it must be remarked that some defeasible logics (*e.g.*, Defeasible Logic (Governatori et al. 2004)) incorporate the notions of *team defeat*, which is in some respect similar to the notion of accrual.

Valuable features of our approach

Next we will summarize the most valuable features of our approach, contrasting it with Prakken's and Verheij's approaches.

An extension of DeLP to model Accrual

In first place, DeLP is of particular interest so that adding to it the capability of modeling accrual is significant in itself. DeLP has been applied in several real-world domains (Chesñevar, Maguitman, and Simari 2006), and so this new capability can be used to improve those existing applications.

There exists another advantage of using DeLP as a basis of our approach to model accrual. Several extensions of DeLP have already been developed, notably P-DeLP (Alsinet et al. 2008), a formalism that incorporates to DeLP the treatment of possibilistic uncertainty and fuzzy knowledge. This extension to DeLP can be applied almost directly to our approach for modeling accrual.

Finally, the argumentation mechanism proposed by DeLP is intended to answer queries from a given program, performing this task very efficiently. For instance, the analysis to determine if a given conclusion is justified (query) is formalized as a dialectical procedure, where only those attacking arguments which are relevant to the status of the analyzed conclusion are considered, leading to an optimized way of answering the query. Moreover, pruning strategies were defined for this dialectical procedure in order to improve efficiency. Our approach to model accrual inherits all these features from DeLP. On the other hand, the approaches of Prakken and Verheij define the status of arguments using Dung-like semantics. This semantics considers all the arguments in the system in order to state the set (or the alternative sets) of accepted arguments. Certainly, the query-driven feature of our approach is not a general advantage over the approaches of Prakken and Verheij, but can be valuable for achieving efficient argumentation in interactive real-world applications.

Efficiency of the formalization: a comparison with Prakken's approach

In the dialectical procedure proposed by our approach only maximal accruals are considered. On the other hand, in Prakken's approach, not only maximal accruals, but also all the lesser ones are considered. Then, if there exist n different individual reasons for a given conclusion x, Prakken's system will construct one accrual supporting x for each nonempty subset of this n reasons, which implies a number of accruals for x that is exponential on n (concretely, it is $2^n - 1$). Consider again the KB presented in the previous section and its associated graph of attacks (see Fig. 11). Note that as there exist two different reasons for a, A_1 and A_2, three accruals for a must be considered: the one involving only A_1, the one involving only A_2 and the one involving A_1 and A_2. In our formalization, the same answer is obtained after constructing and analyzing the small dialectical tree shown in Fig. 12a.

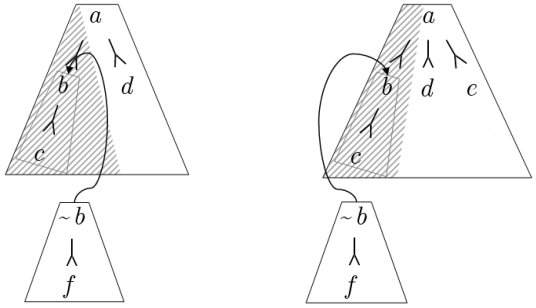


Figure 12: Accrued Dialectical Trees

This situation becomes more evident if (for instance) we add one more reason for a. Consider if we add the rule $r_5 : e \Rightarrow a$, together with the fact e. Fig. 13 shows the graph of attacks constructed by Prakken's system for the new KB. Note that as there exist three different reasons for a, then $7 = 2^3 - 1$ accruals for a are considered. Moreover, the number of accrual undercutters is also exponential on the number of individual reasons for the associated conclusion. Finally, the number of attacks is also considerable. Firstly, each accrual undercutter involving k reasons attacks all the accruals involving a proper (non empty) subset of these k reasons, which implies a number of attacks exponential on k (concretely, $2^k - 2$ attacks). Besides, a conflict with an individual reason implies one attack against each accrual involving this reason. Consider the case in Fig. 13. As the accrual for $\sim b$ contradicts the intermediate conclusion b in the derivation A_1, then the former attacks all the accruals involving A_1. In conclusion, the number of nodes and arrows in the graph of attacks grows exponentially with the num-

ber of individual accruing reasons, so that the computation of Dung's semantics becomes more complex. In contrast, in our formalization, the same answer is obtained after constructing and analyzing the small dialectical tree shown in Fig. 12b.

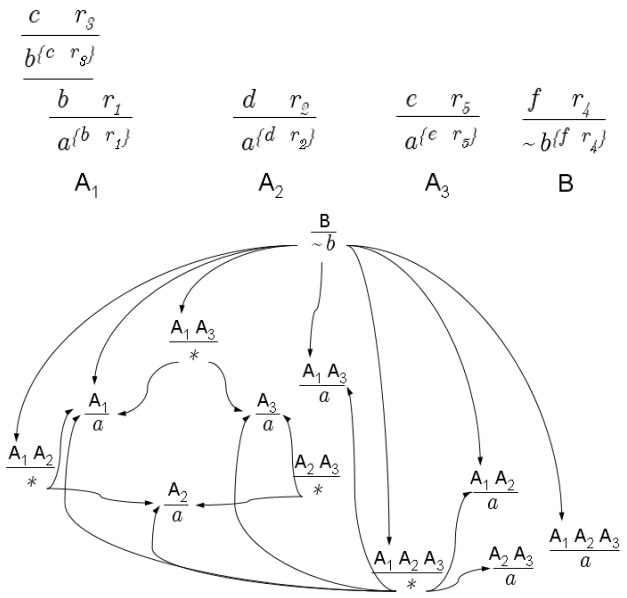


Figure 13: Graph of attacks

Explanations for justified conclusions

In the literature, an argument is often regarded as an explanation for a certain literal. In (García, Rotstein, and Simari 2007) a broader notion of explanation is proposed as providing the necessary information to understand the warrant status of a literal, helping to comprehend and analyze answers provided by argumentation systems based on dialectical proof procedures (as is the case for DeLP). This also applies as a potential feature in our proposal, not exhibited by other frameworks which formalize the status of arguments using Dung's semantics.

Conclusions

In this paper we have proposed a novel formalization to model the accrual of arguments based on the notion of *accrued structure*, which accounts for different arguments supporting a given conclusion. We have shown how accrued structures can be in conflict in terms of the notions of partial attack and defeat, from which defeated and undefeated narrowings can be identified. The notions of combined attack and sequential degradation were also defined, allowing us to characterize a dialectical process which has as an input a maximal a-structure $[\Phi, h]$ for a given conclusion, and gives as an output a justified a-structure (if any) which corresponds to a narrowing of $[\Phi, h]$. We have also shown that our formalization satisfies Prakken's principles for modeling accrual and we enunciated an interesting property (Prop. 3) of our approach which suggests an additional principle: *accrued structures which are ultimately accepted as justified should not involve conflicting arguments*. Finally, we described some valuable features of our approach.

It must be noted that there are several real-world problems in which accrual of arguments plays a major role (e.g. legal reasoning, social networks, etc.). Part of our current work involves representing those problems in terms of our formalism, analyzing the obtained results. In order to test the applicability of our proposal we are developing an implementation of our formalization using the DeLP system [3] as a basis. We are studying different theoretical results emerging from our proposal which could help to speed up the computation of accrued dialectical trees. Research in this direction is currently being pursued.

References

Alsinet, T.; Chesñevar, C. I.; Godo, L.; and Simari, G. R. 2008. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets Syst.* 159(10):1208–1228.

Besnard, P., and Hunter, A. 2001. A logic-based theory of deductive arguments. *Artif. Intell.* 128(1-2):203–235.

Chesñevar, C. I.; Maguitman, A. G.; and Simari, G. R. 2006. Argument-Based Critics and Recommenders: A Qualitative Perspective on User Support Systems. *Journal of Data and Knowledge Engineering* 59(2):293–319.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–358.

García, A., and Simari, G. 2004. Defeasible logic programming: An argumentative approach. *Theory Practice of Logic Programming* 4(1):95–138.

García, A. J.; Rotstein, N. D.; and Simari, G. R. 2007. Dialectical explanations in defeasible argumentation. In *ECSQARU*, 295–307.

Governatori, G.; Maher, M. J.; Antoniou, G.; and Billington, D. 2004. Argumentation semantics for defeasible logic. *J. Log. and Comput.* 14(5):675–702.

Hage, J. 1996. A theory of legal reasoning and a logic to match. *Artificial Intelligence and Law* 4(3–4):157–368.

Pollock, J. L. 1994. Justification and defeat. *Artificial Intelligence* 67(2):377–407.

Prakken, H., and Sartor, G. 1997. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics* 7(1):25–27.

Prakken, H. 2005. A study of accrual of arguments, with applications to evidential reasoning. In *ICAIL '05: Proceedings of the 10th international conference on Artificial intelligence and law*, 85–94. New York, NY, USA: ACM.

Verheij, B. 1995. Accrual of arguments in defeasible argumentation. In *Proceedings of the 2nd Dutch/German Workshop on Nonmonotonic Reasoning*, 217–224.

Verheij, B. 1996. *Rules, Reasons, Arguments: Formal studies of argumentation and defeat*. Doctoral dissertation, University of Maastricht.

[3] See http://lidia.cs.uns.edu.ar/delp

An Abstract Argumentation Framework for Handling Dynamics

Nicolás D. Rotstein and **Martín O. Moguillansky** and
Alejandro J. García and **Guillermo R. Simari**
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur – Bahía Blanca, Argentina
e-mail: {ndr,mom,ajg,grs}@cs.uns.edu.ar

Abstract

This article introduces the notion of dynamics into the concept of abstract argumentation frameworks, by including the concept of evidence to rule the validity of arguments when considering a particular situation. The proposed formalism is a refinement of Dung's abstract argumentation framework, to which many extensions have been defined. Our claim is that this idea could be enriched by the dynamic theory we are proposing. The main subject of this paper is the definition of the Dynamic Argumentation Framework, from which a static instance can be obtained. This instance is shown to be equivalent to the widely adopted Dung's framework. Therefore, multiple frameworks can be obtained from different instances of a given dynamic framework.

Introduction and Background

In this article we present a new abstract argumentation framework capable of dealing with dynamics through the consideration of a varying set of evidence. Certain configurations of the set of evidence will determine an instance of the framework in which some arguments hold and others do not. The extended formalization, which is coherent with the original abstractions, will provide the opportunity to tackle new problems and applications.

Frameworks for abstract argumentation have gained wide acceptance, and are the basis for the implementation of concrete formalisms (Dung 1995). The original proposal by Dung defines an abstract framework and several notions of acceptability of arguments. Since then, many extensions were introduced to enrich this approach, not only by defining new semantics (*i.e.*, different ways of accepting arguments) (Baroni and Giacomin 2007), but also by adding properties to the framework (Amgoud, Cayrol, and Lagasquie-Schiex 2004; Wyner and Bench-Capon 2007; Martínez, García, and Simari 2007) thus broadening the field of application of the original contribution –a survey about the applications of argumentation can be found in (Bench-Capon and Dunne 2007). Part of the argumentation community is moving in this direction, thus evolving the notion of abstract argumentation framework, which have proven to be suitable to model dialogues, negotiation processes and decision making mechanisms. The combination of argumentation and other disciplines is also under study, like belief revision (Rotstein et al. 2008). The spread of argumentation into the knowledge representation area is very promising, and this article aims to keep up with it by exploring a new line that could contribute to the expansion of the existing subareas. That is, the objective of this article is two-fold: it provides an extension of the existing theory and it also can be used as a base to enrich current abstract models.

In this article, we extend the classic theory of abstract argumentation (Chesñevar, Maguitman, and Loui 2000; Prakken and Vreeswijk 2000) to cope with the dynamics of evidence. The framework defined here is a refinement of Dung's, attempting to take a step forward into a not-so-abstract form of argumentation. In the literature, an argument is treated as an indivisible entity that suffices to support a claim, whereas here arguments are also indivisible, but they play a smaller role: they are aggregated in structures. These argumental structures can be thought as if they were arguments (in the usual sense), but they do not always guarantee their actual achievement of the claim. Moreover, in the literature arguments are often considered as completely abstract entities, with no regard about their composition. Here, we not only use arguments to generate structures, but we also explicitly mention a set of premises and a claim within both arguments and structures. The consideration of these features, *i.e.*, premises, inference and claims, has been part of the literature on logic, argumentation, and critical thinking from the early stages of the area (see (Toulmin 1959; Walton 1996) and more recently in (Chesñevar et al. 2006)).

The association of a set of premises to each argument leads us to the consideration of the role of evidence. We rely on the available evidence in order to determine whether arguments actually support their claim; this notion will be translated in terms of argumental structures later on. The study of the variation of the set of evidence and its impact on the status of arguments (*i.e.*, whether they can be used to make inferences) is one of the main contributions of this article, since it is the foundation of the dynamic framework. The other important contribution is the equivalence between Dung's framework and what we call a *static instance* of the DAF. We try to move forward in this direction, provided that "static" argumentation frameworks have been analyzed quite deeply in the last few years, since Dung's seminal work. Although the dynamic framework here proposed is enriched

with a number of features, we establish a relation that keeps us close to the results accomplished in the area.

The article is organized in the following way: the next section provides the theoretical elements (some of them adapted from their standard form) in order to define the dynamic framework; the second section gathers the theory previously defined and formalizes the new framework; afterwards, there is a section devoted to ongoing work regarding this research line; finally, the last section summarizes the results achieved in this paper and describes further extensions to the dynamic framework.

Preliminary Definitions

In this section we give the preliminary definitions from which the dynamic framework can be built. First, we define what an argument is, its relation with the set of evidence, and expected behavior. Then, we organize arguments in argumental structures, which provide arguments with a context when speaking of activeness (the actual availability of arguments to perform reasoning).

Argument

Arguments are pieces of reasoning that provide backing for a *claim* from a set of *premises*. These basic premises are considered as the argument's *support*. In the argumentation frameworks theory it is usually assumed that these premises (thus, the arguments they belong to) always hold, since frameworks show a snapshot of what is happening. However, as we are defining a dynamic system, it is natural to consider that some premises could be not satisfied. That is, we should account for what is not happening. Therefore, we must distinguish between what we call *active* and *inactive* arguments. In this article, arguments deemed as active will be those capable of actually achieving their claim. This will depend on whether the argument's premises are satisfied, *i.e.*, available either as *evidence* or claims of other active arguments. On this matter, a piece of evidence could be considered as a claim supported by an empty argument, or it could be treated separately, as a unique entity. In this article, we choose the latter option. Although we believe that the notion of evidence could be related to that of a claim, we also believe it represents a different concept. We could also consider a piece of evidence as a claim that needs no argument; evidence is there, beyond discussion. In contrast to the concept of active argument we introduce the definition of inactive argument as an argument that, in concordance with the current situation, is incapable of achieving its claim.

Given an argument $\mathcal{A}$, we will identify both its claim and support through the functions $\mathtt{cl}(\mathcal{A})$ and $\mathtt{supp}(\mathcal{A})$, respectively. In the same way, given a set $Args$ of arguments, we assume a function returning the set of all the claims in $Args$: $clset(Args) = \{\mathtt{cl}(\mathcal{A}) \mid \mathcal{A} \in Args\}$, and a function returning the set of all the premises in $Args$: $suppset(Args) = \bigcup_{\mathcal{A} \in Args} supp(\mathcal{A})$. Finally, since arguments are representing reasoning steps, we assume them as being *minimal* and *self-consistent*: no premise is the claim itself[1], and the combination of claim plus premises is non-contradictory (see Definition 3). In this article, we say that two sentences are contradictory or inconsistent if they cannot be assumed together.

Definition 1 (Set of Evidence) *A **set of evidence** is a consistent set of facts representing the current state of the world.*

Evidence is considered an indivisible and self-conclusive piece of knowledge that could come from perception, communication, or might be just agent's own knowledge (*e.g.*, its role). As stated before, evidence "triggers" some arguments, which we will call active. It is important to say that, throughout this article, when we refer to a set of evidence, we assume that it is consistent.

Definition 2 (Argument Support) *Given an argument $\mathcal{A}$ the **argument support** for $\mathcal{A}$ is a set of premises $\mathtt{supp}(\mathcal{A}) = \{s_1, \ldots, s_n\}$, where each premise s_i can be either evidence or a claim of another argument.*

In what follows, arguments will be noted as a pair, where its first element is the argument's set of premises, and the second one, its claim. For instance, if $\mathtt{supp}(\mathcal{A}) = \{a, b\}$ and $\mathtt{cl}(\mathcal{A}) = y$, we will note this argument as $\mathcal{A} = \langle\{a, b\}, y\rangle$. Once the support and claim of an argument are clear (or when they are irrelevant), arguments will be called just by their name.

Definition 3 (Argument) *An **argument** $\mathcal{A}$ is a pair $\langle\{s_1, \ldots, s_n\}, \alpha\rangle$, verifying:*

- *$\{s_1, \ldots, s_n, \alpha\}$ is consistent;*
- *there is no $s_i \in \mathtt{supp}(\mathcal{A})$ such that $s_i = \alpha$.*

Example 1 *Assume an argument $\mathcal{A}$ for considering a route as being dangerous because there are known thieves in that area and the security there is poor. Then, we have that $\mathtt{supp}(\mathcal{A}) = \{th, ps\}$ and $\mathtt{cl}(\mathcal{A}) = dr$. Consider also an argument $\mathcal{B}$ saying that underpaid cops might provide poor security; then $\mathtt{supp}(\mathcal{B}) = \{upc\}$ and $\mathtt{cl}(\mathcal{B}) = ps$. These two arguments are depicted as triangles in Figure 1, each with its corresponding set of premises on its base, and the claim on top.*

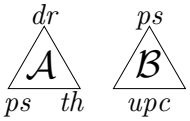


Figure 1: Arguments for dr and ps.

From the definition of argument support, it is clear that a premise of an argument could be "instantiated" in many ways; that is, by evidence or another argument's claim. The latter case gives rise to a larger, more complex structure comprising other arguments. This set of arguments will be called an *argumental structure*, as described in Definition 7.

The active/inactive status of an argument might involve other arguments: sometimes it is not evidence what will be

[1] This is not the usual restriction for minimality, but it is as far as we can go when KR is made through abstract arguments.

directly *activating* arguments, but *supporting arguments* – which are arguments achieving a part of the support of others. That is, an argument could be *activated* by other arguments, rather than by evidence; they, in turn, are to be activated by evidence or by their own supporting arguments, and so on, until the last argument is based solely on evidence.

Definition 4 (Supporting Argument) *An argument $\mathcal{B}$ is a **supporting argument** of an argument $\mathcal{A}$ iff $\mathtt{cl}(\mathcal{B}) \in \mathtt{supp}(\mathcal{A})$. Let $\mathtt{cl}(\mathcal{B}) = s$, then we say that $\mathcal{B}$ supports $\mathcal{A}$ through s.*

The support of an argument $\mathcal{A}$ could lack some pieces of evidence, but other arguments could provide their claims as if they were evidence, so $\mathcal{A}$ should be considered active. However, there will be conditions for an argument to be considered *coherent*, thus preventing some arguments from becoming active. This will be clear next, with the (recursive) definition for an active argument, but before we introduce the notion of *coherent argument*.

Definition 5 (Coherent Argument) *An argument $\mathcal{A}$ is **coherent** wrt. a set $\mathbf{E}$ of evidence iff $\mathcal{A}$ verifies the following properties:*

- ***(Consistency wrt. $\mathbf{E}$)** An argument $\mathcal{A}$ is **consistent** wrt. $\mathbf{E}$ iff $\mathtt{cl}(\mathcal{A})$ does not contradict any evidence in $\mathbf{E}$.*
- ***(Non-Redundancy wrt. $\mathbf{E}$)** An argument $\mathcal{A}$ is **non-redundant** wrt. $\mathbf{E}$ iff $\mathtt{cl}(\mathcal{A}) \notin \mathbf{E}$.*

Redundant arguments wrt. evidence are not harmful, they just introduce new information that is not going to be useful –since evidence is beyond discussion and needs no reasons supporting it. In opposition, inconsistent arguments wrt. evidence may be harmful, since they could be used to activate other arguments, rendering invalid all that reasoning chain, thus requiring further restrictions in order to allow the construction of valid reasoning chains.

From now on, we will assume that any given argument is coherent wrt. the set of evidence corresponding to the context the argument is immersed into, unless stated otherwise.

Definition 6 (Active Argument) *Given a set $Args$ of arguments, a set $\mathbf{E}$ of evidence, a coherent argument $\mathcal{A} \in Args$ is **active** wrt. $\mathbf{E}$ iff for each $s \in \mathtt{supp}(\mathcal{A})$ either:*

- *$s \in \mathbf{E}$, or*
- *there is an active argument $\mathcal{B} \in Args$ that supports $\mathcal{A}$ through s.*

Example 2 *Consider Example 1. If the set of evidence is $\mathbf{E}_2 = \{th, upc\}$, then $\mathcal{A}$ is active, because it can be activated by the evidence 'th' and the active (supporting) argument $\mathcal{B}$ for 'ps'.*

Example 3 *From Example 2, if we consider a set of evidence $\mathbf{E}_{ps} = \{poor\_security\} \cup \mathbf{E}_2$, then argument $\mathcal{B}$ would be incoherent due to its redundancy wrt. $\mathbf{E}_{ps}$. In contrast, if we consider the set of evidence $\mathbf{E}_{nps} = \{\neg poor\_security\} \cup \mathbf{E}_2$, then argument $\mathcal{B}$ should also be incoherent, because it would be inconsistent wrt. $\mathbf{E}_{nps}$. In both cases $\mathcal{B}$ would not be active because is incoherent.*

Regarding $\mathcal{A}$, from the set E_{ps}, it becomes active directly from evidence, whereas from the set E_{nps}, it ends up being inactive since its premises are left unsupported: although $\mathcal{B}$ achieves 'ps', it is not compliant with the consistency argument constraint.

From what we have defined, an argument could be inactive because: it might not have enough evidence and/or active arguments to support it, and/or it might not comply with at least one constraint. In both cases an inactive argument fails in being a support for reaching its associated claim.

Finally, regarding attacks between arguments, there is a normality condition to be taken into account: for a given set of arguments, it must contain every pair of arguments holding claims in contradiction.

Assumption 1 *Let $Args$ be a set of arguments and $\mathbf{R} \subseteq Args \times Args$, an attack relation. Given two arguments $\{\mathcal{A}, \mathcal{B}\} \subseteq Args$, $\mathcal{A} = \langle \{\cdot\}, c_1 \rangle$, $\mathcal{B} = \langle \{\cdot\}, c_2 \rangle$, if c_1 and c_2 are in contradiction, then $\mathcal{A}\mathbf{R}\mathcal{B}$ or $\mathcal{B}\mathbf{R}\mathcal{A}$.*

Argumental Structures

The aggregation of arguments via the support relation needs further formalization, giving rise to the concept of *argumental structure*. Next, we will introduce this core element.

Definition 7 (Argumental Structure) *Given a set $Args$ of arguments, an **argumental structure** for a claim α from a set of arguments $\Sigma^* \subseteq Args$ is a tree of arguments Σ verifying:*

1. *The root argument $\mathcal{A}_{top} \in \Sigma^*$, called **top argument**, is such that $\mathtt{cl}(\mathcal{A}_{top}) = \alpha$;*
2. *An inner node is an argument $\mathcal{A}_i \in \Sigma^*$ such that for each of its premises $\beta \in \mathtt{supp}(\mathcal{A}_i)$ there is at most one child argument $\mathcal{A}_k \in \Sigma^*$ supporting $\mathcal{A}_i$ through β.*
3. *A leaf is an argument $\mathcal{A}_k \in \Sigma^*$ such that there is no argument $\mathcal{A}_k \in \Sigma^*$ supporting it.*

Regarding notation for an argumental structure Σ:

- *The support of Σ is defined as:*

  $\mathtt{supp}(\Sigma) = suppset(\bigcup_k(\mathcal{A}_k))$, *for every leaf* $\mathcal{A}_k \in \Sigma$
- *The claim of Σ is noted as $\mathtt{cl}(\Sigma) = \alpha$.*
- *The set of arguments belonging to Σ is noted as Σ^*.*

Note that the $\mathtt{supp}(\cdot)$ and $\mathtt{cl}(\cdot)$ functions are overloaded: now they are applied to argumental structures. This is not going to be problematic, since either usage will be rather explicit. From now on, when clear enough, we will refer to argumental structures just as 'structures'.

Example 4 *From Example 2 we have the argumental structure Σ_4, such that $\Sigma_4^* = \{\mathcal{A}, \mathcal{B}\}$ (illustrated in Figure 2), where its support is $\mathtt{supp}(\Sigma_4) = \{th, upc\}$ and its claim is $\mathtt{cl}(\Sigma_4) = \mathtt{cl}(\mathcal{A}) = dr$. Note that the support of Σ_4 is different from the set of all premises in it: $suppset(\Sigma_4) = \{upc, ps, th\}$; finally, its set of claims is $clset(\Sigma_4) = \{ps, dr\}$.*

However, the definition for an argumental structure is not enough to represent knowledge in a sensible way. For instance, it allows for contradictory claims in a pair of arguments belonging to the same structure. Therefore, we have to define what is considered a *well-formed argumental structure*, but in order to do this we need the definition for *transitive support*.

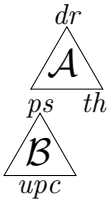


Figure 2: Argumental structure for $dangerous\_route$ (dr).

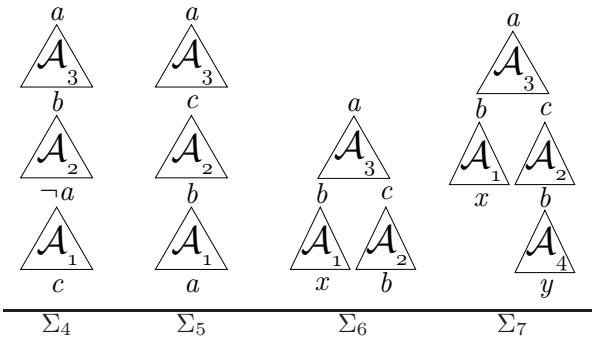
Definition 8 (Transitive Support) *An argument $\mathcal{A}_i$ **transitively supports** an argument $\mathcal{A}_j$ iff there is a sequence of arguments $[\mathcal{B}_1, \ldots, \mathcal{B}_n]$ where $\mathtt{cl}(\mathcal{A}_i) \in \mathtt{supp}(\mathcal{B}_1)$, $\mathtt{cl}(\mathcal{B}_n) \in \mathtt{supp}(\mathcal{A}_j)$ and $\mathtt{cl}(\mathcal{B}_k) \in \mathtt{supp}(\mathcal{B}_{k+1})$, with $1 \leq k \leq n-1$.*

Definition 9 (Well-Formed Argumental Structure) *An argumental structure Σ is **well-formed** iff Σ verifies the following properties:*

- ***(Consistency)** For each argument $\mathcal{A}_i \in \Sigma^*$ there is no argument $\mathcal{A}_k \in \Sigma^*$ $(i \neq k)$ such that $\mathcal{A}_i \mathbf{R} \mathcal{A}_k$;*
- ***(Non-Circularity)** No argument $\mathcal{A}_i \in \Sigma^*$ transitively supports an argument $\mathcal{A}_k \in \Sigma^*$ if $\mathtt{cl}(\mathcal{A}_k) \in \mathtt{supp}(\mathcal{A}_i)$.*
- ***(Uniformity)** If a premise $\beta \in suppset(\Sigma)$ is supported by an argument $\mathcal{B} \in \Sigma^*$, then for every $\mathcal{A}_i \in \Sigma^*$ having β as a premise, $\mathcal{B}$ supports $\mathcal{A}_i$ through β.*

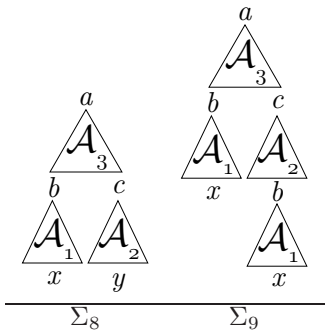
The property of *consistency* invalidates inherently contradictory argumental structures. The requirement of *non-circularity* avoids taking into consideration structures yielding a fallacious reasoning chain, where an argument ends up being transitively supported by itself. Finally, the restriction of *uniformity* refrains non-minimal structures to be deemed as well-formed, it does not allow heterogeneous support for a premise throughout a structure. These constraints are defined so we can trust a well-formed structure as a sensible reasoning chain, independently from the set of evidence. The consideration of a set of evidence is part of the notion of *active argument*, which is addressed in Definition 11.

Example 5 *The following sets of arguments are argumental structures, but they are not well-formed* [2]*:*



- Σ_4 *violates the* consistency *property, due to $\mathcal{A}_1$ and $\mathcal{A}_3$ achieving contradictory claims.*
- Σ_5 *violates the* non-circularity *property, since $\mathcal{A}_1 = \langle \{a\}, b \rangle$ transitively supports $\mathcal{A}_3 = \langle \{c\}, a \rangle$.*
- Σ_6 *and* Σ_7 *violate the* uniformity *property:*
  - *in Σ_6, the premise 'b' has two occurrences, but is supported by $\mathcal{A}_1$ in one case, and left unsupported in the other.*
  - *in Σ_7, the premise 'b' is supported by two different arguments.*

Example 6 *The following sets of arguments do compose well-formed argumental structures:*



- *Structure Σ_8 presents no controversy, is a simple argumental structure.*
- *Structure Σ_9 shows a case similar to that of Σ_8. However, premise 'b' is always supported by the same argument ($\mathcal{A}_1$) therefore verifying the* uniformity *property.*

From now on, we will assume that any given argumental structure is well-formed, unless stated otherwise.

Definition 10 *Let $\Sigma^* = \{\mathcal{A}\}$ be the set of arguments of the argumental structure Σ, then Σ is called a **primitive argumental structure** and $\mathtt{supp}(\Sigma) = \mathtt{supp}(\mathcal{A})$, $\mathtt{cl}(\Sigma) = \mathtt{cl}(\mathcal{A})$.*

The concept of primitive argumental structures shows that an argument can be seen as a particular case of an argumental structure.

Now that we have defined what a well-formed argumental structure is, we can introduce the notion of *active argumental structures*. This will allow us to recognize those structures that are capable of achieving their claims when considering the current situation.

Definition 11 (Active Argumental Structure) *Given a set* **E** *of evidence, a well-formed argumental structure Σ is **active** wrt.* **E** *iff $\mathtt{supp}(\Sigma) \subseteq$* **E** *and every $\mathcal{A} \in \Sigma^*$ is a coherent argument wrt.* **E**.

This definition states an important property: the premises of an active argumental structure is composed just by evidence. This puts this concept nearer to the notion of active argument, showing that argumental structures can be seen as arguments in the usual way if their inner composition is abstracted away. The definition also requires every argument to be coherent wrt. the set of evidence, therefore some well-formed structures having their support satisfied by evidence will not be active due to some argument being redundant and/or inconsistent wrt. the evidence. Note that coherence

[2] Here, we assume a propositional language for claims and premises, where strong negation indicates contradiction.

cannot be put as a requirement for a well-formed structure, since it is tied to a particular set of evidence.

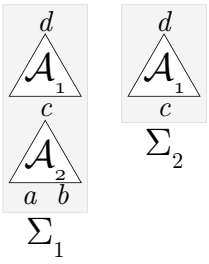
Example 7 *(Extends Example 5) Consider* $\mathbf{E}_7 = \{b, x, y\}$ *to be a set of evidence, then structure* Σ_8 *is not an active structure, because* $\mathcal{A}_1$ *is redundant wrt.* $\mathbf{E}_7$.

There is a subtle relation between active arguments and active argumental structures that will be made explicit by the following propositions.

Proposition 1 *If* Σ *is an active argumental structure wrt. a set* $\mathbf{E}$ *of evidence, then every sub-argument of* Σ *is an active argument wrt.* $\mathbf{E}$.
Proof: By construction of an active structure Σ *wrt.* $\mathbf{E}$*, let* $\mathcal{A}_{top}$ *be its active (thus coherent) top argument. Therefore, either* $\mathsf{supp}(\mathcal{A}_{top}) \subseteq \mathbf{E}$*, or some premise* $\beta \in \mathsf{supp}(\mathcal{A}_{top})$ *is supported by an active argument* $\mathcal{A}_i$*. In the latter case,* $\mathcal{A}_i$ *is included into* Σ*, and then the same analysis is made regarding the premises of* $\mathcal{A}_i$*. The construction of the tree of arguments is performed recursively, and in each step an active argument is included into* Σ*. Therefore,* Σ *contains only active arguments wrt.* $\mathbf{E}$ □

The reverse of the latter proposition is not true, as shown in the following example.

Example 8 *Consider a set of evidence* $\mathbf{E}_8 = \{a, b\}$*, and two structures* Σ_1 *and* Σ_2 *such that* $\Sigma_1^* = \{\mathcal{A}_1, \mathcal{A}_2\}$ *and* $\Sigma_2^* = \{\mathcal{A}_1\}$*, where* $\mathcal{A}_1 = \langle\{c\}, d\rangle$*,* $\mathcal{A}_2 = \langle\{a, b\}, c\rangle$*, as depicted below.*



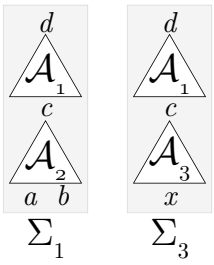
From $\mathbf{E}_8$*, structure* Σ_1 *is active, but* Σ_2 *is not. Clearly, both argumental structures contain active arguments, but this condition does not ensure them to be active.*

Example 8 shows that, in a way, argumental structures have to be 'complete' in order for them to be active. That is, they must include all of the necessary arguments for their top argument to be active. Only then the support of these structures will be composed by evidence. This statement is made clear by the results of the following two propositions.

Proposition 2 *Every active argument wrt. a set* $\mathbf{E}$ *of evidence is the top argument of at least one active argumental structure wrt.* $\mathbf{E}$.
Proof: Let assume that there is an argument $\mathcal{A}$ *active wrt.* $\mathbf{E}$ *that is not the top argument of any argumental structure active wrt.* $\mathbf{E}$*. Since* $\mathcal{A}$ *is active, either (1) its premises are a subset of* $\mathbf{E}$*, or (2) some premise is supported by an active argument* $\mathcal{B}$*. Case (1) clashes with the assumption of* $\mathcal{A}$ *not being the top argument of any active structure, since the primitive structure composed just by* $\mathcal{A}$ *would be active wrt.* $\mathbf{E}$*. Case (2) indicates that a structure could be built containing at least* $\mathcal{A}$ *and* $\mathcal{B}$*, and then the analysis made over* $\mathcal{A}$ *is also applicable to* $\mathcal{B}$*, because it should also be active wrt.* $\mathbf{E}$*, in order for* $\mathcal{A}$ *to be active. Hence, the recursive consideration of new arguments while including them into a new hypothetical structure ends when all the premises of the last argument is a subset of* $\mathbf{E}$*. This would mean that the hypothetical structure is active wrt.* $\mathbf{E}$*, leading to absurdity. This is due to the assumption of* $\mathcal{A}$ *not being the top argument of any argumental structure active wrt.* $\mathbf{E}$ □

Note that Proposition 2 allows for an active argument to be top argument of more than one active argumental structure, which is correct, as depicted in the following example.

Example 9 *Consider* Σ_1 *from Example 8, an argument* $\mathcal{A}_3 = \langle\{x\}, c\rangle$*, and a set of evidence* $\mathbf{E}_9 = \{a, b, x\}$*. Then, the following active argumental structures can be built:*



Note that both are well-formed active argumental structures wrt. $\mathbf{E}_9$ *and have the same top argument.*

Proposition 3 *Given an active argumental structure* Σ *wrt. a set* $\mathbf{E}$ *of evidence, there is no proper substructure* Σ_i *of* Σ *such that* Σ_i *is active wrt.* $\mathbf{E}$.
Proof: Let assume that Σ_i *is an active structure wrt.* $\mathbf{E}$*. Then* $\mathsf{supp}(\Sigma_i) \subseteq \mathbf{E}$*. Now assume that* Σ *is also active wrt.* $\mathbf{E}$*. Therefore, there is at least one premise of* Σ_i *that, in* Σ*, is supported by an argument* $\mathcal{B}$*. Consequently, one of the following holds: (1)* $\mathcal{B}$ *is redundant wrt.* $\mathbf{E}$ *and* Σ *is not an active structure wrt.* $\mathbf{E}$*; (2)* Σ_i *has a premise that is not evidence. Either of these cases leads to absurdity. This is due to the assumption of* Σ_i *being an active proper substructure of* Σ □

The Dynamic Argumentation Framework

Now that we have defined the main components of our theory, we will put them together in the definition of the *dynamic argumentation framework*. In the literature, argumentation frameworks are usually static, in the sense that every argument in them participates in the argumentative interplay, without regard to the actual validity of the arguments in the current situation. This is so because they do not consider such a thing as a possibly changing situation, but instead are restricted to a single snapshot.

Almost every new approach to abstract argumentation is built on top of Dung's argumentation framework (Dung 1995) (from now on, simply 'AF'). This framework is defined as a pair with a set of arguments and a defeat relation ranging over pairs of them. The objective of our approach is to extend this theory to handle dynamics. To cope with

this we consider a set of available evidence, which determines what arguments can be used to make inferences. If we follow Dung's approach, the consideration of a changing set of arguments would involve passing from a framework to another, but this cannot be performed lightly: what is the relation between these frameworks? Where do the new arguments come from and where do the old ones go to? Furthermore, if we are incorporating a set of evidence that activates arguments: how does the set of evidence change? How does the set of evidence affect the status of arguments? These questions have to be properly addressed in order to build a coherent dynamic framework. Therefore, next we define the notion of attack between structures, and then, we introduce the dynamic argumentation framework.

The notion of argumental substructure allows us to redefine attacks, now in terms of structures. Before this definition, we introduce the notion of *argumental substructure*.

Definition 12 (Argumental Substructure) *Given an argumental structure Σ from a set of arguments $Args$, the set Σ_i is an **argumental substructure** of Σ iff $\Sigma_i^* \subseteq \Sigma^*$ and Σ_i is an argumental structure from $Args$. If $\Sigma_i^* \subsetneq \Sigma^*$ then Σ_i is a **proper argumental substructure** of Σ.*

Note that an argumental structure is an argumental substructure of itself. As with the former, we will refer to the latter just as a 'substructure', when convenient.

Definition 13 (Attack Between Argumental Structures) *Given a set $Args$ of arguments, an attack relation $\mathbf{R} \subseteq Args \times Args$ between arguments, and two argumental structures Σ_1 and Σ_2 from $Args$, the structure Σ_1 **attacks** Σ_2 iff there is a substructure Σ_2' of Σ_2 such that $top(\Sigma_1)\mathbf{R}top(\Sigma_2')$.*

The attack relation between arguments is composed of pairs of arguments; that is, given two arguments $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A}\mathbf{R}\mathcal{B}$ ($(\mathcal{A},\mathcal{B}) \in \mathbf{R}$), then we have that $\mathcal{A}$ attacks (or defeats) $\mathcal{B}$. Equivalently, we will say that $\mathcal{A}$ is a defeater for $\mathcal{B}$. When speaking of argumental structures, we use the same vocabulary.

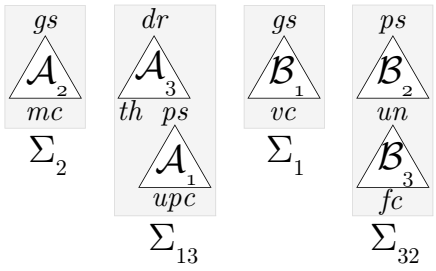
Remark 1 *Given an argument $\mathcal{A}$ from an active argumental structure Σ, if $\mathcal{A}$ is defeated by an active argument $\mathcal{B}$, then Σ is defeated by an argumental structure whose top argument is $\mathcal{B}$.*

The statement made by this remark was referred as *conflict inheritance* in (Martínez, García, and Simari 2007).

Definition 14 (Dynamic Argumentation Framework (DAF)) *A **DAF** is a pair $\langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$, composed by a set $\mathbf{E}$ of **evidence**, and a framework $(\mathbf{U}, \mathbf{R})$, where $\mathbf{U}$ is the **universal set of arguments** and $\mathbf{R} \subseteq \mathbf{U} \times \mathbf{U}$ is the **attack relation** between arguments.*

Different instances of the set of evidence determine different instances of the DAF. Thus, when "restricting" a framework $(\mathbf{U}, \mathbf{R})$ to its associated set of evidence, we can obtain a (static) framework in the classical sense, *i.e.*, a pair in which every argument is active, and the attack relation contains pairs of them. This "restriction" will be called a *static instance*, and is addressed below.

Example 10 *Consider the argumental structure of Example 4, in which knowing that there are thieves in a place and that cops there are underpaid leads us to think that that route is going to be dangerous. Let us assume that there are many cops (noted as 'mc') in the location, therefore we have a reason to think that security there is good ('gs'). Another argument leading us to think of good security is that the cops could be volunteer ('vc'), thus more motivated to do a good job. Nonetheless, if cops are foreigners ('fc'), then they are probably not acquainted with the place (un), and that could give the idea of poor security there ('ps'). Then, we can build the following argumental structures:*



Thus, we have a DAF $\langle \mathbf{E}_{10}, (\mathbf{U}_{10}, \mathbf{R}_{10})\rangle$, where the universal set of arguments is $\mathbf{U}_{10} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}$, and we will consider a set of evidence $\mathbf{E}_{10} = \{many\_cops, underpaid\_cops, thieves\}$, along with an empty attack relation $\mathbf{R}_{10} = \emptyset$. Then, from set $\mathbf{U}_{10}$, arguments $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are active wrt. $\mathbf{E}_{10}$, thus reaching their claims $good\_security$, $poor\_security$ and $dangerous\_route$. The latter claim is achieved via the argumental structure Σ_{13}, whose top argument is $\mathcal{A}_3$. The remaining arguments $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are inactive, as well as structures Σ_1 and Σ_{32}, since they have unfulfilled supports wrt. $\mathbf{E}_{10}$ and thus cannot reach their claims.

A subset of the universal set will be considered as the *set of active arguments* wrt. the set of evidence. This set will contain those arguments that are to be taken into account to perform reasoning in concordance with the current situation.

Definition 15 (Set of Active Arguments) *Given a DAF $F = \langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$, the **set of active arguments** in F is $\mathbb{A} = \{\mathcal{A} \in \mathbf{U} \mid \mathcal{A}$ is active wrt. $\mathbf{E}\}$.*

Given the universal set $\mathbf{U}$ of arguments and the set $\mathbb{A}$ of active arguments we can derive *the set of inactive arguments* as $\mathbf{U} \setminus \mathbb{A}$. This latter set would be very useful when reasoning about possible worlds, potential situations, or even goals and the plausibility of reaching them. Moreover, since the attack relation is given over the universal set, there will be active and inactive attacks. The latter relation (involving at least one inactive argument) would allow us to, say, activate defeaters for currently active arguments. These concepts can be translated into terms of argumental structures.

Definition 16 (Set of Active Argumental Structures) *Given a DAT T and the set $\mathbb{A}$ of active arguments in T, the **set of active argumental structures** in T is the maximal set $\mathbb{S}$ of argumental structures from $\mathbb{A}$.*

Proposition 4 *Given a set $\mathbb{A}$ of active arguments and a set $\mathbb{S}$ of active argumental structures from $\mathbb{A}$, then $\bigcup_{\Sigma^* \in \mathbb{S}}(\Sigma^*) = \mathbb{A}$. Proof: Trivial from Definition 16.*

As in the case of arguments, the set of active structures allows us to distinguish the set $\mathbb{S}^{\mathbb{I}}$ of inactive argumental structures, as a set of arguments composing an argumental structure, but containing at least one argument that is not active.

Example 11 *From Example 10, the set of active argumental structures is: $\mathbb{S}_{11} = \{\Sigma_{13}, \Sigma_2\}$. An inactive argumental structure would be Σ_{32}.*

Definition 17 (Inactive/Active Attacks) *Given a DAF $\langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$ and the set of active (inactive) argumental structures $\mathbb{S}$ ($\mathbb{S}^{\mathbb{I}}$) from $\mathbf{U}$ wrt. $\mathbf{E}$, we have:*

- *The **active attack relation**:* $\mathbb{R} = \{(\Sigma_1, \Sigma_2) \mid \{\Sigma_1, \Sigma_2\} \subseteq \mathbb{S}, \Sigma_1 \textit{ attacks } \Sigma_2\}$
- *The **inactive attack relation**:* $\mathbb{R}^{\mathbb{I}} = \{(\Sigma_1, \Sigma_2) \mid \Sigma_1 \in \mathbb{S}^{\mathbb{I}} \textit{ or } \Sigma_2 \in \mathbb{S}^{\mathbb{I}}, \Sigma_1 \textit{ attacks } \Sigma_2\}$

Static Instance of a DAF

Next, we define the *static instance* of a given DAF, which we will show that is equivalent to a an AF.

Definition 18 (Static Instance) *Given a DAF $\langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$, the **static instance** of $\langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$ is the AF $(\mathbb{S}, \mathbb{R})$, where $\mathbb{S}$ is the set of active argumental structures from $\mathbf{U}$ wrt. $\mathbf{E}$, and $\mathbb{R}$ is the active attack relation between structures in $\mathbb{S}$. The notation is $(\mathbf{U}, \mathbf{R})|_{\mathbf{E}} = (\mathbb{S}, \mathbb{R})$.*

Every DAF, at any moment, has an associated static instance, which is an AF. Therefore, all the work done on acceptability of arguments and argumentation frameworks semantics can be applied to the DAF here defined, just by finding its static instance. Moreover, since we added some structure to the notion of argument, we can go a step further and consider justification of claims, either in a skeptical or a cautious way.

DAFs can be seen as a template for generating multiple AFs representing the same knowledge applied to different situations. The number of static instances that can be obtained from a single DAF is quite large. Provided that each argument has at least one premise (*i.e.*, a possible evidence), then the amount of possible evidence equals or exceeds the amount of arguments in the universal set. Let $\mathbb{E}$ be this set of possible evidence, then if we consider a universal set $\mathbf{U}$ of arguments we have: $|\mathbb{E}| \geq |\mathbf{U}|$. Considering that each possible subset of evidence composes a different static instance of the DAF, we have that the amount of static instances is in the order of $2^{\mathbb{E}}$. Finally, it is now clear that there is a large number of AFs associated to a single DAF.

Updating Evidence in a DAF

Since the set of evidence is dynamic, it defines the particular instance of the DAF that corresponds with the current situation. In order to cope with this, the basic operation performed over a DAF is the *evidence update*. This mechanism should ensure the DAF reflects the new (consistent) state of the world. To ease the legibility of the next definition, we use the complement notation to indicate contradiction between pieces of evidence.

Definition 19 (Evidence Update (resp., Erasure)) *Given a DAF $\langle \mathbf{E}, (\mathbf{U}, \mathbf{R})\rangle$, and $\mathbf{E}_1$, a set of evidence such that for every $\beta \in \mathbf{E}_1$, $\overline{\beta} \notin \mathbf{E}$ (resp., $\beta \in \mathbf{E}$). A (multiple) evidence update (resp., erasure) operation is such that $\langle \mathbf{E} \cup \mathbf{E}_1, (\mathbf{U}, \mathbf{R})\rangle$ (resp., $\langle \mathbf{E} \backslash \mathbf{E}_1, (\mathbf{U}, \mathbf{R})\rangle$).*

The evidence update/erasure changes the *instance* of the DAF: it makes the set of active arguments vary. In that sense, it could be seen as a form of revision: the specification of what holds in the world is represented by active arguments and attacks. However, the impact of the evidence update in these sets neither performs nor is intended as a formal revision of the theory whatsoever.

With updates and erasures we do not change the representation (or specification) of the knowledge about the world, but what is perceived. Our update and erasure operations to change the set of evidence are so far treated shallowly, since it suffices to prove the usefulness of the theory presented in this article. However, ongoing work is devoted to reinforce this aspect, inspired by the original definitions given in (Katsuno and Mendelzon 1991).

Example 12 *Consider Example 10 and a DAF $\langle \mathbf{E}_{10}, (\mathbf{U}_{10}, \mathbf{R}_{12})\rangle$, where $\mathbf{R}_{12} = \{(\mathcal{A}_2, \mathcal{A}_1), (\mathcal{B}_1, \mathcal{A}_1), (\mathcal{B}_2, \mathcal{B}_1)\}$, as depicted in Figure 3[3]. Note that arrows represent the attack relation, and gray dashed triangles are inactive arguments.*

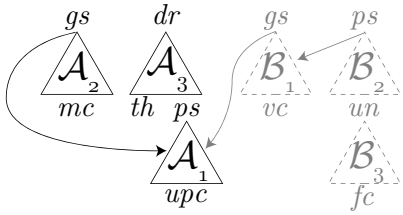


Figure 3: DAF from Example 12.

Then we have:

- $\mathbb{S}_{12} = \{\Sigma_2, \Sigma_{13}\}$ *are active argumental structures;*
- $\mathbb{S}^{\mathbb{I}}_{12} = \{\Sigma_1, \Sigma_{32}\}$ *are inactive argumental structures;*
- $\mathbb{R}_{12} = \{(\Sigma_2, \Sigma_{13})\}$ *are active attacks;*
- $\mathbb{R}^{\mathbb{I}}_{12} = \{(\Sigma_1, \Sigma_{13}), (\Sigma_{32}, \Sigma_1)\}$ *are inactive attacks.*

The static instance $(\mathbf{U}_{10}, \mathbf{R}_{12})|_{\mathbf{E}_{10}}$ is the AF $(\mathbb{S}_{12}, \mathbb{R}_{12})$, which is illustrated in Figure 4(a).

If we update the set of evidence by adding knowledge about the cops saying that they are volunteer, we have that Σ_1 becomes active, as well as its attack against Σ_{13}, leaving Σ_{32} as the only inactive structure, and (Σ_{32}, Σ_1) is the only inactive attack. The static instance of the updated DAF is depicted in Figure 4(b).

[3] Primitive argumental structures composed of $\mathcal{A}_1$, $\mathcal{A}_3$, $\mathcal{B}_2$ and $\mathcal{B}_3$ were not represented separately for the sake of simplicity.

Now consider we find out that the cops are foreigners, and that there is not as many as we were told before. Therefore, we make an update of the piece of evidence 'fc' and an erasure of 'mc'. This activates Σ_{32} and the attack (Σ_{32}, Σ_1), and inactivates Σ_2 along with its attack against Σ_{13}. This static instance is shown in Figure 4(c).

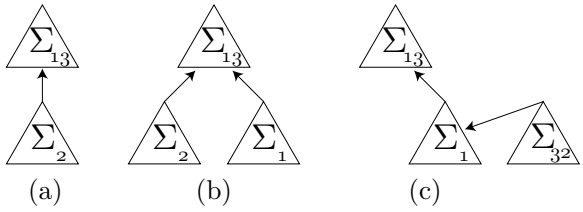


Figure 4: Static instances from Example 12.

Each static instance yields a particular set of accepted arguments. If we pick the grounded *semantics (Dung 1995): the static instance (a) accepts just the structure Σ_2; the static instance (b) accepts Σ_2 and Σ_1; the static instance (c) accepts Σ_{32} and Σ_{13}. Therefore, with this semantics, the last one is the only scenario in which we would believe the path we are analyzing to pass through is dangerous.*

In order to complete the relation between our work and Dung's there is also a way to obtain the associated DAF from a static framework. We only require the static framework to include arguments with an explicit set of premises and a claim, so they can be organized in primitive argumental structures (those containing just one argument). Then, the union of the sets of support of each argumental structure in the system is the set of evidence, and thus we have a DAF. However, this part of the relation is somehow weak: obtaining premises and claim out of an abstract argument may be difficult to do in a standard way. Therefore, the obtention of a DAF from a static framework is not currently in our focus. We are more interested in the reverse relation, so that you can specify a DAF and capture each of its static instances in the well-known AF format, and then make the analysis of acceptability of arguments in the usual way. The static instance of a DAF contains only active argumental structures (*i.e.*, all of them are used to make inferences) and an active attack relation connecting them, so the association between a DAF's static instance and an AF is quite direct. This is captured by Lemma 1.

Lemma 1 *The static instance of a DAF is* ***equivalent*** *to Dung's definition for an abstract argumentation framework.*
Proof: Trivial from Definition 18.

On the Applications of the DAF

This section describes some ongoing research lines that would take advantage of the DAF. Having a dynamic set of evidence that has a direct correlation with the set of active arguments allows reasoning about possible situations.

Argument Theory Change

Let us consider an argumentation-based agent with a certain goal G expressed in the form of a set of accepted arguments. Thus, we wish to know how should we *change* the set of evidence in order to reach G. In a recent paper (Rotstein et al. 2008), we presented a preliminary version of the DAF (that can be easily evolved to the current form) along with the basics of *argument theory change*. In that article, a *warrant-prioritized revision operator* is introduced: it introduces a new argument to a DAF seeking to be accepted; it does so by removing those arguments that interfere with this warrant, on behalf of a minimal change criterion. Hence, we would be able to tell which pieces of evidence are to be dropped for an argument to be accepted. Moreover, this approach could be extended by going in the opposite direction: bringing up the necessary evidence to activate those arguments that (because of the attacks they activate) would ensure the new argument to be accepted. In this way, we could go further and tell which pieces of evidence are to be added and which are to be dropped in order for a whole set of arguments to be accepted, whenever possible.

Argumentation-based Agent Architecture

An agent architecture is defined over a number of components, as the BDI model is composed by the three components handling beliefs, desires and intentions. If the components of an architecture are represented through a DAF, the agent is thus capable of not only updating its perception of the world (*i.e.*, the set of evidence), but also change its preferences and knowledge accordingly by adding/dropping the necessary pieces of evidence to activate/deactivate the corresponding arguments. An agent immersed in a dynamic environment should be prepared to incorporate changes in the world's rules (*i.e.*, the way things are interpreted); for instance, an agent dedicated to schedule processes in an agent-oriented operative system must be able to change the schedule policy. If the set of evidence is kept up-to-date, the agent will make inferences based on the current state of the world, but it could also hypothesize about possible states representing variations of the current one, thus being able to move towards its goal. Finally, the dynamic modification of a particular subset of evidence could allow the agent to learn from its own experience in order to adapt.

Analysis of Legal Cases

Another application that could benefit from the usage of the dynamic argumentation framework is the analysis of legal cases. Assume that a verdict has been reached regarding a certain case and that the accused was found guilty. We have a number of allegations (*i.e.*, arguments) that were posed against the presumption of innocence, and arguments against them, and so on, yielding a graph of arguments interrelated by the attack relation. The argument graph is a visualization of the framework that justifies the verdict. Note that the semantics chosen should be sensible as to classify the presumption of innocence as a rejected argument. We also have the set of evidence from which arguments were based. All the arguments posed should be active, since they were accepted at the trial. Now an appropriate mechanism could be used in order to vary the set of evidence and discover under which circumstances the convict could have

been found innocent (for instance, we could use the warrant-prioritized revision operator mentioned above, in 'Argument Theory Change'). Moreover, we could even add to the framework those arguments that did not have enough support from available evidence (which are inactive), and play with the possibility of actually having that evidence. The dynamic framework plus an appropriate mechanism could be a useful tool to hypothesize about possible scenarios and outcomes of an actual legal case.

Conclusions and Future Work

In this article we have presented a new approach to abstract argumentation frameworks. Our model, as many others, is based on Dung's framework (AF) and represents an extension that is the basis of several research lines, some of which were introduced in the previous section. The main subject of this paper is the definition of the Dynamic Argumentation Framework, from which a static instance can be obtained. This instance was shown to be equivalent to the AF. However, throughout several examples, it was shown that the DAF allows for a more general representation of knowledge than the mentioned framework: it considers a varying set of evidence that changes the base to make inferences (*i.e.*, the underlying static instance); therefore, multiple AFs can be obtained from different instances of a given DAF.

Regarding future work, besides what was already discussed in the previous section, we are also interested in exploring the capability of reasoning about possible situations, and establishing a relation with the area of modal logics. This work is currently underway. Although this research line is mainly theoretical, one of its main goals is to make an implementation out of each application for the DAF. This is likely to be done in Defeasible Logic Programming (DeLP) (García and Simari 2004) or an extension of it. For instance, the formalism defined in (Rotstein et al. 2008) found its DeLP reification in (Moguillansky et al.).

Finally, we will explore two extensions of the DAF: (1) a more natural way to specify and build the attack relation between arguments from constraints and preferences; (2) change operators to modify the universal set of arguments and the attack relation. The first extension will involve a slight change in the definition of the framework: it will build the attack relation from the specification of a set of constraints among claims (each constraint is an n-tuple of claims), stating which claims cannot hold together. As expected, some constraints are implicit, such as pairs of complementary claims. Once conflicts among arguments are obtained, a preference function will decide, for each pair, which argument prevails. The second extension acts as a meta-level debugging tool, allowing the dynamic modification of the sets of arguments and attacks. Note that the modification of the universal set of arguments turns it into a working set instead. Special care has to be taken regarding the addition of attacks, since this may introduce new arguments. The same applies to the deletion of arguments, since some attacks will no longer be valid.

Acknowledgements

This work is partially financed by CONICET (PIP 5050), Universidad Nacional del Sur and Agencia Nacional de Promoción Científica y Tecnológica.

References

Amgoud, L.; Cayrol, C.; and Lagasquie-Schiex, M.-C. 2004. On the bipolarity in argumentation frameworks. In *NMR*, 1–9.

Baroni, P., and Giacomin, M. 2007. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.* 171(10-15):675–700.

Bench-Capon, T., and Dunne, P. 2007. Argumentation in artificial intelligence. *Artif. Intell.* 171(10-15):619–641.

Chesñevar, C.; McGinnis, J.; Modgil, S.; Rahwan, I.; Reed, C.; Simari, G.; South, M.; Vreeswijk, G.; and Willmott, S. 2006. Towards an argument interchange format. *Knowl. Eng. Rev.* 21(4):293–316.

Chesñevar, C.; Maguitman, A.; and Loui, R. 2000. Logical Models of Argument. *ACM Computing Surveys* 32(4):337–383.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning and Logic Programming and n-person Games. *Artificial Intelligence* 77:321–357.

García, A. J., and Simari, G. R. 2004. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Katsuno, H., and Mendelzon, A. 1991. On the difference between updating a knowledge base and revising it. In *Proc. of KR'91*. 387–394.

Martínez, D. C.; García, A. J.; and Simari, G. R. 2007. Modelling well-structured argumentation lines. In *Proc. of Int. Joint Conf. on Artif. Intelligence IJCAI-2007*, 465–470.

Moguillansky, M.; Rotstein, N.; Falappa, M.; García, A.; and Simari, G. Argument Theory Change Applied to Defeasible Logic Programming. In *23rd. AAAI Conf. on Artificial Intelligence (AAAI 2008)*, 132–137.

Prakken, H., and Vreeswijk, G. 2000. Logical systems for defeasible argumentation. In D.Gabbay., ed., *Handbook of Philosophical Logic, 2nd ed.* Kluwer Academic Pub.

Rotstein, N. D.; Moguillansky, M.; Falappa, M. A.; García, A. J.; and Simari, G. R. 2008. Argument theory change: Revision upon warrant. In Besnard, P.; Doutre, S.; and Hunter, A., eds., *COMMA*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, 336–347. IOS Press.

Toulmin, S. 1959. *The Uses of Argument*. Cambridge University Press.

Walton, D. 1996. *Argument Structure: A Pragmatic Theory (Toronto Studies in Philosophy)*. Univ. of Toronto Press.

Wyner, A., and Bench-Capon, T. 2007. Towards an extensible argumentation system. In *ECSQARU*, 283–294.

Special Session on Declarative Programming Paradigms and Systems for NMR

For many years now, formalisms rooted in the research area of Nonmonotonic Reasoning have been used as the theoretical foundation for declarative programming paradigms. These programming paradigms provide expressive languages to represent nonmonotonic concepts besides other knowledge, and systems that are implemented to automate nonmonotonic reasoning. For instance, one of the most successful of such paradigms is Answer Set Programming.

Some of the existing NMR systems are reaching a level of maturity where *serious* real-world applications can be – and are being – developed. At the same time, we are still witnessing an impressive research effort in creating, developing and extending (nonmonotonic) declarative languages to meet the (new) needs of specific application domains (e.g., Multi-Agent Systems, Semantic Web, Web Services, Textual Entailment, Computational Biology).

Session Chairs

Esra Erdem, Sabanci University, Turkey
João Leite, New University of Lisbon, Portugal

Program Committee

Jose Alferes, New University of Lisbon, Portugal
Marcello Balduccini, Kodak Research Labs, USA
Chitta Baral, Arizona State University, USA
Marina de Vos, University of Bath, UK
Jürgen Dix, TU Clausthal, Germany
Thomas Eiter, Vienna University of Technology, Austria
Paolo Ferraris, Google, USA
Michael Fink, Vienna University of Technology, Austria
Norman Foo, NICTA, University of New South Wales, Australia
Alfredo Gabaldon, NICTA, University of New South Wales, Australia
Martin Gebser, University of Potsdam, Germany
Michael Gelfond, Texas Tech University, USA
Giovambattista Ianni, University of Calabria, Italy
Joohyung Lee, Arizona State University, USA
Nicola Leone, University of Calabria, Italy
Yuliya Lierler, University of Texas at Austin, USA
Fangzhen Lin, Hong Kong University of Science and Technology, Hong Kong
Marco Maratea, University of Genoa, Italy
Ilkka Niemelä, Helsinki University of Technology, Finland
Enrico Pontelli, New Mexico State University, USA
Chiaki Sakama, Wakayama University, Japan
Ken Satoh, National Institute of Informatics, Japan
Torsten Schaub, University of Potsdam, Germany
Tran Cao Son, New Mexico State University, USA
Terrance Swift, XSB Inc., USA
Mirek Truszczynski, University of Kentucky, USA

Heuristics in Conflict Resolution

Christian Drescher and **Martin Gebser** and **Benjamin Kaufmann** and **Torsten Schaub**
Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam, Germany

Abstract

Modern solvers for Boolean Satisfiability (SAT) and Answer Set Programming (ASP) are based on sophisticated Boolean constraint solving techniques. In both areas, conflict-driven learning and related techniques constitute key features whose application is enabled by conflict analysis. Although various conflict analysis schemes have been proposed, implemented, and studied both theoretically and practically in the SAT area, the heuristic aspects involved in conflict analysis have not yet received much attention. Assuming a fixed conflict analysis scheme, we address the open question of how to identify "good" reasons for conflicts, and we investigate several heuristics for conflict analysis in ASP solving. To our knowledge, a systematic study like ours has not yet been performed in the SAT area, thus, it might be beneficial for both the field of ASP as well as the one of SAT solving.

Introduction

The popularity of Answer Set Programming (ASP; (Baral 2003)) as a paradigm for knowledge representation and reasoning is mainly due to two factors: first, its rich modeling language and, second, the availability of high-performance ASP systems. In fact, modern ASP solvers, such as *clasp* (Gebser *et al.* 2007a), *cmodels* (Giunchiglia, Lierler, & Maratea 2006), and *smodels*$_{cc}$ (Ward & Schlipf 2004), have meanwhile closed the gap to Boolean Satisfiability (SAT; (Mitchell 2005)) solvers. In both fields, conflict-driven learning and related techniques have led to significant performance boosts (Bayardo & Schrag 1997; Marques-Silva & Sakallah 1999; Moskewicz *et al.* 2001; Gebser *et al.* 2007d). The basic prerequisite for the application of such techniques is *conflict analysis*, that is, the extraction of non-trivial reasons for dead ends encountered during search. Even though ASP and SAT solvers exploit different inference patterns, their underlying search techniques are closely related to each other. For instance, the basic search strategy of SAT solver *chaff* (Moskewicz *et al.* 2001), nowadays a quasi standard in SAT solving, is also exploited by ASP solver *clasp*, in particular, the principles of conflict analysis are similar. Vice versa, the solution enumeration approach implemented in *clasp* (Gebser *et al.* 2007b) could also be applied by SAT solvers. Given these similarities, general search or, more specifically, conflict analysis techniques developed in one community can (almost) immediately be exploited in the other field too.

In this paper, we address the problem of identifying "good" reasons for conflicts to be recorded within an ASP solver. In fact, conflict-driven learning exhibits several degrees of freedom. For instance, several constraints may become violated simultaneously, in which case one can choose the conflict(s) to be analyzed. Furthermore, distinct schemes may be used for conflict analysis, such as the resolution-based First-UIP and Last-UIP scheme (Zhang *et al.* 2001). Finally, if conflict analysis is based on resolution, several constraints may be suitable resolvents, likewise permitting to eliminate some literal in a resolution step.

For the feasibility of our study, it was necessary to prune dimensions of freedom in favor of predominant options. In the SAT area, the *First-UIP scheme* (Marques-Silva & Sakallah 1999) has empirically been shown to yield better performance than other known conflict resolution strategies (Zhang *et al.* 2001). We thus fix the conflict analysis strategy to conflict resolution according to the First-UIP scheme. Furthermore, it seems reasonable to analyze the first conflict detected by a solver (although conflicts encountered later on may actually yield "better" reasons). This leaves to us the choice of the resolvents to be used for conflict resolution, and we investigate this issue with respect to different goals: reducing the size of reasons to be recorded, skipping greater portions of the search space by backjumping (explained below), reducing the number of conflict resolution steps, and reducing the overall number of encountered conflicts (roughly corresponding to runtime). To this end, we modified the conflict analysis procedure of our ASP solver *clasp*[1] for accommodating a variety of heuristics for choosing resolvents. The developed heuristics and comprehensive empirical results for them are presented in this paper.

Logical Background

We assume basic familiarity with answer set semantics (see, for instance, (Baral 2003)). This section briefly introduces notations and recalls a constraint-based characterization of answer set semantics according to (Gebser *et al.* 2007c). We consider propositional (normal) logic programs over an alphabet $\mathcal{P}$. A *logic program* is a finite set of *rules*

$$p_0 \leftarrow p_1, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n \qquad (1)$$

[1] http://www.cs.uni-potsdam.de/clasp

where $0 \leq m \leq n$ and $p_i \in \mathcal{P}$ is an *atom* for $0 \leq i \leq n$. For a rule r as in (1), let $head(r) = p_0$ be the *head* of r and $body(r) = \{p_1, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n\}$ be the *body* of r. The set of atoms occurring in a logic program Π is denoted by $atom(\Pi)$, and the set of bodies in Π is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping bodies sharing the same head p, define $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$.

For characterizing the answer sets of a program Π, we consider Boolean assignments A over *domain* $dom(A) = atom(\Pi) \cup body(\Pi)$. Formally, an *assignment* A is a sequence $(\sigma_1, \ldots, \sigma_n)$ of (signed) *literals* σ_i of the form $\mathbf{T}v$ or $\mathbf{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$. Intuitively, $\mathbf{T}v$ expresses that v is *true* and $\mathbf{F}v$ that it is *false* in A. We denote the complement of a literal σ by $\overline{\sigma}$, that is, $\overline{\mathbf{T}v} = \mathbf{F}v$ and $\overline{\mathbf{F}v} = \mathbf{T}v$. Furthermore, we let $A \circ B$ denote the sequence obtained by concatenating two assignments A and B. We sometimes abuse notation and identify an assignment with the set of its contained literals. Given this, we access the true and false propositions in A via $A^{\mathbf{T}} = \{p \in dom(A) \mid \mathbf{T}p \in A\}$ and $A^{\mathbf{F}} = \{p \in dom(A) \mid \mathbf{F}p \in A\}$. Finally, we denote the prefix of A up to a literal σ by

$$A[\sigma] = \begin{cases} (\sigma_1, \ldots, \sigma_m) & \text{if } A = (\sigma_1, \ldots, \sigma_m, \sigma, \ldots, \sigma_n) \\ A & \text{if } \sigma \notin A\,. \end{cases}$$

In our context, a *nogood* (Dechter 2003) is a set $\{\sigma_1, \ldots, \sigma_m\}$ of literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_m$. An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a *solution* for a set Δ of nogoods if $\delta \not\subseteq A$ for all $\delta \in \Delta$. Given a logic program Π, we below specify nogoods such that their solutions correspond to the answer sets of Π.

We start by describing nogoods capturing the models of the Clark's *completion* (Clark 1978) of a program Π. For $(\beta = \{p_1, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n\}) \in body(\Pi)$, let

$$\Delta_\beta = \left\{ \begin{array}{l} \{\mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n, \mathbf{F}\beta\}, \\ \{\mathbf{F}p_1, \mathbf{T}\beta\}, \ldots, \{\mathbf{F}p_m, \mathbf{T}\beta\}, \\ \{\mathbf{T}p_{m+1}, \mathbf{T}\beta\}, \ldots, \{\mathbf{T}p_n, \mathbf{T}\beta\} \end{array} \right\}.$$

Observe that every solution for Δ_β must assign body β equivalent to the conjunction of its elements. Similarly, for an atom $p \in atom(\Pi)$, the following nogoods stipulate p to be equivalent to the disjunction of $body(p) = \{\beta_1, \ldots, \beta_k\}$:

$$\Delta_p = \left\{ \begin{array}{l} \{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k, \mathbf{T}p\}, \\ \{\mathbf{T}\beta_1, \mathbf{F}p\}, \ldots, \{\mathbf{T}\beta_k, \mathbf{F}p\} \end{array} \right\}.$$

Combining the above nogoods for Π, we get

$$\Delta_\Pi = \bigcup_{\beta \in body(\Pi)} \Delta_\beta \cup \bigcup_{p \in atom(\Pi)} \Delta_p\,.$$

The solutions for Δ_Π correspond one-to-one to the models of the completion of Π. If Π is *tight* (Fages 1994; Erdem & Lifschitz 2003), these models are guaranteed to match the answer sets of Π. This can be formally stated as follows.

Theorem 1 ((Gebser *et al.* 2007c)) *Let Π be a tight logic program. Then, $X \subseteq atom(\Pi)$ is an answer set of Π iff $X = A^{\mathbf{T}} \cap atom(\Pi)$ for a (unique) solution A for Δ_Π.*

We proceed by considering non-tight programs Π. As shown in (Lin & Zhao 2004), *loop formulas* can be added to the completion of Π to establish full correspondence to the answer sets of Π. For $U \subseteq atom(\Pi)$, let $EB_\Pi(U)$ be

$$\{body(r) \mid r \in \Pi, head(r) \in U, body(r) \cap U = \emptyset\}\,.$$

Observe that $EB_\Pi(U)$ contains the bodies of all rules in Π that can *externally support* (Lee 2005) an atom in U. Given $U = \{p_1, \ldots, p_j\}$ and $EB_\Pi(U) = \{\beta_1, \ldots, \beta_k\}$, the following nogoods capture the loop formula of U:

$$\Lambda_U = \left\{ \begin{array}{l} \{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k, \mathbf{T}p_1\}, \ldots, \\ \{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k, \mathbf{T}p_j\} \end{array} \right\}.$$

Furthermore, we define

$$\Lambda_\Pi = \bigcup_{U \subseteq atom(\Pi)} \Lambda_U\,.$$

By augmenting Δ_Π with Λ_Π, Theorem 1 can be extended to non-tight programs.

Theorem 2 ((Gebser *et al.* 2007c)) *Let Π be a logic program. Then, $X \subseteq atom(\Pi)$ is an answer set of Π iff $X = A^{\mathbf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.*

By virtue of Theorem 2, the nogoods in $\Delta_\Pi \cup \Lambda_\Pi$ provide us with a constraint-based characterization of the *answer sets* of Π. However, it is important to note that the size of Δ_Π is linear in $atom(\Pi) \times body(\Pi)$, while Λ_Π contains exponentially many nogoods. As shown in (Lifschitz & Razborov 2006), under current assumptions in complexity theory, the exponential number of elements in Λ_Π is inherent, that is, it cannot be reduced significantly in the worst case. Hence, ASP solvers do not determine the nogoods in Λ_Π a priori, but include mechanisms to determine them on demand. This is illustrated further in the next section.

Algorithmic Background

This section recalls the basic decision procedure of *clasp* (Gebser *et al.* 2007c), abstracting Conflict-Driven Clause Learning (CDCL; (Mitchell 2005)) for SAT solving from clauses, that is, Conflict-Driven Nogood Learning (CDNL).

Conflict-Driven Nogood Learning

Algorithm 1 shows our main procedure for deciding whether a program Π has some answer set. The algorithm starts with an empty assignment A and an empty set ∇ of recorded nogoods (Lines 1–2). Note that dynamic nogoods added to ∇ in Line 5 are elements of Λ_Π, while those added in Line 9 result from conflict analysis (Line 8). In addition to conflict-driven learning, the procedure performs backjumping (Lines 10–11), guided by a decision level k determined by conflict analysis. Via decision level dl, we count *decision literals*, that is, literals in A that have been heuristically selected in Line 15. The initial value of dl is 0 (Line 3), and it is incremented in Line 16 before a decision literal is added to A (Line 17). All literals in A that are not decision literals have been derived by propagation in Line 5, and we call them *implied literals*. For any literal σ in A, we write $dl(\sigma)$ to refer to the decision level of σ, that is, the value dl had when σ was added to A. After propagation, the main loop

Algorithm 1: CDNL

Input : A program Π.
Output: An answer set of Π.

```
1  A ← ∅                                  // assignment over atom(Π) ∪ body(Π)
2  ∇ ← ∅                                  // set of (dynamic) nogoods
3  dl ← 0                                 // decision level
4  loop
5      (A, ∇) ← PROPAGATION(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then
7          if dl = 0 then return no answer set
8          (δ, k) ← CONFLICTANALYSIS(ε, Π, ∇, A)
9          ∇ ← ∇ ∪ {δ}
10         A ← A \ {σ ∈ A | k < dl(σ)}
11         dl ← k
12     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then
13         return A^T ∩ atom(Π)
14     else
15         σ_d ← SELECT(Π, ∇, A)
16         dl ← dl + 1
17         A ← A ∘ (σ_d)
```

(Lines 4–17) distinguishes three cases: a conflict detected via a violated nogood (Lines 6–11), a solution (Lines 12–13), or a heuristic selection with respect to a partial assignment (Lines 14–17). Finally, note that a conflict at decision level 0 signals that Π has no answer set (Line 7).

Propagation

Our propagation procedure, shown in Algorithm 2, derives implied literals and adds them to A. Lines 3–9 describe unit propagation (cf. (Mitchell 2005)) on $\Delta_\Pi \cup \nabla$. If a conflict is detected in Line 4, unit propagation terminates immediately (Line 5). Otherwise, in Line 6, we determine all nogoods δ that are *unit-resulting* wrt A, that is, the complement $\overline{\sigma}$ of some literal $\sigma \in \delta$ must be added to A because all other literals of δ are already true in A. If there is some unit-resulting nogood δ (Line 7), A is augmented with $\overline{\sigma}$ in Line 8. Observe that δ is chosen non-deterministically, and several distinct nogoods may imply $\overline{\sigma}$ wrt A. This non-determinism gives rise to our study of heuristics for conflict resolution, selecting a resolvent among the nogoods δ that imply $\overline{\sigma}$.

The second part of Algorithm 2 (Lines 10–14) checks for unit-resulting or violated nogoods in Λ_Π. If Π is tight (Line 10), sophisticated checks are unnecessary (cf. Theorem 1). Otherwise, we consider sets $U \subseteq atom(\Pi)$ such that $EB_\Pi(U) \subseteq A^{\mathbf{F}}$, called *unfounded sets* (Van Gelder, Ross, & Schlipf 1991). An unfounded set U is determined in Line 12 by a dedicated algorithm, where $U \cap A^{\mathbf{F}} = \emptyset$. If such a nonempty unfounded set U exists, each nogood $\delta \in \Lambda_U$ is either unit-resulting or violated wrt A, and an arbitrary $\delta \in \Lambda_U$ is recorded in Line 14 for triggering unit propagation. Note that all atoms in U must be falsified before another unfounded set is determined (cf. Lines 11–12). Eventually, propagation terminates in Line 13 if no nonempty unfounded set has been detected in Line 12.

Algorithm 2: PROPAGATION

Input : A program Π, a set ∇ of nogoods, and an assignment A.
Output: An extended assignment and set of nogoods.

```
1  U ← ∅                                  // unfounded set
2  loop
3      repeat
4          if δ ⊆ A for some δ ∈ Δ_Π ∪ ∇ then
5              return (A, ∇)
6          Σ ← {δ ∈ Δ_Π ∪ ∇ | δ \ A = {σ}, σ̄ ∉ A}
7          if Σ ≠ ∅ then let σ ∈ δ \ A for some δ ∈ Σ in
8              A ← A ∘ (σ̄)
9      until Σ = ∅
10     if TIGHT(Π) then return (A, ∇)
11     U ← U \ A^F
12     if U = ∅ then U ← UNFOUNDEDSET(Π, A)
13     if U = ∅ then return (A, ∇)
14     let δ ∈ Λ_U in ∇ ← ∇ ∪ {δ}
```

Algorithm 3: CONFLICTANALYSIS

Input : A violated nogood δ, a program Π, a set ∇ of nogoods, and an assignment A.
Output: A derived nogood and a decision level.

```
1  loop
2      let σ ∈ δ such that δ \ A[σ] = {σ}
3      k ← max({dl(ρ) | ρ ∈ δ \ {σ}} ∪ {0})
4      if k = dl(σ) then
5          Σ ← {ε ∈ Δ_Π ∪ ∇ | ε \ A[σ] = {σ̄}}
6          ε ← SELECTANTECEDENT(Σ)
7          δ ← (δ \ {σ}) ∪ (ε \ {σ̄})
8      else return (δ, k)
```

Conflict Analysis

Algorithm 3 shows our conflict analysis procedure, which is based on resolution. Given a nogood δ that is violated wrt A, we determine in Line 2 the literal $\sigma \in \delta$ added last to A. If σ is the single literal of its decision level $dl(\sigma)$ in δ (cf. Line 3), it is called a *unique implication point* (UIP; (Marques-Silva & Sakallah 1999)). Among a number of conflict resolution schemes, the *First-UIP* scheme, stopping conflict resolution as soon as the first UIP is reached, has turned out to be the most efficient and most robust strategy (Zhang *et al.* 2001). Our conflict analysis procedure follows the First-UIP scheme by performing conflict resolution only if σ is not a UIP (tested in Line 4) and, otherwise, returning δ along with the smallest decision level k at which $\overline{\sigma}$ is implied by δ after backjumping (Line 8).

Let us take a closer look at conflict resolution steps in Lines 5–7. It is important to note that, if σ is not a UIP, it cannot be the decision literal of $dl(\sigma)$. Rather, it must have been implied by some nogood $\varepsilon \in \Delta_\Pi \cup \nabla$. As a consequence, the set Σ determined in Line 5 cannot be empty, and

we call its elements *antecedents* of σ. Note that each antecedent ε contains $\overline{\sigma}$ and had been unit-resulting immediately before σ was added to A; we thus call $\varepsilon \setminus \{\overline{\sigma}\}$ a *reason* for σ. Knowing that σ may have more than one antecedent, a non-deterministic choice among them is made in Line 6. Exactly this choice is subject to the heuristics studied below. Furthermore, as σ is the literal of δ added last to A, $\delta \setminus \{\sigma\}$ is also a reason for $\overline{\sigma}$. Since they imply complementary literals, no solution can jointly contain both reasons, viz., $\delta \setminus \{\sigma\}$ and $\varepsilon \setminus \{\overline{\sigma}\}$. Hence, combining them in Line 7 gives again a nogood violated wrt A. Finally, note that conflict resolution is guaranteed to terminate at some UIP, but different heuristic choices in Line 6 may result in different UIPs.

Implication Graphs and Conflict Graphs

To portray the matter of choosing among several distinct antecedents, we modify the notion of an implication graph (Beame, Kautz, & Sabharwal 2004). At a given state of CDNL, the *implication graph* contains a node for each literal σ in assignment A and, for a violated nogood $\delta \subseteq A$, a node $\overline{\sigma}$ is included, where σ is the literal of δ added last to A, that is, $\delta \setminus A[\sigma] = \{\sigma\}$. Furthermore, for each antecedent δ of an implied literal σ, the implication graph contains directed edges labeled with δ from all literals in the reason $\delta \setminus \{\overline{\sigma}\}$ to σ. Different from (Beame, Kautz, & Sabharwal 2004), where implication graphs reflect exactly one reason per implied literal, our implication graph thus includes all of them. If the implication graph contains both σ and $\overline{\sigma}$, we call them *conflicting literals*. Note that an implication graph contains at most one such pair $\{\sigma, \overline{\sigma}\}$, called *conflicting assignment*, because our propagation procedure in Algorithm 2 stops as soon as a nogood becomes violated (cf. Lines 4–5).

An exemplary implication graph is shown in Figure 1. Each of its nodes (except for one among the two conflicting literals) corresponds to a literal that is true in assignment

$$A = \left(\mathbf{F}a, \mathbf{F}b, \underline{\mathbf{F}p}, \underline{\mathbf{T}q}, \underline{\mathbf{T}r}, \mathbf{T}s, \mathbf{F}v, \mathbf{T}t, \mathbf{F}u, \mathbf{F}w, \mathbf{T}x\right) .$$

The three decision literals in A are underlined, and all other literals are implied. For each literal σ, its decision level $dl(\sigma)$ is also provided in Figure 1 in parentheses. Every edge is labeled with at least one antecedent of its target, that is, the edges represent the following nogoods:

$$\begin{aligned} n_0 &= \{\mathbf{F}a, \mathbf{T}b\} & n_1 &= \{\mathbf{T}r, \mathbf{F}s\} \\ n_2 &= \{\mathbf{T}s, \mathbf{F}t\} & n_3 &= \{\mathbf{T}s, \mathbf{T}u\} \\ n_4 &= \{\mathbf{T}s, \mathbf{T}w\} & n_5 &= \{\mathbf{T}r, \mathbf{T}v\} \\ n_6 &= \{\mathbf{T}q, \mathbf{F}v, \mathbf{T}w\} & n_7 &= \{\mathbf{T}t, \mathbf{F}u, \mathbf{F}x\} \\ n_8 &= \{\mathbf{F}p, \mathbf{T}t, \mathbf{F}x\} & n_9 &= \{\mathbf{F}w, \mathbf{T}x\} . \end{aligned}$$

Furthermore, nogood $\{\mathbf{T}a\}$ is unit-resulting wrt the empty assignment, thus, implied literal $\mathbf{F}a$ (whose decision level is 0) does not have any incoming edge. Observe that the implication graph contains conflicting assignment $\{\mathbf{T}x, \mathbf{F}x\}$, where $\mathbf{T}x$ has been implied by nogood n_7 and likewise by n_8. It is also the last literal in A belonging to violated nogood n_9, so that its complement $\mathbf{F}x$ is the second conflicting literal in the implication graph. Besides $\mathbf{T}x$, literal $\mathbf{F}w$ has multiple antecedents, namely, n_4 and n_6, which can be read off the labels of the incoming edges of $\mathbf{F}w$.

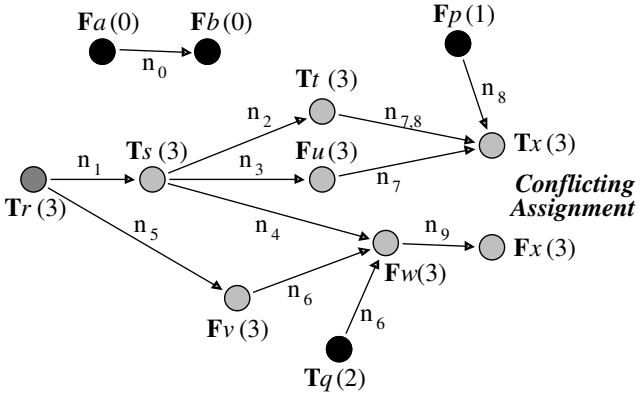


Figure 1: An exemplary implication graph containing a conflicting assignment.

The conflict resolution done in Algorithm 3, in particular, the heuristic choice of antecedents in Line 6, can now be viewed as an iterative projection of the implication graph. In fact, if an implied literal has incoming edges with distinct labels, all edges with a particular label are taken into account, while the edges with different labels only are dropped. This observation motivates the following definition: a subgraph of an implication graph is a *conflict graph* if it contains a conflicting assignment and, for each implied literal σ in the subgraph, the set of predecessors of σ is a reason for σ. Note that this definition allows us to drop all literals that do not have a path to any conflicting literal, such as $\mathbf{F}a$ and $\mathbf{F}b$ in Figure 1. Furthermore, the requirement that the predecessors of an implied literal form a reason corresponds to the selection of an antecedent, where only the incoming edges with a particular label are traced via conflict resolution.

The next definition accounts for a particularity of ASP solving related to unfounded set handling: a conflict graph is *level-aware* if each conflicting literal σ has some predecessor ρ such that $dl(\rho) = dl(\sigma)$. In fact, propagation in Algorithm 2 is limited to falsifying unfounded atoms, thus, unit propagation on nogoods in Λ_Π is performed only partially and may miss implied literals corresponding to external bodies (cf. (Gebser *et al.* 2007c)). If a conflict graph is not level-aware, the violated nogood δ provided as input to Algorithm 3 already contains a UIP, thus, δ itself is returned without performing any conflict resolution inbetween. Given that we are interested in conflict resolution, we below consider level-aware conflict graphs only.

Finally, we characterize nogoods derived by Algorithm 3 by cuts in conflict graphs (cf. (Zhang *et al.* 2001; Beame, Kautz, & Sabharwal 2004)). A *conflict cut* in a conflict graph is a bipartition of the nodes such that all decision literals belong to one side, called *reason side*, and the conflicting assignment is contained in the other side, called *conflict side*. The set of nodes on the reason side that have some edge into the conflict side form the *conflict nogood* associated with a particular conflict cut. For illustration, a First-New-Cut (Beame, Kautz, & Sabharwal 2004) is shown in Figure 2. For the underlying conflict graph, we can choose among the

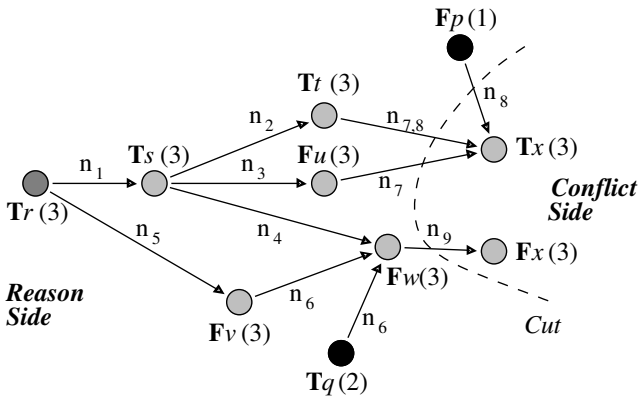


Figure 2: The implication graph with a First-New-Cut.

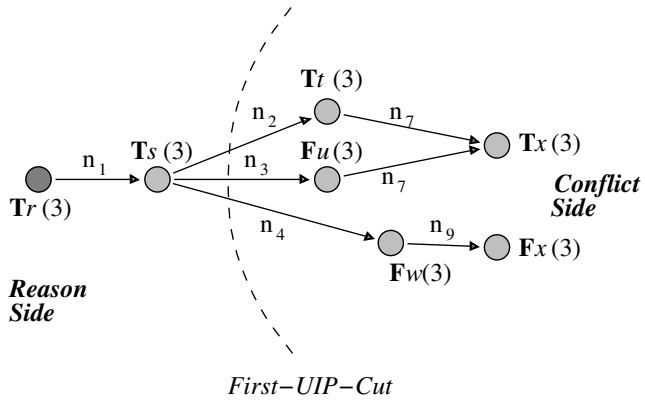


Figure 3: A First-UIP-Cut obtained with H_{short}.

incoming edges of $\mathbf{T}x$ whether to include the edges labeled with n_7 or the ones labeled with n_8. With n_7, we get conflict nogood $\{\mathbf{T}t, \mathbf{F}u, \mathbf{F}w\}$, while n_8 yields $\{\mathbf{F}p, \mathbf{T}t, \mathbf{F}w\}$.

Different conflict cuts correspond to different resolution schemes, where we are particularly interested in the First-UIP scheme. Given a conflict graph and conflicting assignment $\{\sigma, \overline{\sigma}\}$, a UIP σ_{UIP} can be identified as a node such that all paths from σ_d, the decision literal of decision level $dl(\sigma) = dl(\overline{\sigma})$, to either σ or $\overline{\sigma}$ go through σ_{UIP} (cf. (Zhang *et al.* 2001)). In view of this alternative definition of a UIP, it becomes even more obvious than before that σ_d is indeed a UIP, also called the Last-UIP. In contrast, a literal σ_{UIP} is the First-UIP if it is the UIP "closest" to the conflicting literals, that is, if no other UIP is reachable from σ_{UIP}. The *First-UIP-Cut* is then given by the conflict cut that has all literals lying on some path from the First-UIP to a conflicting literal, except for the First-UIP itself, on the conflict side and all other literals (including the First-UIP) on the reason side. The *First-UIP-Nogood*, that is, the conflict nogood associated with the First-UIP-Cut, is exactly the nogood derived by conflict resolution in Algorithm 3 when antecedents that contribute edges to the conflict graph are selected for conflict resolution. Also note that the First-UIP-Cut for a conflict graph is unique, thus, by projecting an implication graph to a conflict graph, we implicitly fix the First-UIP-Nogood. With this is mind, the next section deals with heuristics for extracting conflict graphs from implication graphs.

Heuristics

In this section, we propose several heuristics for conflict resolution striving for different goals.

Recording Short Nogoods

Under the assumption that short nogoods prune larger portions of the search space than longer ones, a First-UIP-Nogood looks the more attractive the less literals it contains. In addition, unit propagation on shorter nogoods is usually faster and might even be enabled to use particularly optimized data structures, for instance, specialized to binary or ternary nogoods (Ryan 2004). As noticed in (Mahajan, Fu, & Malik 2005), a conflict nogood stays short when the resolvents are short, when the number of resolvents is small, or when the resolvents have many literals in common. In the SAT area, it has been observed that preferring short nogoods in conflict resolution may lead to resolution sequences involving mostly binary and ternary nogoods, so that derived conflict nogoods are not much longer than the originally violated nogoods (Mitchell 2005). Our first heuristics, H_{short}, thus selects an antecedent containing the smallest number of literals among the available antecedents of a literal. Given the same implication graph as in Figure 1 and 2, H_{short} may yield the conflict graph shown in Figure 3 by preferring antecedent n_7 of $\mathbf{T}x$ over n_8 and antecedent n_4 of $\mathbf{F}w$ over n_6 during conflict resolution. The corresponding First-UIP-Nogood, $\{\mathbf{T}s\}$, is indeed short and enables CDNL to after backjumping derive $\mathbf{F}s$ by unit propagation at decision level 0. However, the antecedents n_7 and n_8 of $\mathbf{T}x$ are of the same size, thus, H_{short} may likewise pick n_8, in which case the First-UIP-Cut in Figure 4 is obtained. The corresponding First-UIP-Nogood, $\{\mathbf{F}p, \mathbf{T}s\}$, is longer. Nonetheless, our experiments below empirically confirm that H_{short} tends to reduce the size of First-UIP-Nogoods. But before, we describe further heuristics focusing also on other aspects.

Performing Long Backjumps

By backjumping, CDNL may skip the exhaustive exploration of regions of the search space, possibly escaping spare regions not containing any solution. Thus, it seems reasonable to aim at First-UIP-Nogoods such that their literals belong to small decision levels, as they are the determining factor for the lengths of backjumps. Our second heuristics, H_{lex}, thus uses a lexicographic order to rank antecedents according to the decision levels of their literals. Given an antecedent δ of a literal σ, we arrange the literals in the reason $\delta \setminus \{\overline{\sigma}\}$ for σ in descending order of their decision levels. The so obtained sequence $(\sigma_1, \ldots, \sigma_m)$, where $\delta \setminus \{\overline{\sigma}\} = \{\sigma_1, \ldots, \sigma_m\}$, induces a descending list $levels(\delta) = (dl(\sigma_1), \ldots, dl(\sigma_m))$ of decision levels. An antecedent δ is then considered to be smaller than another antecedent ε, viz., $\delta < \varepsilon$, if the first element that differs in $levels(\delta)$ and $levels(\varepsilon)$ is smaller in $levels(\delta)$ or if $levels(\delta)$

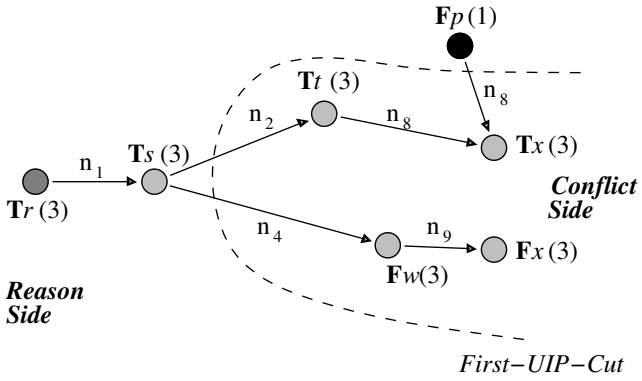


Figure 4: A First-UIP-Cut obtained with H_{lex}.

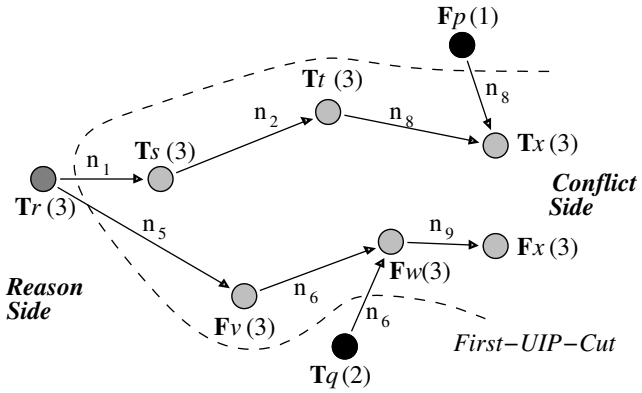


Figure 5: A First-UIP-Cut obtained with H_{avg}.

is a prefix of $levels(\varepsilon)$ and shorter than $levels(\varepsilon)$. Due to the last condition, H_{lex} also prefers an antecedent δ that is shorter than ε, provided that literals of the same decision levels as in δ are also found in ε. Reconsidering the implication graph in Figure 1 and 2, we obtain $levels(n_8) = (3,1) < (3,3) = levels(n_7)$ for antecedents n_7 and n_8 of $\mathbf{T}x$, and we have $levels(n_4) = (3) < (3,2) = levels(n_6)$ for antecedents n_4 and n_6 of $\mathbf{F}w$. By selecting antecedents that are lexicographically smallest, H_{lex} leads us to the conflict graph shown in Figure 4. In this example, the corresponding First-UIP-Nogood, $\{\mathbf{F}p, \mathbf{T}s\}$, is weaker than $\{\mathbf{T}s\}$, which may be obtained with H_{short} (cf. Figure 3).

Given that lexicographic comparisons are computationally expensive, we also consider a lightweight variant of ranking antecedents according to decision levels. Our third heuristics, H_{avg}, prefers an antecedent δ over ε if the average of $levels(\delta)$ is smaller than the average of $levels(\varepsilon)$. In our example, we get $avg[levels(n_8)] = avg(3,1) = 2 < 3 = avg(3,3) = avg[levels(n_7)]$ and $avg[levels(n_6)] = avg(3,2) = 2.5 < 3 = avg(3) = avg[levels(n_4)]$, yielding the conflict graph shown in Figure 5. Unfortunately, the corresponding First-UIP-Nogood, $\{\mathbf{F}p, \mathbf{T}q, \mathbf{T}r\}$, does not match the goal of H_{avg} as backjumping only returns to decision level 2, where $\mathbf{T}r$ is then flipped to $\mathbf{F}r$. Note that this behavior is similar to chronological backtracking, which can be regarded as the most trivial form of backjumping.

Shortening Conflict Resolution

Our fourth heuristics, H_{res}, aims at speeding up conflict resolution itself by shortening resolution sequences. In order to earlier encounter a UIP, H_{res} prefers antecedents such that the number of literals at the current decision level dl is smallest. In our running example, H_{res} prefers n_8 over n_7 as it contains fewer literals whose decision level is 3. However, antecedents n_4 and n_6 of $\mathbf{F}w$ are indifferent, thus, H_{res} may yield either one of the conflict graphs in Figure 4 and 5.

Search Space Pruning

The heuristics presented above rank antecedents merely by structural properties, thus disregarding their contribution in the past to solving the actual problem. The latter is estimated by nogood deletion heuristics of SAT solvers (Goldberg & Novikov 2002; Mahajan, Fu, & Malik 2005), and *clasp* also maintains activity scores for nogoods (Gebser *et al.* 2007a). Our fifth heuristics, H_{active}, makes use of them and ranks antecedents according to their activities.

Finally, we investigate a heuristics, H_{prop}, that stores (and prefers) the smallest decision level at which a nogood has ever been unit-resulting. The intuition underlying H_{prop} is that the number of implied literals at small decision levels can be viewed as a measure for the progress of CDNL, in particular, as attesting unsatisfiability requires a conflict at decision level 0. Thus, it might be a good idea to prefer nogoods that gave rise to implications at small decision levels.

Experiments

For their empirical assessment, we have implemented the heuristics proposed above in a prototypical extension of our ASP solver *clasp* version 1.0.2. (Even though there are newer versions of *clasp*, a common testbed, omitting some optimizations, is sufficient for a representative comparison.) Note that *clasp* (Gebser *et al.* 2007a) incorporates various advanced Boolean constraint solving techniques, e.g.:

- lookback-based decision heuristics (Goldberg & Novikov 2002),
- restart and nogood deletion policies (Eén & Sörensson 2003),
- watched literals for unit propagation on "long" nogoods (Moskewicz *et al.* 2001),
- dedicated treatment of binary and ternary nogoods (Ryan 2004), and
- early conflict detection (Mahajan, Fu, & Malik 2005).

Due to this variety, the solving process of *clasp* is a complex interplay of different features. Thus, it is almost impossible to observe the impact of a certain feature, such as our conflict resolution heuristics, in isolation. However, we below use a considerable number of benchmark classes with different characteristics and shuffled instances, so that noise effects should be compensated at large.

For accommodating conflict resolution heuristics considering several antecedents per literal, the low-level implementation of *clasp* had to be modified. These modifications

are less optimized than the original implementation, so that our prototype incurs some disadvantages in raw speed that can potentially be reduced by optimizing the implementation. However, for comparison, we include unmodified *clasp* version 1.0.2, not applying any particular heuristics in conflict resolution. Given that unit propagation in *clasp* privileges binary and ternary nogoods, they are more likely to be used as antecedents than longer nogoods, as original *clasp* simply stores the first antecedent it encounters and ignores others. In view of this, unit propagation of original *clasp* leads conflict resolution into the same direction as H_{short}, though in a less exact way. The next table summarizes all *clasp* variants and conflict resolution heuristics under consideration, denoting the unmodified version simply by *clasp*:

Label	Heuristics	Goal
clasp	—	speeding up unit propagation
$clasp_{short}$	H_{short}	recording short nogoods
$clasp_{lex}$	H_{lex}	performing long backjumps
$clasp_{avg}$	H_{avg}	performing long backjumps
$clasp_{res}$	H_{res}	shortening conflict resolution
$clasp_{active}$	H_{active}	search space pruning
$clasp_{prop}$	H_{prop}	search space pruning

Note that all *clasp* variants perform early conflict detection, that is, they encounter a unique conflicting assignment before beginning with conflict resolution. Furthermore, all of them perform conflict resolution according to the First-UIP scheme. Thus, we do not explore the first two among the three degrees of freedom mentioned in the introductory section and concentrate fully on the choice of resolvents.

We conducted experiments on the benchmarks used in categories *SCore* and *SLparse* of the first ASP system competition (Gebser *et al.* 2007d). Tables 1–4 group benchmark instances by their classes, viz., Classes 1–11. Via superscripts ^{s} and ^{r} in the first column, we indicate whether the n instances belonging to a class are structured (e.g., 15-Puzzle) or randomly generated (e.g., BlockedN-Queens). We omit classifying Factoring, which is a worst-case problem where an efficient algorithm would yield a cryptographic attack. Furthermore, Tables 1–4 show results for computing one answer set or deciding that an instance has no answer set. For each benchmark instance, we performed five runs on different shuffles, resulting in $5n$ runs per benchmark class. All experiments were run on a 3.4GHz PC under Linux; each run was limited to 600s time and 1GB RAM. Note that, in Tables 1–3, we consider only the instances on which runs were completed by all considered *clasp* variants.

Table 1 shows the average lengths of First-UIP-Nogoods for the heuristics aiming at short nogoods, implemented by $clasp_{short}$ and $clasp_{lex}$, among which the latter uses the lengths of antecedents as a tie breaker. For comparison, we also include original *clasp*. On most benchmark classes, we observe that $clasp_{short}$ as well as $clasp_{lex}$ tend to reduce the lengths of First-UIP-Nogoods, up to 14 percent shorter than the ones of *clasp* on BlockedN-Queens. But there remains only a slight reduction of about 6 percent shorter First-UIP-Nogoods of $clasp_{lex}$ in the summary of all benchmark classes (weighted equally). We also observe that $clasp_{short}$, more straightly preferring short antecedents than $clasp_{lex}$,

No.	Class	n	$clasp_{short}$	$clasp_{lex}$	*clasp*
1^s	15-Puzzle	10	22.33	22.35	23.03
2^r	BlockedN-Queens	7	27.32	28.23	31.85
3^s	EqTest	5	172.12	178.27	189.12
4	Factoring	5	134.95	130.67	141.34
5^s	HamiltonianPath	14	12.96	11.73	12.04
6^r	RandomNonTight	14	31.82	32.07	32.74
7^r	BoundedSpanningTree	5	35.06	36.68	33.95
8^s	Solitaire	4	24.55	22.02	25.03
9^s	Su-Doku	3	16.22	15.09	13.99
10^s	TowersOfHanoi	5	52.89	52.31	58.29
11^r	TravelingSalesperson	5	101.37	90.35	99.26
Average First-UIP-Nogood Length			45.15	44.46	47.21

Table 1: Average lengths of First-UIP-Nogoods per conflict.

No.	Class	n	$clasp_{avg}$	$clasp_{lex}$	*clasp*
1^s	15-Puzzle	10	2.12	2.14	2.10
2^r	BlockedN-Queens	7	1.07	1.08	1.07
3^s	EqTest	5	1.03	1.04	1.03
4	Factoring	5	1.20	1.21	1.20
5^s	HamiltonianPath	14	2.53	2.58	2.62
6^r	RandomNonTight	14	1.15	1.16	1.15
7^r	BoundedSpanningTree	5	3.12	3.47	3.06
8^s	Solitaire	4	3.34	3.28	2.92
9^s	Su-Doku	3	2.55	3.01	2.76
10^s	TowersOfHanoi	5	1.46	1.46	1.40
11^r	TravelingSalesperson	5	1.27	1.51	1.43
Average Backjump Length			1.89	1.99	1.89

Table 2: Average backjump lengths per conflict.

does not reduce First-UIP-Nogood lengths any further. Interestingly, there is no clear distinction between structured and randomly generated instances, neither regarding magnitudes nor reduction rates of First-UIP-Nogood lengths.

Table 2 shows the average backjump lengths in terms of decision levels for the *clasp* variants aiming at long backjumps, viz., $clasp_{avg}$ and $clasp_{lex}$. We note that average backjump lengths of more than 2 decision levels indicate structured instances, except for BoundedSpanningTree. Regarding the increase of backjump lengths, $clasp_{avg}$ does not exhibit significant improvements, and the polarity of differences to original *clasp* varies. Only the more sophisticated heuristics of $clasp_{lex}$ almost consistently leads to increased backjump lengths (except for HamiltonianPath), but the amounts of improvements are rather small.

Table 3 shows the average numbers of conflict resolution steps for $clasp_{res}$ and $clasp_{lex}$, among which the former particularly aims at their reduction. Somewhat surprisingly, $clasp_{res}$ in all performs more conflict resolution steps even than original *clasp*, while $clasp_{lex}$ almost consistently exhibits a reduction of conflict resolution steps (except for Su-Doku). This negative result for $clasp_{res}$ suggests that trimming conflict resolution regardless of its outcome is not advisable. The quality of recorded nogoods certainly is a key factor for the performance of conflict-driven learning solvers for ASP and SAT, thus, shallow savings in their retrieval are not worth it and might even be counterproductive globally.

No.	Class	n	*clasp*$_{short}$	*clasp*$_{lex}$	*clasp*$_{avg}$	*clasp*$_{res}$	*clasp*$_{active}$	*clasp*$_{prop}$	*clasp*
1^s	15-Puzzle	10	195.00	203.96	203.54	248.00	261.44	226.96	241.18
			0.13	0.14	0.14	0.15	0.16	0.15	0.14
2^r	BlockedN-Queens	7	27289.06	26989.57	28176.00	27553.63	30240.71	29119.60	28588.34
			116.87 (24)	122.04 (21)	39.70 (27)	86.24 (24)	138.01 (22)	68.10 (25)	24.52 (22)
3^s	EqTest	5	62430.92	62648.96	59330.52	62705.00	62374.84	63303.44	62290.76
			19.47	21.66	19.41	19.98	20.03	21.30	15.66
4	Factoring	5	15468.44	14838.64	14985.72	16016.56	16365.52	15404.64	16920.68
			6.30	5.85	6.27	6.55	6.36	6.36	5.11
5^s	HamiltonianPath	14	703.70	683.29	653.19	564.83	764.16	694.33	650.70
			0.05	0.05	0.05	0.04	0.06	0.05	0.05
6^r	RandomNonTight	14	427031.71	411024.73	402846.21	429955.23	423332.74	405476.81	406007.41
			53.85	55.17	51.53	54.92	53.33	52.78	41.79
7^r	BoundedSpanningTree	5	879.92	640.88	801.76	634.96	662.22	940.92	949.84
			4.51	4.37	4.38	4.36	4.27	4.98	4.42
8^s	Solitaire	4	193.85	145.85	103.40	134.75	103.40	95.90	134.00
			66.14 (2)	0.22 (5)	30.81 (4)	0.22 (5)	0.21 (5)	0.21 (5)	0.23 (4)
9^s	Su-Doku	3	123.40	127.80	164.60	111.93	108.67	119.87	123.93
			18.89	19.85	19.75	19.10	19.39	19.77	19.96
10^s	TowersOfHanoi	5	145064.20	124222.96	71220.52	140386.64	97411.80	134192.96	133760.48
			62.43	46.69	21.86	52.19	32.76	47.63	37.60
11^r	TravelingSalesperson	5	2512.20	1018.80	3243.16	2535.40	1334.32	2500.16	947.56
			34.06	21.63	42.22	36.77	25.70	34.42	20.89
Average Number of Conflicts			56824.37	53545.45	48477.39	56737.24	52748.20	54339.63	54217.91
Average Time (Sum Timeouts)			31.89 (26)	24.81 (26)	19.68 (31)	23.38 (29)	25.02 (27)	21.31 (30)	14.20 (26)
Average Penalized Time			49.25	46.75	45.28	48.05	47.12	47.14	37.27

Table 4: Average numbers of conflicts and runtimes.

No.	Class	n	*clasp*$_{res}$	*clasp*$_{lex}$	*clasp*
1^s	15-Puzzle	10	102.95	103.45	103.77
2^r	BlockedN-Queens	7	18.17	17.61	17.74
3^s	EqTest	5	86.94	84.78	85.76
4	Factoring	5	325.54	290.36	296.07
5^s	HamiltonianPath	14	11.87	12.03	12.14
6^r	RandomNonTight	14	16.41	16.47	32.74
7^r	BoundedSpanningTree	5	20.11	20.27	20.66
8^s	Solitaire	4	79.05	67.70	79.89
9^s	Su-Doku	3	21.48	20.86	19.73
10^s	TowersOfHanoi	5	41.60	40.36	42.69
11^r	TravelingSalesperson	5	141.68	96.06	122.98
Average Number of Resolution Steps			78.71	70.00	75.83

Table 3: Average numbers of resolution steps per conflict.

Finally, Table 4 provides average numbers of conflicts and average runtimes in seconds for all *clasp* variants. For each benchmark class, the first line provides the average numbers of conflicts encountered on instances where runs were completed by all *clasp* variants, while the second line gives the average times of completed runs and numbers of timeouts in parentheses. (Recall that all *clasp* variants were run on $5n$ shuffles of the n instances per class, leading to more than n timeouts on BlockedN-Queens and, with some *clasp* variants, also on Solitaire.) At the bottom of Table 4, we summarize average numbers of conflicts and average runtimes over all benchmark classes (weighted equally). Note that the last but one line provides the sums of timeouts in parentheses, while the last line penalizes timeouts with maximum time, viz., 600 seconds. As mentioned above, original *clasp* is highly optimized and does not suffer from the overhead incurred by the extended infrastructure for applying heuristics in conflict resolution. As a consequence, we observe that original *clasp* outperforms its variants on most benchmark classes as regards runtime. Among the variants of *clasp*, *clasp*$_{avg}$ in all exhibits the best average number of conflicts and runtime. However, it also times out most often and behaves unstable, as the poor performance on Classes 2 and 11 shows. In contrast, *clasp*$_{short}$ and *clasp*$_{lex}$ lead to fewest timeouts (in fact, as many timeouts as *clasp*), and *clasp*$_{lex}$ encounters fewer conflicts than *clasp*$_{short}$. Variant *clasp*$_{active}$, preferring "critical" antecedents, exhibits a comparable performance, while *clasp*$_{res}$ and *clasp*$_{prop}$ yield more timeouts and also encounter relatively many conflicts. Overall, we notice that some *clasp* variants perform reasonably well, but without significantly decreasing the number of conflicts in comparison to original *clasp*. As there is no clear winner among our *clasp* variants, unfortunately, they do not suggest any "universal" conflict resolution heuristics.

Discussion

We have proposed a number of heuristics for conflict resolution and conducted a systematic empirical study in the context of our ASP solver *clasp*. However, it is too early to conclude any dominant approach or to make general recommendations. As has also been noted in (Mitchell 2005), conflict resolution strategies are almost certainly important but have received little attention in the literature so far. In fact, dedicated approaches in the SAT area (Ryan 2004; Mahajan, Fu,

& Malik 2005) merely aim at reducing the size of recorded nogoods. Though this might work reasonably well in practice, it is unsatisfactory when compared to sophisticated decision heuristics (Goldberg & Novikov 2002; Ryan 2004; Mahajan, Fu, & Malik 2005; Dershowitz, Hanna, & Nadel 2005) resulting from more profound considerations. We thus believe that heuristics in conflict resolution deserve further attention. Future lines of research may include developing more sophisticated scoring mechanisms than the ones proposed here, combining several scoring criterions, or even determining and possibly recording multiple reasons for a conflict (corresponding to different conflict graphs). Any future improvements in these directions may significantly boost the state-of-the-art in both ASP and SAT solving.

References

Baral, C.; Brewka, G.; and Schlipf, J., eds. 2007. *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Springer-Verlag.

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

Bayardo, R., and Schrag, R. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, 203–208. AAAI Press/MIT Press.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22:319–351.

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

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers.

Dershowitz, N.; Hanna, Z.; and Nadel, A. 2005. A clause-based heuristic for SAT solvers. In Bacchus, F., and Walsh, T., eds., *Proceedings of the Eigth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, 46–60. Springer-Verlag.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, 502–518.

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *Theory and Practice of Logic Programming* 3(4-5):499–518.

Fages, F. 1994. Consistency of Clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science* 1:51–60.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007a. clasp: A conflict-driven answer set solver. In Baral et al. (2007), 260–265.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007b. Conflict-driven answer set enumeration. In Baral et al. (2007), 136–148.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007c. Conflict-driven answer set solving. In Veloso, M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 386–392. AAAI Press/MIT Press.

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczyński, M. 2007d. The first answer set programming system competition. In Baral et al. (2007), 3–17.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning* 36(4):345–377.

Goldberg, E., and Novikov, Y. 2002. BerkMin: A fast and robust SAT solver. In *Proceedings of the Fifth Conference on Design, Automation and Test in Europe (DATE'02)*, 142–149. IEEE Press.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In Kaelbling, L., and Saffiotti, A., eds., *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, 503–508. Professional Book Center.

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.

Mahajan, Y.; Fu, Z.; and Malik, S. 2005. Zchaff2004: An efficient SAT solver. In Hoos, H., and Mitchell, D., eds., *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 360–375. Springer-Verlag.

Marques-Silva, J., and Sakallah, K. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Mitchell, D. 2005. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science* 85:112–133.

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, 530–535. ACM Press.

Ryan, L. 2004. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University.

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

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

Zhang, L.; Madigan, C.; Moskewicz, M.; and Malik, S. 2001. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, 279–285.

A Versatile Intermediate Language for Answer Set Programming

Martin Gebser [1] and **Tomi Janhunen** [2] and **Max Ostrowski** [1] and **Torsten Schaub** [1] and **Sven Thiele** [1]
[1] Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam, Germany
[2] Helsinki University of Technology, Department of Information and Computer Science, P.O. Box 5400, FI-02015 TKK, Finland

Abstract

The attractiveness of Answer Set Programming (ASP) and related paradigms for declarative problem solving is considerably due to the availability of highly efficient yet easy-to-use implementations. A major driving force for the development and improvement of tools are standardized problem representations, for several reasons. First, they relieve developers from the burden of inventing their own input formats. Second, they establish interoperability between separate tools, allowing users to easily compare and exchange them without extensively converting their problem representations. Third, they facilitate the acquisition of problem descriptions from distinct sources, which is useful for benchmarking and assessment purposes. Historically, however, standards for representing logic programs, serving as inputs to ASP systems, were mainly dictated by the few available tools. In fact, there currently are two quasi standards, namely, the formats used by *lparse* and *dlv*, incompatible with each other. As a first step towards overcoming this deficiency, this work proposes an intermediate format for ground logic programs, intended for the representation of inputs to ASP solvers. The format is not designed to be a primary input language, given that ASP systems usually deploy a second component, called a grounder, to deal with the inputs provided by users. In view of this, our format is situated intermediate a grounder and a solver, guided by the example of grounder *lparse* and solver *smodels*, the latter marking the first among nowadays a variety of solvers processing the output of *lparse*. However, the output format of *lparse* has some decisive drawbacks, namely, its restrictive range and limited extensibility. We thus propose a new intermediate language, where our major design goals are flexibility in problem representation and easy extensibility to new language constructs.

Introduction

Answer Set Programming (ASP; (Baral 2003; Gelfond & Leone 2002; Marek & Truszczyński 1999; Niemelä 1999)) is a declarative approach to modeling and solving search problems, represented as logic programs. As illustrated in Figure 1, an ASP system usually consists of two components, a *grounder* and a *solver*. The input to an ASP system usually consists of a non-ground problem encoding and a ground problem instance. In such *uniform* encodings, the use of first-order variables reduces size and permits simpler, and therefore easier to write, logic programs. Furthermore, regarding the input language, several extensions have been proposed, like aggregates, cardinality and weight constraints, and optimize statements (Dell'Armi *et al.* 2003; Leone *et al.* 2006; Simons, Niemelä, & Soininen 2002). A grounder translates such a problem representation (typically a pair of an encoding and an instance) from the input language into a ground logic program, represented in a simplified, solver-readable form. Starting from a grounder's output, a solver then searches for answer sets, corresponding to solutions of the original problem. The most common solving approaches are based on the Davis-Putnam-Logemann-Loveland (DPLL; (Davis, Logemann, & Loveland 1962; Davis & Putnam 1960)) algorithm, like in *dlv* (Leone *et al.* 2006) and *smodels* (Simons, Niemelä, & Soininen 2002), or Conflict-Driven Clause Learning (CDCL; (Marques-Silva & Sakallah 1999; Mitchell 2005; Moskewicz *et al.* 2001)), e.g., used in *clasp* (Gebser *et al.* 2007a).

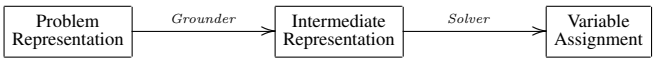


Figure 1: Basic Architecture of an ASP System

There currently is a single intermediate language accessible to ASP solvers, namely, the output format of grounder *lparse* (Syrjänen).[1] However, the format is not standardized and might thus change over different *lparse* versions, which is a delicate issue since no version information is included, e.g., for backward compatibility. The latter also makes ad hoc extensions of *lparse*'s output format intricate and error-prone.[2] Furthermore, the fact that *lparse*'s output format is designed to match *smodels*' internal data structures necessitates program transformations incurring a loss of structural information (Liu & Truszczyński 2005). We thus consider the restrictedness, on the one hand, and the limited extensibility, on the other hand, of *lparse*'s output format as serious drawbacks, making it unsuitable as a general standard.

This work proposes a new intermediate format, called *ASPils* ("ASP intermediate language standard"), for the use in-between grounders and solvers. Important design goals are:

[1] The *dlv* system uses an internal grounder that is directly coupled with the solver.

[2] For instance, the tool *dencode* (Janhunen) includes functionality for making conversions between the disjunctive rule output formats of old (up to version 1.0.14) and new versions of *lparse*.

- simplicity and efficiency in outputting and parsing;
- independence of grounder and solver implementations;
- support of the existing (input) language constructs (cf. (Dell'Armi *et al.* 2003; Leone *et al.* 2006) & (Syrjänen));
- support of version information, meta-information, and user comments; and
- flexibility and easy extensibility.

The development of *ASPils* is inspired by experiences made in related fields, such as Boolean Satisfiability (SAT), having standardized problem description languages, e.g., DIMACS format (DIMACS 1993).[3] For one, such standardized languages establish interoperability between solvers and further tools, for instance, tools generating solver inputs. As a concrete example in ASP, a standardized intermediate language might enable (arbitrary) solvers to process the output of *dlv*'s grounding component, and also *dlv*'s solving component to process the output of an external grounder. From a user's point of view, interoperability facilitates running different solvers, as a problem encoding written in the input language of a particular grounder could after grounding be processed by an arbitrary solver. Furthermore, a standardized intermediate language supported by all solvers would greatly foster ASP solver competitions, where in the past the different input formats of *dlv* and other solvers have been a major bottleneck (Gebser *et al.* 2007b). In fact, as a secondary benefit, a common language eases collecting challenging benchmarks from distinct sources and might thus push the further development of ASP solvers, like it has been experienced in SAT. To this end, this paper introduces *ASPils* and illustrates its potential usage on examples; full details are provided in (Gebser *et al.* 2008a).

In order to put this document in perspective, let us stress that *ASPils* is proposed as a standard for the transmission of ground logic programs from grounders to solvers. At this stage, our proposal aims at recording the language constructs currently supported by *lparse*-based solvers as well as *dlv*'s solving component and at integrating them into a common framework, also anticipating future extensions to a certain extent. Of course, an input format for ASP solvers can only turn into a standard if it is widely supported and used in practice, which requires a community effort, especially, from ASP system developers. Our proposal of *ASPils* thus aims at providing a starting point for a community-wide discussion of a standardized input format for ASP solvers. Even if such a standard is successfully established, it will not instantly abolish all differences and peculiarities of ASP systems. For instance, grounders and integrated ASP systems may further (have to) use proprietary input languages, and any modular (Oikarinen & Janhunen 2006), incremental (Gebser *et al.* 2008b), or even a system not following the computational pattern shown in Figure 1 might not be readily supplied with an appropriate input format. However, before succeeding to standardize the simplest element in the workflow of ASP systems, namely, the intermediate language used in-between a grounder and a solver, any attempts to standardize more diversified matters would most likely be prone to fail. Hence, even though the scope of *ASPils* is limited to an intermediate representation according to Figure 1, we think that it may initiate a worthwhile discussion on language standards in ASP.

[3] See (Janhunen 2007; Gebser *et al.* 2008a) for detailed discussions of intermediate formats.

General Design of *ASPils*

We now briefly describe the design decisions underlying *ASPils*, the new intermediate language for ASP we propose here. Our global goal is to specify a language that has the potential to become a standard format for inputs to ASP solvers. Thus, we have to respect that different ASP solvers support different language constructs, e.g., *dlv* deals with aggregates (Dell'Armi *et al.* 2003) and *smodels* with extended rules (Simons, Niemelä, & Soininen 2002). In order to reflect this diversity, our language must be general and solver-independent. Furthermore, it is impossible to foresee language constructs that might evolve in the future. Hence, extensibility of the language is an important issue. We thus include a *version number* in problem descriptions of *ASPils*. In addition, *normal forms* are used to specify language fragments. Their main purpose is to reflect different capabilities of solvers, which are thus enabled to check whether a problem description is appropriate before processing it further.

The body of *ASPils* consists of *entries*, mainly defining *objects* of particular *types*. The idea is similar to the output format of *lparse* (Syrjänen), using several rule types. However, *ASPils* goes further than *lparse* by not restricting types to rules, rather, all entities in a ground logic program, e.g., atoms, conjunctions, disjunctions, etc., are objects having a type. An advantage of this is that complex structures occurring in a program, e.g., conjunctions of cardinality and weight constraints, can be represented in a modular and structure-preserving way. In contrast, *lparse* would have to introduce new atoms and rules to represent complex structures in its restrictive output format. Another advantage of types is the easy embedding of new language constructs, as it only requires the definition of a type identifier and a syntax for objects of the type. To avoid clashes of custom type identifiers, we use numbers consisting of a "major" and a "minor" slot (similar to IP addresses), where the major slot ought to be related to a research group defining the type. If newly introduced types turn into a standard, they can be integrated into *ASPils* via an additional normal form or even a new language version.

As mentioned above, every *object* occurring in a problem description has an associated type. In addition, each object has a unique ID, that is, a positive integer, to refer to the object. This makes the language modular because, on the syntactic level, other objects make use only of the ID of an referenced object, but not of its internal structure. Hence, if new language constructs are introduced, their objects can immediately be used within available structures, without needing to exchange them. A second potential benefit of object IDs is the possibility to re-use them if the same object has multiple occurrences in a ground logic program, thus compacting the representation. Note that this accounts

for ordinary propositional atoms as well as for non-atomic structures, such as conjunctions and disjunctions.

On the technical level, our language shall be easy to parse, independent of system environments, and resistant against potential parsing errors. To this end, we use a numerical text format and number `0` as an explicit delimiter for entries. As `0` can also occur within an entry (not as delimiter), each entry must specify its number of consecutive (numeric and/or symbolic) parameters directly after its type. This shall enable the correct syntactic decomposition of *ASPils* sentences, even without recognizing the contents, and it deliberately introduces a layer of redundancy in order to avoid parsing errors. Also note that most parameters occurring in entries are numeric and thus represented by integers, which should facilitate their recognition by solvers. In particular, negative integers are used to denote the default negations of objects having the absolute values as their IDs. Of course, the use of numbers makes *ASPils* less human-readable, but human-readability is not one of our design goals anyway. Symbolic information can still be included, most likely, for defining atom names, but usually such information needs not be interpreted by solvers. Furthermore, arbitrary meta-information as well as user comments can be provided using dedicated types. Note that comments are the only kind of objects not having an ID, as they ought to be ignored by solvers. In contrast, meta-information may be exploited by solvers, but it should not be mandatory for solvers to recognize it. We do not suggest any kind of meta-information, but information like whether a logic program is tight (Fages 1994; Erdem & Lifschitz 2003) or whether it has an answer set might be useful for particular purposes.

Language Description

This section describes the elementary constituents of our proposed intermediate language *ASPils*, where we focus on intuitions and examples. The formal specification of *ASPils* can be found in (Gebser *et al.* 2008a). Below, *italic* and `typewriter` fonts indicate non-terminals and terminals, respectively, in the grammar of *ASPils*.

Header

Every sentence of *ASPils* starts with a *header*.[4] E.g., header

```
1 3 1 3 0 0
```

consists of a `1` indicating the *type header*, a `3` providing the number of parameters before the delimiter, the second `1` stating that this is the first *version* of *ASPils*, the second `3` indicating conformance to *normal form* "SModels" (introduced below), a `0` stating that there are no *additional headers*, and the second `0` delimiting the header. Note that language version `1` of *ASPils*, defined in (Gebser *et al.* 2008a), does not specify any additional headers, hence, their number will always be `0`. The pattern that a type number is followed by the number of parameters before delimiter `0` recurs in each of the types described below, while the parameters themselves are specific.

[4] Only *comment*s are allowed before *header* and after *object eof*.

End Of File

Every sentence of *ASPils* is terminated with an occurrence of *object eof*,[4] looking as follows:

```
0 0 0 .
```

The first `0` indicates the object's type, the second its number of parameters, and the third one delimits it. (The full stop sign "." is part of the surrounding text, but not of *ASPils*.)

Entries

In-between the header and end of file, a ground logic program is specified by entries, providing meta-information, comments, or defining elements of the program at hand. Each *entry* starts with its type number, an up to eight digits long hexadecimal cipher. The first four digits denote the research group that has developed the type; for the core types described below, the four leading digits are zeros and can thus be omitted. The last four digits of a type number must not evaluate to zero (reserved for *object eof*). Each type number is followed by the number of consecutive parameters before the *entry* is delimited by an occurrence of `0`. Each entry type imposes particular parameters, that is, the slots specified in the grammar (Gebser *et al.* 2008a) must be filled with terminals.

Meta-Information. Objects of type `2` can be used to provide meta-information. Although we do not suggest any particular meta-object, the following one is syntactically valid:

```
2 3 42 "tight program" 23 0 .
```

Among the `3` parameters of the entry, `42` is the *object ID*, `"tight program"` is a *safe verbal*, that is, a list of strings and whitespaces enclosed in double quotes, and `23` is the single element of a list of *meta-option*s (that is, *integer*s). Meta-information may be exploited by solvers, but any such information should not affect the semantics of the specified ground logic program, so that it is admissible for solvers to simply ignore unrecognized meta-objects.

Comments. Comments of type `3` do not define any object (i.e., they do not have an *object ID* as parameter). An example comment looks as follows:

```
3 1 "grounded by GrinGo version 2" 0 .
```

As *comment*s are not associated with an ID, they cannot be referenced by objects defined in an *ASPils* problem description. In fact, comments are understood to be completely up to user information, such as the author of a problem description or the grounder that generated it. Unlike meta-information, comments must always be ignored by solvers.

Atoms. Objects of type `4` define atoms. E.g., consider:

```
4 2  8   p(a,1)         0
4 5 15 "-p(a,1)" 1 2 8 0 .
```

The first entry specifies that *object ID* `8` stands for an *atom* whose name is `p(a,1)`, and the second entry defines *ob-*

ject ID `15` to represent another atom with name `-p(a,1)`.[5] Furthermore, *atom option* `1`, provided for `-p(a,1)`, declares the atom as hidden, that is, the atom name shall be suppressed in the output of an ASP solver. We include this option in order to reflect the effect of hide declarations in the input language of *lparse* (Syrjänen). However, while *lparse* suppresses atom names by not including them in the symbol table, we choose to keep the symbolic names of atoms and to signal their hidden nature via an option. In this way, it stays possible to recover a symbolic representation from the intermediate format, as it is done by tool *lplist* (Janhunen) for *lparse*'s output format using the symbolic information still available there. The second *atom option* `2` for `-p(a,1)` declares the atom to be the classical negation of the object with ID `8`, viz., of atom `p(a,1)`. Classical negation is understood in the sense of (Gelfond & Lifschitz 1991). In our example, it means that atoms `p(a,1)` and `-p(a,1)` cannot jointly belong to a stable model of the program at hand.

Rules, Facts, and Integrity Constraints. In ASP, a logic program is a set of rules, each rule consisting of a head and a body. Either the head or the body may be constant, in which case the rule is called a fact or an integrity constraint, respectively. Let us consider the following example logic program containing a rule, a fact, and an integrity constraint:

```
-p(a,1) :- not p(a,1).
 p(a,1).
        :- -p(a,1).
```

Reusing *object ID*s `8` and `15` for `p(a,1)` and `-p(a,1)`, respectively, corresponding entries in *ASPils* are as follows:

```
5 3 55 15 -8 0
6 2 66  8    0
7 2 77    15 0 .
```

Here, type numbers `5`, `6`, and `7` indicate that the objects with IDs `55`, `66`, and `77` are a *rule*, a *fact*, and an *integrity constraint*, respectively. Note that `-8` in the first entry refers to the default negation of the atom with ID `8`, viz., of atom `p(a,1)`. Furthermore, observe that the second entry specifies only a head *literal* and the third one only a body *literal*, while the rule defined by the first entry contains both a head and a body *literal*. The choice of introducing three different types is motivated by the goal of not imposing any hard-wired assumptions on the structure of heads and bodies, while still being able to identify the role of particular *literal*s. Importantly, workarounds such as the `_false` atom, introduced by *lparse* (Syrjänen) as the head of integrity constraints, ought to be avoided. Due to the generic design of entries for rules, facts, and integrity constraints, allowing to refer to arbitrary and possibly default negated objects, there are no restrictions on the structure of heads and bodies (rules might even reference each other) a priori. Below, the issue of ensuring certain formats is dealt with via normal forms.

[5]The name of the second atom must be provided as a *safe verbal*, enclosed in double quotes. Double quotes may only be omitted for *atom name*s starting with a *letter*. Due to this requirement, the first symbols of *atom name*s and *integer*s become unambiguous.

Conjunctions and Disjunctions. In order to express more complex rules than the ones given above, entries may specify conjunctions and disjunctions of *literal*s. For instance,

```
8 3 89 -8 -15 0
```

defines an object with ID `89` as the *conjunction* of the default negations of the objects with IDs `8` and `15`. Similarly,

```
9 3 98 8 15 0
```

defines a *disjunction* of the objects with IDs `8` and `15`. Assuming that ID `8` stands for atom `p(a,1)` and `15` for `-p(a,1)`, the entries

```
7 2 78 89 0
6 2 67 98 0
```

describe the following integrity constraint and disjunctive fact:

```
:- not p(a,1), not -p(a,1).
p(a,1) | -p(a,1).
```

Finally, note that the normal forms described below restrict conjunctions to occur in bodies of rules and disjunctions to being used in rule heads.

Default Negation. Programs in "canonical form" (Lee 2005; Lifschitz, Tang, & Turner 1999) permit double (default) negation of atoms. Rather than permitting multiple occurrences of "−" at the beginning of a *literal* (which would make *literal*s and *integer*s syntactically different), we introduce entries defining default negation objects for representing nested negation. This allows us to express a "choice"

```
p(a,1) :- not not p(a,1).
```

in terms of the following entries:

```
a 2 44   -8 0
5 3 45 8 44 0 .
```

Observe that the object defined by the first entry stands for the *default negation* of *literal* `-8` and, thus, for the double negation of the object with ID `8`, viz., of atom `p(a,1)`. Finally, note that, under answer set semantics, rules using an atom and those using its double negation are not necessarily equivalent (Lifschitz, Tang, & Turner 1999), so that double negation constitutes a proper syntactic feature.

Cardinality and Weight Constraints. Cardinality and weight constraints (Simons, Niemelä, & Soininen 2002; Syrjänen) permit expressing conditions on sets of literals, that is, true literals can be counted or their weights can be summed up in order to compare the result against a lower and an upper bound. Letting *object ID*s `1`, `2`, `3`, and `4` stand for atoms `a`, `b`, `c`, and `d`, we can represent the expressions

```
2{a, b, not c, not d}3
-1[a=-2, b=1, not c=3, not d=-4]2
```

in *ASPils* as follows:

```
b  7 5  2 3 1 2 -3 -4           0
c 11 6 -1 2 1 2 -3 -4 -2 1 3 -4 0 .
```

Here, the *object ID*s `5` and `6` of the *cardinality* and *weight constraint*, respectively, are immediately followed by their lower and upper bounds. Afterwards, the *literal*s are provided, and for weight constraint `6`, also a list of *weight*s (exactly one weight per literal). The primary constituents of a

logic program, viz., *rule*s, *fact*s, and *integrity constraint*s, can incorporate *cardinality* and *weight constraint*s just like *atom*s, simply by referencing their IDs directly or indirectly through literals. Observe that, in general, we allow for upper bounds as well as for negative weights, both of which can occur in the input language, but not in the output language, of *lparse*. In fact, *lparse* performs a number of transformations and introduces new atoms to remove them. Some of the normal forms below impose similar restrictions, thus, specifying *lparse*-like fragments of *ASPils*. In such fragments, only *trivial upper bounds* are permitted, obtained by summing up all (positive) weights, where weight `1` is used for the literals of cardinality constraints.

Weighted Literals. *Weighted literal*s are auxiliary concepts, devised for the use with aggregates and optimization statements (see below), thus, establishing a uniform way of referencing objects (simple or complex ones) evaluating to numbers. For instance, we may associate *weight*s to *literal*s, as in the above weight constraint, via the following entries:

```
d 3 11  1 -2 0
d 3 12  2  1 0
d 3 13 -3  3 0
d 3 14 -4 -4 0 .
```

Aggregates. We adopt the aggregates supported by *dlv* (Dell'Armi *et al.* 2003), allowing for five operations, viz., *count*, *sum*, *max*, *min*, and *times*. While *count* applies to Boolean operands, that is, to *literal*s, the other four aggregates operate on numerical values, thus, they require *object ID*s rather than *literal*s as parameters. Reusing atoms `a`, `b`, `c`, and `d` as well as weighted literals as specified above, we may define a *count* and a *sum* aggregate as follows:

```
e 5 21  1  2 -3 -4 0
f 5 22 11 12 13 14 0 .
```

Observe that *count* applies to (possibly negative) *literal*s, while *dlv*'s aggregates are restricted to atoms. As such a restriction does not significantly simplify dealing with aggregates, we do not adopt it here, and the *weighted literal*s used by *sum* may also apply to negative *literal*s (as it is the case for the objects with IDs `13` and `14`). However, in order to reasonably apply an aggregate, operands must have appropriate types, being an issue to the normal forms below.

Operators. Arithmetic comparison operators can be applied to *weighted literal*s and *aggregate*s in order to retrieve Boolean values from them. Thus, *operator*s can be referenced by *rule*s, *fact*s, and *integrity constraint*s in the same way as *atom*s. We provide two kinds of operators: (binary) *operator*s of type in-between `13` and `17` can be used to compare the numerical values of two *object*s with one another, while *unary operator*s of type in-between `18` and `1c` allow for comparing an *object*'s numerical value to an *integer*. Both kinds of *operator*s support the following comparison operations: *eq* ("equal"), *leq* ("less or equal"), *lt* ("less than"), *geq* ("greater or equal"), and *gt* ("greater than"). For instance, the following entry describes the application of the (binary) *operator eq* to the aggregates with IDs `21` and `22` as defined above:

```
13 3 31 21 22 0 .
```

An application of *unary operator leq* to the aggregate with ID `22` and integer `0` can be specified as follows:

```
19 3 91 22 0 0 .
```

Finally, note that current ASP solvers do not support (binary) *operator*s of type in-between `13` and `17`, hence, they will not be permitted by the normal forms below.

Optimization. Amongst the most common optimization techniques in ASP are "minimize statements" (Simons, Niemelä, & Soininen 2002) supported by *smodels* and "weak constraints" (Leone *et al.* 2006) supported by *dlv*. In order to reflect them, we introduce an *optimize* object whose underlying *strategy* is minimization of (arbitrarily many) objective functions of distinct priorities, the priorities forming a strict total order. Other strategies than lexicographic ordering of objective functions, e.g., Pareto optimality, would also be possible but are currently not in use, so we do not (yet) consider them. In what follows, we detail how "minimize statements" and "weak constraints" can be represented in *ASPils*.

Minimize Statements of *smodels*: Assume that an input program provided to *lparse* contains two minimize statements (in order):

```
minimize[not a, b, c].
minimize[a=4, not b=3, c=2].
```

The first statement expresses that a minimum number of its literals should be true, while the second one is about minimizing the sum of weights of true literals. The corresponding objective functions are expressed by a *count* and a *sum* aggregate (over *weighted literal*s), respectively, where we assume IDs `1`, `2`, and `3` for atoms `a`, `b`, and `c`:

```
d 3 11  1  4    0
d 3 12 -2  3    0
d 3 13  3  2    0
e 4 21 -1  2  3 0
f 4 22 11 12 13 0 .
```

Note that the *count* aggregate with ID `21` gives the objective function minimized by the first statement over literals, and the *sum* aggregate with ID `22` takes the weights provided in the second minimize statement into consideration. We are now ready to define a single *optimize* object that incorporates both aggregates:

```
1d 4 43 1e 22 21 0 .
```

The type of this object is `1d`, `4` the number of parameters, and `43` the ID. Furthermore, `1e` specifies the lexicographic optimization strategy, which is the only *strategy* included in the first version of *ASPils*. Finally, minimizing the numerical value of the *sum* aggregate with ID `22` takes higher priority than minimizing the value of the *count* aggregate with ID `21`. This priority, reverse to the order of minimize statements in the input, is indeed applied by *smodels*. While *smodels* derives the (reverse) priorities of minimize statements implicitly from the order in the input, in the *ASPils* representation, priorities are explicit because the objective functions to be minimized are combined within an *optimize* object.

Weak Constraints of *dlv*: We start with an example. Consider the following weak constraints:

```
:~ a, not b. [1:2]
:~ not c. [1:1]
:~ b, c. [2:1]
```

Note that the numbers in brackets describe weights and levels. As level `2` of the first weak constraint is greater than `1`, the first weak constraint is of higher priority than the second and the third one. Among the last two weak constraints, the priority of the third one is greater simply because its weight `2` is greater than `1`. In order to express the non-singleton bodies of weak constraints, we define *conjunction*s as follows:

```
8 3 81 1 -2 0
8 3 83 2  3 0 .
```

The *conjunction* with ID `81` stands for the body of the first weak constraint, and the one with ID `83` for the body of the third weak constraint. We can now proceed by defining *weighted literal*s, one per weak constraint:

```
d 3 11 81 1 0
d 3 21 -3 1 0
d 3 22 83 2 0 .
```

Observe that the *weight* of each weak constraint is the weight given in the input. Finally, we use a *sum* aggregate for multiple weak constraints at the same level and define an *optimize* object as follows:

```
f  3 24    21 22 0
1d 4 35 1e 11 24 0 .
```

The last entry expresses that we minimize weights starting with the weak constraint at level `2` and, secondarily, for the weak constraints at level `1`.

We now provide a general scheme of how to represent multiple weak constraints of distinct levels in *ASPils*. Consider the following weak constraints ordered by their levels l_j, where we assume $l_i > l_j$ if $i < j$ in order to respect level priorities:

$$\begin{array}{cccc} \texttt{:}\sim\ body_{1_1}.\ [w_{1_1} : l_1] & \ldots & \texttt{:}\sim\ body_{n_1}.\ [w_{n_1} : l_1] \\ & \vdots & \\ \texttt{:}\sim\ body_{1_m}.\ [w_{1_m} : l_m] & \ldots & \texttt{:}\sim\ body_{n_m}.\ [w_{n_m} : l_m]\,. \end{array}$$

In *ASPils*, the bodies $body_{1_1}, \ldots, body_{n_m}$ of weak constraints can be defined by *literal*s either over *atom*s (for singleton bodies) or over *conjunction*s, which can be defined as usual. Thus, we keep notation $body_{i_j}$ in the following *weighted literal*s:

$$\begin{array}{l} \texttt{d 3}\ x_{1_1}\ \ body_{1_1}\ \ w_{1_1}\ \ \texttt{0} \\ \vdots \\ \texttt{d 3}\ x_{n_1}\ \ body_{n_1}\ \ w_{n_1}\ \ \texttt{0} \\ \vdots \\ \texttt{d 3}\ x_{1_m}\ \ body_{1_m}\ \ w_{1_m}\ \ \texttt{0} \\ \vdots \\ \texttt{d 3}\ x_{n_m}\ \ body_{n_m}\ \ w_{n_m}\ \ \texttt{0}\,. \end{array}$$

The *weighted literal*s defined above are similar to those used in the representation of minimize statements. In fact, it makes no difference in *ASPils* whether they refer to *atom*s or to *conjunction*s. The next step of adding level-wise *sum* aggregates and a single *optimize* object is similar to minimize statements:

$$\begin{array}{l} \texttt{f}\ n_1{+}1\ \ s_1\ \ \ \ x_{1_1}\ \ldots\ x_{n_1}\ \ \texttt{0} \\ \vdots \\ \texttt{f}\ n_m{+}1\ \ s_m\ \ \ \ x_{1_m}\ \ldots\ x_{n_m}\ \ \texttt{0} \\ \texttt{1d}\ m{+}2\ \ o\ \ \texttt{1e}\ \ s_1\ \ldots\ s_m\ \ \texttt{0}\,. \end{array}$$

The given embeddings in *ASPils* indicate that minimize statements and weak constraints are handled likewise, using *weighted literal*s, *sum* aggregates (sometimes, simpler constructs are sufficient), and an *optimize* object.

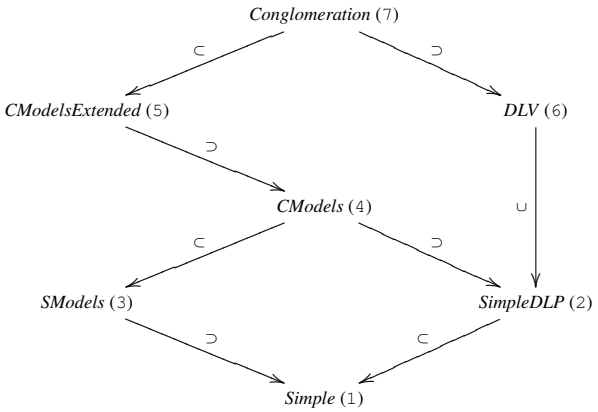


Figure 2: Normal Form Hierarchy

Normal Forms

This section describes seven normal forms corresponding to different language fragments handled by existing ASP solvers. The normal forms stand in a hierarchy, as shown in Figure 2. Each of the normal forms is identified via a corresponding number, given in parentheses in Figure 2, to be provided within the *header* of a problem description in *ASPils*. The full specification of the normal forms presented below can be found in (Gebser *et al.* 2008a). In particular, admissible object types and reference relationships are defined for each normal form in turn. In what follows, we focus on their main features and provide examples.

Normal Form *Simple*

This *normal form* corresponds to the input language used in the *SCore* category of the ASP system competition (Gebser *et al.* 2007b). It allows for representing ground normal logic programs without any extended constructs (like aggregates, etc.). For example, consider the following input program:

```
a :- not b.
b :- not a.
:- b.
c :- d, not b.
d.
#hide a.
```

This program can be represented in *ASPils* as follows:[6]

[6]We indicate the meanings of *object*s in comments preceding their definitions.

```
3 1 "header, language version =: 1,
             normal form =: 1" 0
1 3 1 1 0 0
3 1 "a =: 1 and hidden,
     b =: 2, c =: 3, d =: 4" 0
4 3 1 a 1 0
4 2 2 b 0
4 2 3 c 0
4 2 4 d 0
3 1 "(a :- not b.) =: 5" 0
5 3 5 1 -2 0
3 1 "(b :- not a.) =: 6" 0
5 3 6 2 -1 0
3 1 "(:- b.) =: 7" 0
7 2 7 2 0
3 1 "(d, not b) =: 8" 0
8 3 8 4 -2 0
3 1 "(c :- d, not b.) =: 9" 0
5 3 9 3 8 0
3 1 "(d.) =: 10" 0
6 2 10 4 0
3 1 "end of file" 0
0 0 0 .
```

Note that the above representation is not unique, for instance, we could have assigned different *object ID*s or changed the order of entries.

Normal Form *SimpleDLP*

This *normal form*, corresponding to the input language used in the *SCore*$^{\vee}$ category of the ASP system competition (Gebser *et al.* 2007b), extends normal form "Simple" by allowing *disjunction*s over *atom*s to occur in heads of *rule*s and *fact*s. E.g., consider the following disjunctive program:

```
a | b.
b | c | d :- a, not d.
```

This program can be represented in *ASPils* as follows:

```
3 1 "header, language version =: 1,
             normal form =: 2" 0
1 3 1 2 0 0
3 1 "a =: 1, b =: 2, c =: 3, d =: 4" 0
4 2 1 a 0
4 2 2 b 0
4 2 3 c 0
4 2 4 d 0
3 1 "(a | b) =: 5" 0
9 3 5 1 2 0
3 1 "(a | b.) =: 6" 0
6 2 6 5 0
3 1 "(b | c | d) =: 7" 0
9 4 7 2 3 4 0
3 1 "(a, not d) =: 8" 0
8 3 8 1 -4 0
3 1 "(b | c | d :- a, not d.) =: 9" 0
5 3 9 7 8 0
3 1 "end of file" 0
0 0 0 .
```

As in the previous subsection, this representation in *ASPils* is not unique.

Normal Form *SModels*

This *normal form* is inspired by the input language of solver *smodels* (Simons, Niemelä, & Soininen 2002), that is, it extends "Simple" by *cardinality* and *weight constraint*s as well as *optimize* objects. Note that the *weight*s used in *weight constraint*s and *weighted literal*s have to be non-negative. Furthermore, the upper bounds of *cardinality* and *weight constraint*s must be trivial, that is, they cannot be smaller than the number of literals or the sum of weights, respectively, in a constraint. If a *cardinality constraint* occurs as the head of a *rule* or *fact*, its lower bound must also be trivial, viz., it must be `0`, while *weight constraint*s are not permitted as heads. Finally, note that *weighted literal*s as well as *count* and *sum* aggregates may only be used in combination with an *optimize* object, but not as a part of a *rule*, a *fact*, or an *integrity constraint*. Let us consider the following program:

```
{a, b}.
c :- a, not b.
:- 3[a=2, b=1, not c=2].
minimize[not a=1, not b=2, c=2].
```

The following is a possible representation in *ASPils*:

```
3 1 "header, language version =: 1,
             normal form =: 3" 0
1 3 1 3 0 0
3 1 "a =: 1, b =: 2, c=: 3" 0
4 2 1 a 0
4 2 2 b 0
4 2 3 c 0
3 1 "{a, b} =: 4" 0
b 5 4 0 2 1 2 0
3 1 "({a, b}.) =: 5" 0
6 2 5 4 0
3 1 "(a, not b) =: 6" 0
8 3 6 1 -2 0
3 1 "(c :- a, not b.) =: 7" 0
5 3 7 3 6 0
3 1 "3[a=2, b=1, not c=2] =: 8" 0
c 9 8 3 5 1 2 -3 2 1 2 0
3 1 "(:- 3[a=2, b=1, not c=2].) =: 9" 0
7 2 9 8 0
3 1 "(not a=1) =: 10, (not b=2) =: 11,
          (c=2) =: 12" 0
d 3 10 -1 1 0
d 3 11 -2 2 0
d 3 12  3 2 0
3 1 "sum[not a=1, not b=2, c=2] =: 13" 0
f 4 13 10 11 12 0
3 1 "(minimize[not a=1, not b=2, c=2].)
     =: 14" 0
1d 3 14 1e 13 0
3 1 "end of file" 0
0 0 0 .
```

Normal Form *CModels*

This *normal form* is closely related to the input language of solver *cmodels* (Giunchiglia, Lierler, & Maratea 2006; Lierler 2005), basically, augmenting "SModels" normal form with *disjunction*s in heads of *rule*s and *fact*s.[7]

Normal Form *CModelsExtended*

This *normal form* is derived from "CModels" by dropping some restrictions. Non-trivial upper bounds are permitted for *cardinality* and *weight constraint*s. Furthermore, both of them can occur with non-trivial bounds as heads of *rule*s and

[7] While *cmodels* does not process minimize statements, they can be expressed in "CModels" via *optimize* objects.

*fact*s. Finally, negative *weight*s can be used within *weight constraint*s and *weighted literal*s being subject to *optimize* objects. The following program uses these extra features:

```
0[a=1, b=-1]0 :- 0[c=-1, d=1]0.
0[c=1, d=-1]0 :- 0[a=-1, b=1]0.
minimize[not a=-1, not b=2, c=-2, d=1].
```

This program can be represented in *ASPils* as follows:

```
3 1 "header, language version =: 1,
            normal form =: 5" 0
1 3 1 5 0 0
3 1 "a =: 1, b =: 2, c =: 3, d =: 4" 0
4 2 1 a 0
4 2 2 b 0
4 2 3 c 0
4 2 4 d 0
3 1 "0[a=1, b=-1]0 =: 5" 0
c 7 5 0 0 1 2 1 -1 0
3 1 "0[c=-1, d=1]0 =: 6" 0
c 7 6 0 0 3 4 -1 1 0
3 1 "(0[a=1, b=-1]0 :- 0[c=-1, d=1]0.)
     =: 7" 0
5 3 7 5 6 0
3 1 "0[c=1, d=-1]0 =: 8" 0
c 7 8 0 0 3 4 1 -1 0
3 1 "0[a=-1, b=1]0 =: 9" 0
c 7 9 0 0 1 2 -1 1 0
3 1 "(0[c=1, d=-1]0 :- 0[a=-1, b=1]0.)
     =: 10" 0
5 3 10 8 9 0
3 1 "(not a=-1) =: 11, (not b=2) =: 12,
         (c=-2) =: 13,      (d=1) =: 14" 0
d 3 11 -1 -1 0
d 3 12 -2  2 0
d 3 13  3 -2 0
d 3 14  4  1 0
3 1 "sum[not a=-1, not b=2, c=-2, d=1]
     =: 15" 0
f 5 15 11 12 13 14 0
3 1 "(minimize[not a=-1, not b=2, c=-2,
                 d=1].) =: 16" 0
1d 3 16 1e 15 0
3 1 "end of file" 0
0 0 0 .
```

Normal Form *DLV*

This *normal form* is inspired by the input language of solver *dlv* (Dell'Armi *et al.* 2003; Leone *et al.* 2006). Hence, it allows for *disjunction*s over *atom*s in heads of *rule*s and *fact*s. Furthermore, aggregates may be used in bodies, under the proviso that all referenced *weighted literal*s have non-negative *weight*s.[8] As *dlv* does not deal with cardinality and weight constraints, we exclude *cardinality* and *weight constraint*s from "DLV" normal form. (However, *cardinality* and *weight constraint*s can equivalently be expressed in terms of *count* and *sum* aggregates, respectively.) Finally, weak constraints (Leone *et al.* 2006) can be represented using *optimize* objects. For illustration, consider program:

```
a | b | c.
d :- sum[a=1, b=1, c=2] >= 2.
:~ d, not b. [1:2]
:~ a. [2:1]
:~ not c. [1:1]
```

The following is a representation of this program in *ASPils*:

```
3 1 "header, language version =: 1,
            normal form =: 6" 0
1 3 1 6 0 0
3 1 "a =: 1, b =: 2, c =: 3, d =: 4" 0
4 2 1 a 0
4 2 2 b 0
4 2 3 c 0
4 2 4 d 0
3 1 "(a | b | c) =: 5" 0
9 4 5 1 2 3 0
3 1 "(a | b | c.) =: 6" 0
6 2 6 5 0
3 1 "(a=1) =: 7, (b=1) =: 8, (c=2) =: 9" 0
d 3 7 1 1 0
d 3 8 2 1 0
d 3 9 3 2 0
3 1 "sum[a=1, b=1, c=2] =: 10" 0
f 4 10 7 8 9 0
3 1 "(sum[a=1, b=1, c=2] >= 2) =: 11" 0
1b 3 11 10 2 0
3 1 "(d :- sum[a=1, b=1, c=2] >= 2.)
     =: 12" 0
5 3 12 4 11 0
3 1 "(d, not b) =: 13" 0
8 3 13 4 -2 0
3 1 "((d, not b)=1) =: 14, (a=2) =: 15,
          (not c=1) =: 16" 0
d 3 14 13 1 0
d 3 15  1 2 0
d 3 16 -3 1 0
3 1 "sum[a=2, not c=1] =: 17" 0
f 3 17 15 16 0
3 1 "(minimize[a=2, not c=1].
      minimize[(d, not b)=1].) =: 18" 0
1d 4 18 1e 14 17 0
3 1 "end of file" 0
0 0 0 .
```

Normal Form *Conglomeration*

This *normal form* is the most general one provided here. It results from "CModelsExtended" and "DLV" by dropping some restrictions of the latter, that is, *unary operator*s and *aggregate*s may occur in the heads of *rule*s and *fact*s, and negative *weight*s are allowed within *weighted literal*s. Furthermore, we include *default negation* objects on negative *literal*s over *atom*s in order to account for double negation. Though double negation is a syntactical feature that increases neither computational complexity nor technical difficulties of ASP solving, somewhat astonishingly, it is currently not supported by any ASP solver nor by accompanying grounders. This is why *default negation* objects were not permitted in previous normal forms. The following program, comprising a single rule, uses the additional features:

```
sum[a=1, b=1, c=1, d=-2] == 0 :-
  2[a=-1, not b=2, not c=3]3, not not d.
```

This program can be represented in *ASPils* as follows:

```
3 1 "header, language version =: 1,
            normal form =: 7" 0
1 3 1 7 0 0
3 1 "a =: 1, b =: 2, c =: 3, d =: 4" 0
```

[8] The *dlv* solver requires logic programs to be "aggregate-stratified" (Dell'Armi *et al.* 2003), which is not reflected herein.

```
4 2 1 a 0
4 2 2 b 0
4 2 3 c 0
4 2 4 d 0
3 1 "(a=1) =: 5, (b=1) =: 6, (c=1) =: 7,
    (d=-2) =: 8" 0
d 3 5 1  1 0
d 3 6 2  1 0
d 3 7 3  1 0
d 3 8 4 -2 0
3 1 "sum[a=1, b=1, c=1, d=-2] =: 9" 0
f 5 9 5 6 7 8 0
3 1 "(sum[a=1, b=1, c=1, d=-2] == 0)
     =: 10" 0
18 3 10 9 0 0
3 1 "2[a=-1, not b=2, not c=3]3 =: 11" 0
c 9 11 2 3 1 -2 -3 -1 2 3 0
3 1 "(not not d) =: 12" 0
a 2 12 -4 0
3 1 "(2[a=-1, not b=2, not c=3]3,
      not not d) =: 13" 0
8 3 13 11 12 0
3 1 "(sum[a=1, b=1, c=1, d=-2] == 0 :-
        2[a=-1, not b=2, not c=3]3,
        not not d.) =: 14" 0
5 3 14 10 13 0
3 1 "end of file" 0
0 0 0 .
```

Discussion and Outlook

We have described language version `1` of *ASPils* ("ASP intermediate language standard"); full details can be found in (Gebser *et al.* 2008a). The primary motivation for this work is making a step towards a standard input language for ASP solvers to be generated by grounders, for which we propose *ASPils*. The major design goals of *ASPils* are generality, by supporting language constructs processed by existing ASP grounders and solvers, and extensibility, by using an object-based approach and including version information. For solvers, parsing a problem description in *ASPils* should still be reasonably simple, thus, *ASPils* defines a numerical format not intended to be manually written by users. However, *ASPils* also provides means to specify symbolic information, enabling the reconstruction of a human-readable format. Beyond that, via comments and meta-information, arbitrary contents can be included in a problem description without disturbing solvers. The current proposal of *ASPils* is a good response to the recommendations presented in (Janhunen 2007) as regards extensibility and support for comments as well as symbolic information.

As a proof of concept, we are currently working on a new version of grounder *gringo* (Gebser, Schaub, & Thiele 2007) able to output *ASPils* format and also on an *ASPils* front-end for solver *clasp* (Gebser *et al.* 2007a). In the course of this, we take advantage of the generic design of *ASPils* allowing us to preserve the structure of ground logic programs. For instance, *gringo* can output cardinality and weight constraints specifying both a lower and a non-trivial upper bound, and such constraints can occur both in the bodies and in the heads of rules. In contrast, in *lparse*'s output format (Syrjänen), upper bounds (and in rule heads, also lower bounds) have to be compiled away, introducing additional atoms and rules. Such structure-degrading transformations are performed by *lparse* in order to match the internal data structures of *smodels* (Simons, Niemelä, & Soininen 2002), and in the past, tools (Liu & Truszczyński 2005) were particularly developed to undo such transformations. As regards grounding, we think that the two tasks of a grounder are, first, substituting constants for variables in an input program and, second, presenting the grounding result to a solver in some basic format that is easy to parse. Beyond these two tasks, a grounder should keep the input program intact in order to be solver-independent and to abolish the need of applying structure-restoring tools. In particular, introducing additional atoms during the grounding phase ought to be avoided, as it is very likely to spoil the desired equivalence between the input program and the grounding result.

In long-term, we hope that our proposal of an intermediate language standard leads to the establishment of a common input format for ASP solvers, comparable to the role of DIMACS format (DIMACS 1993) in SAT. On the one hand, it would make ASP more user-friendly if solvers could be interchanged without redoing problem encodings, given that the two main input languages, the one of *dlv* (Leone *et al.* 2006) and the one of *lparse* (Syrjänen), are incompatible with each other. A common intermediate language would enhance the interoperability of other auxiliary tools as well. Similarly, the assessment of ASP solvers would be greatly facilitated, for instance, in future ASP solver competitions. On the other hand, the non-availability of a standardized intermediate language (as *dlv* does not supply an intermediate format and *lparse*'s output language is not standardized) makes ASP solvers and related tools satellites of particular grounders, addicted to their capabilities and supported language fragments. We think that the establishment of an extensible intermediate language standard, not dictated by the capabilities of grounders, might motivate future works on knowledge representation for applications, inventing new language constructs when they are useful and then integrating them into the standard. At the moment, incorporating new language features would mean hacking one of the few available grounding tools, making the broad acceptance and usage of the feature rather unlikely. However, the establishment of an intermediate language standard must be a community effort, requiring a representative standardization committee and developers motivated to implement the standard in their tools. In view of these requirements, our proposal of *ASPils* can serve as a starting point for future discussions within the community.

Let us note that the establishment of *ASPils* or a comparable intermediate language standard can only be a small step in making ASP tools more general, more interoperable, and thus more user-friendly. In fact, tools are needed to perform various useful tasks on problem descriptions in the new language, similar to what the Helsinki collection (Janhunen) of tools offers for *lparse*'s output format. Let us give some examples. For the use in ASP system competitions, solver inputs must contain neither meta-information nor comments, thus, a (trivial to develop) tool to delete such information would be needed. For benchmarking, a tool

like *shuffle* is desirable, in particular, considering that shuffling *ASPils* sentences requires more care than needed with *lparse*'s output format as the order (Gebser *et al.* 2008a) among object definitions and references has to be maintained. If a grounder generates *ASPils* output on-the-fly, it is hard to predict the most restrictive normal form sufficient for the input program, hence, a postprocessor calculating this simplest normal form might be useful. As the last example given here, a tool like *lplist* should be made available for reconstructing a symbolic representation from the intermediate format. Finally, we note that *ASPils* or any other intermediate language cannot establish compatibility among the input languages of grounders or integrated ASP systems (such as *dlv*). Furthermore, modular (Oikarinen & Janhunen 2006), incremental (Gebser *et al.* 2008b), or even systems dealing with non-ground input programs are currently out of the scope of our proposal. However, we think that the relatively simple concept of an intermediate language provides a good basis for standardization efforts, and more sophisticated matters could be addressed in the future.

Acknowledgments. We are grateful to Marcello Balduccini, Martin Brain, Maarten Mariën, and Axel Polleres for fruitful discussions and valuable ideas that contributed to this work. Also, we would like to thank the anonymous reviewers whose comments helped us to improve the presentation. The second author was supported by the project #122399 "*Methods for Constructing and Solving Large Constraint Models*" funded by the Academy of Finland.

References

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

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

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

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 *Proc. of IJCAI'03*, 847–852.

DIMACS. 1993. Satisfiability suggested format. [9]

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *TPLP* 3(4-5):499–518.

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

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007a. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR'07*, 260–265.

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczyński, M. 2007b. The first answer set programming system competition. In *Proc. of LPNMR'07*, 3–17.

Gebser, M.; Janhunen, T.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008a. A versatile intermediate language for answer set programming: Syntax proposal. [10]

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008b. Engineering an incremental ASP solver. In *Proc. of ICLP'08*.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A new grounder for answer set programming. In *Proc. of LPNMR'07*, 266–271.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation — the A-Prolog perspective. *AIJ* 138(1-2):3–38.

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

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *JAR* 36(4):345–377.

Janhunen, T. The ASPTOOLS collection. [11]

Janhunen, T. 2007. Intermediate languages of ASP systems and tools. In *Proc. of SEA'07*, 12–25.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proc. of IJCAI'05*, 503–508.

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.

Lierler, Y. 2005. cmodels - SAT-based disjunctive answer set solver. In *Proc. of LPNMR'05*, 447–451.

Lifschitz, V.; Tang, L.; and Turner, H. 1999. Nested expressions in logic programs. *AMAI* 25(3-4):369–389.

Liu, L., and Truszczyński, M. 2005. Pbmodels - software to compute stable models by pseudoboolean solvers. In *Proc. of LPNMR'05*, 410–415.

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

Marques-Silva, J., and Sakallah, K. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE TC* 48(5):506–521.

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

Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proc. of DAC'01*, 530–535.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *AMAI* 25(3-4):241–273.

Oikarinen, E., and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *Proc. of ECAI'06*, 412–416.

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

Syrjänen, T. Lparse 1.0 user's manual. [12]

[9] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/`

[10] `http://www.cs.uni-potsdam.de/wv/pdfformat/gejaosscth08a.pdf`

[11] `http://www.tcs.hut.fi/Software/asptools/`

[12] `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`

Engineering an Incremental ASP Solver: Preliminary Report

M. Gebser and **R. Kaminski** and **B. Kaufmann** and **M. Ostrowski** and **T. Schaub** and **S. Thiele**
Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam, Germany

Abstract

We provide the first theoretical and practical account of incremental answer set solving. In fact, many real-world applications, like planning or model-checking, comprise a parameter reflecting the size of a solution. In a propositional formalism like Answer Set Programming (ASP), such problems can only be dealt with in a bounded way, considering one problem instance after another by gradually increasing the bound on the solution size. We thus propose an incremental approach to both grounding and solving in ASP. Our goal is to avoid redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the (gradually extended) entire problem. We start by furnishing a formal framework capturing our incremental approach in terms of module theory. In turn, we take advantage of this framework for guiding the successive treatment of program slices during grounding and solving. Finally, we describe the first integrated incremental ASP system, *iclingo*, and provide an experimental evaluation.

Introduction

Answer Set Programming (ASP; (Baral 2003)) faces a growing range of applications. The popularity of ASP is due to the availability of efficient off-the-shelf ASP solvers and its rich modeling language, jointly allowing for an easy yet efficient handling of knowledge-intensive applications. Among them, many real-world applications, as in bioinformatics, model checking or finding, planning, etc., comprise parameters reflecting solution sizes. However, in the propositional setting of ASP, such problems can only be dealt with in a bounded way by considering in turn one problem instance after another by gradually increasing the bound on the solution size. Such an approach can nonetheless be highly efficient as demonstrated by Satisfiability (SAT) solvers in the area of planning (Kautz and Selman 1992) and model checking (Clarke et al. 2001). However, while SAT has its focus on solving, ASP is furthermore concerned with grounding in view of its modeling language. We thus propose an incremental approach to both grounding and solving in ASP.

Our goal is to avoid redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem. To this end, we express a (*parametrized*) *domain description* as a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers. In view of this, we sometimes denote P and Q by $P[k]$ and $Q[k]$. The base program B is meant to describe static knowledge, independent of parameter k. The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each k and aims at expressing query conditions. Our goal is then to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i] \qquad (1)$$

has an (or no) answer set for some (minimum) integer $i \geq 1$. In what follows, we write $R[i]$ rather than $R[k/i]$ whenever clear from the context.

To illustrate this, consider an action description in language $\mathcal{C}+$ (Giunchiglia et al. 2004) involving an action a and a fluent p. We complement this with a query formulated in action query language $\mathcal{Q}_n$ (Gelfond and Lifschitz 1998) for expressing queries about trajectories of length n.

a **causes** p	$\neg p$ **holds at** 0
exogenous a	p **holds at** n
inertial p	$\neg a$ **occurs at** n

We translate these statements into the following domain description (following (Lifschitz and Turner 1999)):[1]

$$B = \left\{ \begin{array}{rcl} p(0) & \leftarrow & not\ \neg p(0) \\ \neg p(0) & \leftarrow & not\ p(0) \\ & \leftarrow & p(0), \neg p(0) \end{array} \right\}$$

$$P[k] = \left\{ \begin{array}{rcl} a(k) & \leftarrow & not\ \neg a(k) \\ \neg a(k) & \leftarrow & not\ a(k) \\ p(k) & \leftarrow & a(k) \\ p(k) & \leftarrow & p(k{-}1), not\ \neg p(k) \\ \neg p(k) & \leftarrow & \neg p(k{-}1), not\ p(k) \\ & \leftarrow & p(k), \neg p(k) \\ & \leftarrow & a(k), \neg a(k) \end{array} \right\} \qquad (2)$$

$$Q[k] = \left\{ \begin{array}{rcl} \leftarrow & not\ \neg p(0) \\ \leftarrow & not\ p(k) \\ \leftarrow & not\ \neg a(k) \end{array} \right\}$$

This domain description induces no answer sets for $R[1]$, but we obtain a single one for $R[2]$, that is, $AS(R[2]) = \{\{\neg p(0), a(1), p(1), \neg a(2), p(2)\}\}$.

Such an answer is usually found by appeal to iterative deepening search. That is, one first checks whether $R[1]$ has an answer set, if not, the same is done for $R[2]$, and so on. For a given i, this approach involves re-processing B for i times and $(i{-}j{+}1)$ times each $P[j]$, where $1 \leq j \leq i$, while each $Q[j]$ is dealt with only once.

[1] For admitting consistent answer sets only, we add integrity constraints to B and $P[k]$ accounting for classical negation.

Unlike this, we propose to compute answers sets of (1) in an incremental fashion, starting from $R[1]$ but then gradually dealing with the program slices $P[i]$ and $Q[i]$ rather than the entire program $R[i]$ in (1). However, B and the previously processed slices $P[j]$ and $Q[j]$, $1 \leq j < i$, must be taken into account when dealing with $P[i]$ and $Q[i]$. While the rules in $P[j]$ must be accumulated, the ones in $Q[j]$ must be discarded. For accomplishing this, an ASP system has to operate in a "stateful way." That is, it has to maintain its previous state for processing the current program slices. In this way, all components, B, $P[j]$, and $Q[i]$, of (1) are dealt with only once, and duplicated work is avoided when increasing i. Given that an ASP system is composed of a grounder and a solver, our incremental approach has the following specific advantages over the standard approach. As regards grounding, it reduces efforts by avoiding reproducing previously grounded rules. Regarding solving, it reduces efforts, in particular, if a learning ASP solver is used, given that previously gathered information on heuristics, conflicts, or loops (cf. (Gebser et al. 2007b)), respectively, remains available and can thus be continuously exploited.

Background

Our language is built from a set $\mathcal{F}$ of *function* symbols (including the natural numbers), a set $\mathcal{V}$ of *variable* symbols, and a set $\mathcal{P}$ of *predicate* symbols. The set $\mathcal{T}$ of *terms* is the smallest set containing $\mathcal{V}$ and all expressions of the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$ and $t_i \in \mathcal{T}$ for $1 \leq i \leq n$. The set $\mathcal{A}$ of *atoms* contains expressions of the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ and $t_i \in \mathcal{T}$ for $1 \leq i \leq n$. A *literal* is an atom a or its (default) negation $not\ a$. Given a set L of literals, let $L^+ = \{a \in \mathcal{A} \mid a \in L\}$ and $L^- = \{a \in \mathcal{A} \mid not\ a \in L\}$.

A *logic program* over $\mathcal{A}$ is a set of *rules* of the form[2]

$$a \leftarrow b_1, \ldots, b_m, not\ c_{m+1}, \ldots, not\ c_n\ , \qquad (3)$$

where $a, b_i, c_j \in \mathcal{A}$ for $0 \leq i \leq m \leq j \leq n$. For a rule r as in (3), let $head(r) = a$ be the *head* of r and the set $body(r) = \{b_1, \ldots, b_m, not\ c_{m+1}, \ldots, not\ c_n\}$ be the *body* of r. Furthermore, let $atom(r) = \{head(r)\} \cup body(r)^+ \cup body(r)^-$. For a logic program P, define $head(P) = \{head(r) \mid r \in P\}$ and $atom(P) = \bigcup_{r \in P} atom(r)$.

Let $var(e)$ denote the set of all variables occurring in $e \in \mathcal{T} \cup \mathcal{A}$; analogously, $var(r)$ gives all variables in rule r. Expression $e \in \mathcal{T} \cup \mathcal{A}$ is *ground*, if $var(e) = \emptyset$. The *ground instantiation* of a program P is defined as $grd(P) = \{r\theta \mid r \in P, \theta : var(r) \to \mathcal{U}\}$, where $\mathcal{U} = \{t \in \mathcal{T} \mid var(t) = \emptyset\}$; analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$.

A set $X \subseteq grd(\mathcal{A})$ is an *answer set* of a logic program P over $\mathcal{A}$, if X is the $\subseteq$-smallest model of $\{head(r) \leftarrow body(r)^+ \mid r \in grd(P), body(r)^- \cap X = \emptyset\}$. The set of answer sets of a program P is denoted $AS(P)$. Two programs, P and P', are *equivalent*, $P \equiv P'$, if $AS(P) = AS(P')$.

In view of our goal, the variable set $\mathcal{V}$ contains a distinguished parameter symbol k. A rule r is *parametrized*, if $k \in var(r)$, and *non-parametrized* otherwise. Accordingly, a (*parametrized*) *domain description* (B, P, Q) consists of a set B of non-parametrized rules and two sets P, Q of parametrized rules.

[2] The semantics of integrity constraints and choice rules is given through program transformations. For instance, $\{a\} \leftarrow$ is a shorthand for $a \leftarrow not\ a', a' \leftarrow not\ a$ and similarly $\leftarrow a$ for $a' \leftarrow a, not\ a'$, for a new atom a'.

Incremental Modularity

For the sake of compositionality, we build upon the concept of a *module* developed in (Oikarinen and Janhunen 2006): Given a ground logic program P over $grd(\mathcal{A})$ and sets $I, O \subseteq grd(\mathcal{A})$ such that $I \cap O = \emptyset$, a *module* $\mathbb{P}$ is a triple (P, I, O) such that $atom(P) \subseteq I \cup O$ and $head(P) \subseteq O$. The elements of I and O are called *input* and *output* atoms; also denoted $I(\mathbb{P})$ and $O(\mathbb{P})$, respectively. Similarly, we refer to P by $P(\mathbb{P})$. We say that $\mathbb{P}$ is *input-free*, if $I(\mathbb{P}) = \emptyset$.

For defining an incremental account of modularity, we begin with associating a (non-ground) program P and a set I of (ground) input atoms with a module $\mathbb{P}(I)$ imposing certain restrictions on the ground program induced by P. We need the following auxiliary definition: For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$, define $P|_X$ as

$$\{head(r) \leftarrow body(r)^+ \cup L \mid r \in P, body(r)^+ \subseteq X, \\ L = \{not\ c \mid c \in body(r)^- \cap X\}\}\ .$$

Definition 1 *Let P be a logic program over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$. We define $\mathbb{P}(I)$ as the module*

$$(\ grd(P)|_Y\ ,\ I\ ,\ head(grd(P)|_X)\ )\ ,$$

where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$.

As a simple example, consider $P[k] = \{p(k) \leftarrow p(Y); p(k) \leftarrow p(2)\}$. Note that $grd(P[1])$ is infinite. However, when restricting our attention to $X = \{p(0), p(1)\}$, we get

$$grd(P[1])|_X = \{p(1) \leftarrow p(0) \qquad p(1) \leftarrow p(1)\}$$

and $head(grd(P[1])|_X) = \{p(1)\}$. Taking $I = \{p(0)\}$, note that $I \cup head(grd(P[1])) = I \cup head(grd(P[1])|_X) = \{p(0)\} \cup \{p(1)\} = X$. Thus, $\mathbb{P}[1](\{p(0)\}) = (\ grd(P[1])|_X, \{p(0)\}, \{p(1)\}\ )$ and $P(\mathbb{P}[1](I)) = grd(P[1])|_X$ are finite.

Syntactically, Definition 1 yields the following property.

Proposition 1 *Let P be a logic program over $\mathcal{A}$, $I \subseteq grd(\mathcal{A})$, and $\mathbb{P}(I) = (P', I, O)$. Then, we have $O \subseteq grd(\mathcal{A})$ and $atom(P') \subseteq I \cup O$.*

Given two modules $\mathbb{P}$ and $\mathbb{Q}$, we define the *join* of $\mathbb{P}$ and $\mathbb{Q}$, denoted by $\mathbb{P} \sqcup \mathbb{Q}$, as the module

$$(\ P(\mathbb{P}) \cup P(\mathbb{Q})\ ,\ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P}))\ ,\ O(\mathbb{P}) \cup O(\mathbb{Q})\ )\ ,$$

provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$. This definition of the join is simpler than the original one in (Oikarinen and Janhunen 2006), but also more restrictive. For instance, our definition does not permit (negative) recursion between two modules to be joined, which is similar to splitting (Lifschitz and Turner 1994). Also note that the join of $\mathbb{P}$ and $\mathbb{Q}$, as defined above, is not commutative: Even if $\mathbb{P} \sqcup \mathbb{Q}$ is defined, $\mathbb{Q} \sqcup \mathbb{P}$ might be undefined. However, (lacking) commutativity is not an issue in our incremental setting, where portions of a domain description are always processed in order.

We make use of the join to attribute domain descriptions.

Definition 2 *We define a domain description $(B, P[k], Q[k])$ as modular, if the modules*

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$.

This definition formalizes compositionality of modules arising from domain descriptions. In particular, the requirement of the join being defined demands that gradually obtained ground programs must define distinct atoms. Furthermore, the directedness of the join, in a sense, permits an information flow between ground programs in increasing order of values substituted for k, but not the other way round.

As an example, consider $(B, P[k], Q[k])$ over $\mathcal{A}$, where:

$$\begin{array}{rl} B = & \{\ dbl(0,0) \leftarrow\ \ \} \\ P[k] = & \left\{ \begin{array}{c} n(k) \leftarrow \\ dbl(k, Y{+}Y) \leftarrow n(Y), \mathit{not}\ n(Y{+}1) \end{array} \right\} \\ Q[k] = & \{\ \leftarrow dbl(Y, k{-}1)\ \}\ . \end{array} \quad (4)$$

Letting $\mathcal{F}$ (underlying $\mathcal{A}$) be the set of natural numbers, the above domain description induces the following modules:[3]

$$\mathbb{P}_0 = (\ B = \{dbl(0,0) \leftarrow\},\ \emptyset,\ \{dbl(0,0)\}\ ),$$

$$\mathbb{P}_1 = (\ P(\mathbb{P}_1),\ \emptyset,\ O(\mathbb{P}_0) \cup \{n(1), dbl(1,2)\}\ )$$
$$\text{where } P(\mathbb{P}_1) = B \cup \left\{ \begin{array}{c} n(1) \leftarrow \\ dbl(1,2) \leftarrow n(1) \end{array} \right\},$$

$$\mathbb{Q}_1 = (\ P(\mathbb{Q}_1) = P(\mathbb{P}_1) \cup \{\leftarrow dbl(0,0)\},\ \emptyset,\ O(\mathbb{P}_1)\ ),$$

$$\mathbb{P}_2 = (\ P(\mathbb{P}_2),\ \emptyset,\ O(\mathbb{P}_1) \cup \{n(2), dbl(2,2), dbl(2,4)\}\ )$$
$$\text{where } P(\mathbb{P}_2) = P(\mathbb{P}_1) \cup \left\{ \begin{array}{c} n(2) \leftarrow \\ dbl(2,2) \leftarrow n(1), \mathit{not}\ n(2) \\ dbl(2,4) \leftarrow n(2) \end{array} \right\},$$

$$\mathbb{Q}_2 = (\ P(\mathbb{Q}_2) = P(\mathbb{P}_2)\ ,\ \emptyset,\ O(\mathbb{P}_2)\ ),$$

$$\mathbb{P}_3 = (\ P(\mathbb{P}_3),\ \emptyset,\ O(\mathbb{P}_2) \cup \{n(3), dbl(3,2), dbl(3,4), dbl(3,6)\}\ )$$
$$\text{where } P(\mathbb{P}_3) = P(\mathbb{P}_2) \cup \left\{ \begin{array}{c} n(3) \leftarrow \\ dbl(3,2) \leftarrow n(1), \mathit{not}\ n(2) \\ dbl(3,4) \leftarrow n(2), \mathit{not}\ n(3) \\ dbl(3,6) \leftarrow n(3) \end{array} \right\},$$

$$\mathbb{Q}_3 = (\ P(\mathbb{Q}_3),\ \emptyset,\ O(\mathbb{P}_3)\ )$$
$$\text{where } P(\mathbb{Q}_3) = P(\mathbb{P}_3) \cup \left\{ \begin{array}{l} \leftarrow dbl(1,2) \\ \leftarrow dbl(2,2) \\ \leftarrow dbl(3,2) \end{array} \right\}, \text{etc.}$$

Note that all of the above modules are defined (in terms of the join) and input-free. Since this also applies to modules $\mathbb{P}_i$ and $\mathbb{Q}_i$ for every $i > 3$, we have that domain description $(B, P[k], Q[k])$ is modular. Given that all $\mathbb{Q}_i$ are defined for $i \geq 1$, we can read off the results of the expressed queries from the answer sets of each $P(\mathbb{Q}_i)$. If $i \geq 1$ is odd, we get $AS(P(\mathbb{Q}_i)) = \emptyset$. Otherwise, if $i \geq 1$ is even, then $AS(P(\mathbb{Q}_i)) = \{\{dbl(0,0)\} \cup \{n(j), dbl(j, 2{*}j) \mid 1 \leq j \leq i\}\}$.

The composition of modules obtained from some modular domain description has the following properties.

Proposition 2 *Let $(B, P[k], Q[k])$ be a modular domain description, and let $(\mathbb{P}_i)_{i\geq 0}$ and $(\mathbb{Q}_i)_{i\geq 1}$ as in Definition 2. Then, we have the following for $i \geq 1$:*

1. *$\mathbb{P}_i$ and $\mathbb{Q}_i$ are input-free;*
2. *$atom(P(\mathbb{P}_i)) \subseteq O(\mathbb{P}_i)$ and $atom(P(\mathbb{Q}_i)) \subseteq O(\mathbb{Q}_i)$;*
3. *$P(\mathbb{P}_i) = P(\mathbb{B}(\emptyset)) \cup \bigcup_{1\leq j\leq i} P(\mathbb{P}[j](O(\mathbb{P}_{j-1})))$ and $P(\mathbb{Q}_i) = P(\mathbb{P}_i) \cup P(\mathbb{Q}[i](O(\mathbb{P}_i)))$;*
4. *$head(P(\mathbb{P}[i](O(\mathbb{P}_{i-1})))) \cap atom(P(\mathbb{P}_{i-1})) = \emptyset$ and $head(P(\mathbb{Q}[i](O(\mathbb{P}_i)))) \cap atom(P(\mathbb{P}_i)) = \emptyset$.*

The third item essentially states that the combined logic programs obtained for $i \geq 1$ equal the union of subprograms added for each $1 \leq j \leq i$. Importantly, the fourth item expresses that the head atoms of a newly added subprogram are different from all atoms encountered before. Hence, the sequence $(O(\mathbb{P}_i))_{i\geq 0}$ of output atoms amounts to a splitting sequence (Lifschitz and Turner 1994) for $\bigcup_{i\geq 0} P(\mathbb{P}_i)$.

Note that we take advantage of module theory only for establishing a well-defined formal setting for incremental ASP solving. Our computational approach deals directly with logic programs in order to exploit existing ASP technology. In view of this, the next result shows when the module-guarded formation of ground logic programs coincides with isolated grounding. For this, we define a domain description $(B, P[k], Q[k])$ as *bound*, if for all $i \geq 1$:

1. $atom(grd(B)) \subseteq head(grd(B))$ and
2. $atom(grd(P[i])) \subseteq head(grd(B \cup \bigcup_{1\leq j\leq i} P[j]))$.

The following result shows that, for bound modular domain descriptions, the same result is obtained when grounding is done either module-wise or in a single pass.

Theorem 3 *Let $(B, P[k], Q[k])$ be a bound modular domain description, and let $(\mathbb{P}_i)_{i\geq 0}$ and $(\mathbb{Q}_i)_{i\geq 1}$ as in Definition 2. Then, we have the following for $i \geq 1$:*

1. *$P(\mathbb{P}_i) \equiv grd(B \cup \bigcup_{1\leq j\leq i} P[j])$;*
2. *$P(\mathbb{Q}_i) \equiv grd(B \cup \bigcup_{1\leq j\leq i} P[j] \cup Q[i])$.*

Note that the domain description given in the introductory section is modular and bound. Likewise, the domain description in (4) is modular, but it is not bound.

Incremental ASP Solving

The computation of answer sets of logic programs is divided into two phases: a *grounding* phase aiming at a compact ground instantiation of the original program, and a *solving* phase computing the answer sets of the obtained ground program. As mentioned in the introductory section, our incremental approach is based on the idea that the grounder as well as the solver are implemented in a stateful way. Thus, both keep their previous states when increasing the parameter k in (1). As regards grounding, at each step i, the goal is to produce only ground rules stemming from program slices $P[i]$ and $Q[i]$, without re-producing any previous ground rules. The ground program slices are then gradually passed to the solver that accumulates all ground rules from $P[j]$, for $1 \leq j \leq i$, while discarding the rules from $Q[j]$ if $j < i$.

Grounding Let us now characterize the consecutive program slices in terms of grounding logic programs. In practice, given a program P, the goal of a grounder is to produce a finite and compact yet equivalent representation of $grd(P)$ by applying answer-set preserving simplifications (cf. (Brass and Dix 1999; Brass et al. 2001; Eiter et al. 2004)). In our setting, $P[i]$ and $Q[i]$ are not

[3] For simplicity, we evaluate arithmetic expressions.

grounded in isolation for $i \geq 1$. Rather the ground programs obtained from previous program slices are augmented with newly derived ground rules. We thus assume a grounder to be stateful, where states are represented by the head atoms of ground rules belonging to the output of previous grounding steps.

Given a logic program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, we define an *(incremental) grounder* as a partial function

$$\texttt{ground} : (P, I) \mapsto (P', O) \,,$$

where P' is a logic program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$. Thereby, P' stands for the ground program obtained from P, where the input atoms I provide domain information used to instantiate (non-ground) atoms in the positive bodies of rules from P. The output atoms in O essentially correspond to $head(P')$, though some atoms in $head(P')$ might actually be excluded from O (cf. Definition 3). The main use of O is to carry state information, as it can serve as input to subsequent grounding steps. Also note that `ground` is not required to be total, given that existing grounders, like *lparse* (Syrjänen) and *gringo* (Gebser, Schaub, and Thiele 2007), impose certain restrictions on (non-ground) programs, such as being ω- or λ-restricted, not necessarily met by P.

The next definition formalizes *adequacy* of a grounder in the context of our incremental approach.

Definition 3 *We define a grounder* `ground` *as adequate, if for every logic program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$ such that* $\texttt{ground}(P, I) = (P', O)$ *is defined, the following holds*:

1. $\big(P \cup \{\{a\} \leftarrow \mid a \in I\}\big) \equiv \big(P' \cup \{\{a\} \leftarrow \mid a \in I\}\big)$,
2. $\bigcup_{X \in AS(P \cup \{\{a\} \leftarrow \mid a \in I\})}(X \setminus I) \subseteq O \subseteq head(grd(P)|_Y)$, *where* $Y = I \cup head(grd(P))$, *and*
3. *for each* $r' \in P'$, *there is some* $r \in grd(P)$ *such that* $head(r) = head(r')$ *and* $body(r)^+ \setminus (I \cup O) \subseteq body(r')^+$.

The first condition expresses that P and P', each augmented with choice rules generating any combination of input atoms in I,[4] should be equivalent to each other. The second condition stipulates that all non-input atoms belonging to some answer set X of $P \cup \{\{a\} \leftarrow \mid a \in I\}$ are contained in O. In addition, O must not exceed the head atoms of $grd(P)|_{(I \cup head(grd(P)))}$ in order to suitably restrict ground rules subsequently produced in our incremental setting, thereby, using O as an input (cf. Definition 4). Finally, the third condition forbids the introduction of rules that cannot be obtained from $grd(P)$ using permissible simplifications only. Beyond the requirements postulated in Definition 3, an adequate grounder `ground` may or may not perform further answer-set preserving simplifications, such as removing the heads of facts from positive bodies or eliminating rules containing them in their negative bodies.

For illustration, reconsider $P[k]$ in (4). Given $I = \{n(1)\}$, an adequate grounder `ground` could, e.g., map $(P[2], I)$ to $\texttt{ground}(P[2], I) = (P, O = \{n(2), dbl(2,2), dbl(2,4)\})$, where

$$P = \left\{ \begin{array}{c} n(2) \leftarrow \\ dbl(2,2) \leftarrow n(1), \mathit{not}\ n(2) \\ dbl(2,4) \leftarrow n(2), \mathit{not}\ n(3) \end{array} \right\}. \qquad (5)$$

[4]The application strategies for "modular equivalence" in (Oikarinen and Janhunen 2006) are based upon a similar technique.

Note that $AS(P \cup \{\{n(1)\} \leftarrow\}) = \{\{n(1), n(2), dbl(2,4)\}, \{n(2), dbl(2,4)\}\} = AS(P[2] \cup \{\{n(1)\} \leftarrow\})$. Due to fact $n(2) \leftarrow$, the second rule could also be dropped from P; similarly, $dbl(2,2)$ can optionally be removed from O. Furthermore, literals $n(2)$ and $\mathit{not}\ n(3)$ could be dropped from the body of the last rule, still satisfying the requirements in Definition 3. Note that it is crucial to restrict the atoms in O to $head(grd(P[2])|_{(\{n(1)\} \cup head(grd(P[2])))}) = head(P)$. For instance, this forbids the inclusion of $n(3)$ in O, permitting further simplifications of P wrt O.

The following definition formalizes the (ground) program slices gradually obtained from a domain description using a (stateful) grounder.

Definition 4 *Let $(B, P[k], Q[k])$ be a domain description, and let* `ground` *be a grounder. We define for $i \geq 1$:*

$$\begin{array}{rl} (P_0, O_0) = & (P_0'|_{O_0}, O_0)\,, \\ & \textit{where } (P_0', O_0) = \texttt{ground}(B, \emptyset)\,, \\ (P_i, O_i) = & (P_i'|_{(\bigcup_{0 \leq j \leq i} O_j)}, O_i)\,, \\ & \textit{where } (P_i', O_i) = \texttt{ground}(P[i], \bigcup_{0 \leq j < i} O_j)\,, \\ (Q_i, O_i') = & \texttt{ground}(Q[i], \bigcup_{0 \leq j \leq i} O_j)\,. \end{array}$$

Note that the successively identified output atoms in O_j, for $0 \leq j \leq i$, are used to simplify ground programs P_i' by eliminating either rules or negative body literals. We thus obtain ground program slices P_i such that $\bigcup_{r \in P_i}(body(r)^+ \cup body(r)^-) \subseteq \bigcup_{0 \leq j \leq i} O_j$. This reduction is important in view of the compositional semantics of domain descriptions in Definition 2. For instance, if not done by `ground` itself, literal $\mathit{not}\ n(3)$ must a posteriori be removed from the body of the third rule in (5), in order to obtain the intended ground program slice. However, ground programs Q_i need not be reduced, since their rules are neither accumulated nor reused.

The next result links the semantics of modular domain descriptions to that of ground programs gradually produced by an adequate grounder `ground`.

Theorem 4 *Let $(B, P[k], Q[k])$ be a modular domain description, and let* `ground` *be an adequate grounder. Furthermore, let $(\mathbb{P}_i)_{i \geq 0}$ and $(\mathbb{Q}_i)_{i \geq 1}$ as in Definition 2, and let $(P_i, O_i)_{i \geq 0}$ and $(Q_i, O_i')_{i \geq 1}$ as in Definition 4. If (P_j, O_j) is defined for all $0 \leq j \leq i$, we have the following for $i \geq 1$:*

1. $P(\mathbb{P}_0) \equiv P_0$;
2. $P(\mathbb{P}_i) \equiv \bigcup_{0 \leq j \leq i} P_j$;
3. $P(\mathbb{Q}_i) \equiv \bigcup_{0 \leq j \leq i} P_j \cup Q_i$, *provided (Q_i, O_i') is defined.*

Recall that `ground` can be partial. In fact, existing grounders impose further restrictions on the (non-ground) programs of a domain description, such as being ω- or λ-restricted, guaranteeing the finiteness of equivalent ground programs. Assuming that such additional requirements are met, we next detail how grounding output, i.e., P_j for $j \geq 0$ and Q_i for $i \geq 1$, can be processed by an answer set solver.

Solving As with grounding, special care must be taken for customizing existing ASP solving technology in an incremental setting. First, we have to guarantee the compositionality of successive program slices. Second, a solver has to

respect the cumulative and volatile roles of P_j and Q_i, respectively. And finally, we have to furnish a clear interface between the grounding and the solving component.

For capturing the desired notion of compositionality, let us have recourse to the Lin-Zhao theorem (Lin and Zhao 2004), characterizing the answer sets of a program P over $grd(\mathcal{A})$ by the (classical) models of its completion and loop formulas. With respect to some $Y \subseteq grd(\mathcal{A})$, we define the *completion* of P, $CF(P,Y)$, as the set of formulas

$$a \leftrightarrow \bigvee_{r\in P, head(r)=a}(\bigwedge_{b\in body(r)^+} b \wedge \bigwedge_{c\in body(r)^-} \neg c)$$

for all $a \in Y$. Furthermore, $Y \subseteq grd(\mathcal{A})$ is a *loop* of P, if $(Y, E = \{(head(r), b) \mid r \in P, head(r) \in Y, b \in Y \cap body(r)^+\})$ is a strongly connected graph such that $E \neq \emptyset$. Then, the set of *loop formulas* for P, $LF(P)$, is given by

$$\bigvee_{a\in Y} a \rightarrow \bigvee_{r\in P_Y}(\bigwedge_{b\in body(r)^+} b \wedge \bigwedge_{c\in body(r)^-} \neg c)$$

for all loops Y of P, where $P_Y = \{r \in P \mid head(r) \in Y, body(r)^+ \cap Y = \emptyset\}$. As shown in (Lin and Zhao 2004), a set $X \subseteq grd(\mathcal{A})$ is an answer set of a logic program P iff $X \models CF(P, grd(\mathcal{A})) \cup LF(P)$.

For programs induced by modular domain descriptions, completion and loop formulas can be sliced as follows.

Theorem 5 *Let $(B, P[k], Q[k])$ be a modular domain description, let* `ground` *be an adequate grounder, and let $(P_i, O_i)_{i\geq 0}$ and $(Q_i, O'_i)_{i\geq 1}$ as in Definition 4. If (P_j, O_j) is defined for all $0 \leq j \leq i$ and if (Q_i, O'_i) is defined, we have the following for $i \geq 1$:*[5]

$$CF(\bigcup_{0\leq j\leq i} P_j \cup Q_i, grd(\mathcal{A})) \equiv \bigcup_{0\leq j\leq i} CF(P_j, O_j) \cup CF(Q_i, grd(\mathcal{A}) \setminus \bigcup_{0\leq j\leq i} O_j)\,,$$

$$LF(\bigcup_{0\leq j\leq i} P_j \cup Q_i) \equiv \bigcup_{0\leq j\leq i} LF(P_j) \cup LF(Q_i|_{head(\bigcup_{0\leq j\leq i} P_j \cup Q_i)})\,.$$

The above assertions are obtained from structural properties resulting from (incremental) module theory together with the assumption that `ground` is adequate. First, the fact that all programs P_j for $j \geq 0$ and $Q_i|_{head(\bigcup_{0\leq j\leq i} P_j \cup Q_i)}$ for $i \geq 1$ have disjoint head atoms allows us to decompose the completion of the entire program into "slice-wise" program completions. Second, the definition of the module join guarantees that all circular dependencies are confined to single program slices. As a consequence, loop formulas do not spread over multiple programs, so that they can also be combined in a "slice-wise" manner. The practical consequence of the decomposability of program completion and loop formulas is that a solver can successively build its data structures in a modular way.

When processing consecutive program slices, we have to distinguish cumulative and volatile ones. That is, while the ground rules in P_j are accumulated within the solver for $0 \leq j \leq i$, the ones in Q_j must be discarded for $1 \leq j < i$ when Q_i is added. We accomplish this by adding to each rule in Q_j a new body atom α_j, along with rules achieving that α_j holds only at step j. To this end, we define the following set of rules for a program Q over $grd(\mathcal{A})$ and a new atom $\alpha \notin grd(\mathcal{A})$:

$$Q(\alpha) = \{head(r) \leftarrow body(r) \cup \{\alpha\} \mid r \in Q\}\,.$$

[5] We abuse notation and let $\equiv$ stand for logical equivalence here.

In our incremental setting, the addition of new atoms allows us to selectively (de)activate volatile program slices.

Proposition 6 *Let $(P_i)_{i\geq 0}$ and $(Q_i)_{i\geq 1}$ be sequences of logic programs over $grd(\mathcal{A})$, and let $F_j = \{\alpha_j \leftarrow\}$ for $\alpha_j \notin grd(\mathcal{A})$ and $j \geq 1$. Then, we have the following for $i \geq 1$:*

$$\bigcup_{0\leq j\leq i} P_j \cup Q_i \cup F_i \;\equiv\; P_0 \cup \bigcup_{1\leq j\leq i}(P_j \cup Q_j(\alpha_j)) \cup F_i\,.$$

The addition of F_i on the left hand side is merely for establishing formal equivalence, considering that α_i does not occur in Q_i but only in $Q_i(\alpha_i)$. The fact that programs $Q_j(\alpha_j)$ behave neutrally, as long as α_j is underivable, provides us with a handle to control the effective program slices.

In addition to activating some $Q_j(\alpha_j)$ for $j \geq 1$, we also have to deactivate it in subsequent steps. Thus, a solver cannot include α_j persistently as a fact. But rather than explicitly deleting any fact (or rule) previously passed to the solver, we build upon an interface supporting *assumptions*.[6] This trims the required solver interface to only two functions:

- `add`(P) incorporates a ground logic program P into the rule database of the solver;
- `solve`(L) takes a set L of ground literals and computes the answer sets X of the ground logic program comprised in the solver that satisfy $L^+ \subseteq X$ and $L^- \cap X = \emptyset$.

This simple interface is similar to the one for incremental SAT solving given in (Eén and Sörensson 2003). The literals L passed to `solve` constitute assumptions, which can semantically be viewed as the following set of integrity constraints: $\{\leftarrow not\ a \mid a \in L^+\} \cup \{\leftarrow a \mid a \in L^-\}$. However, as regards *clasp*, the crucial difference between integrity constraints and assumptions is that the former give rise to permanent program simplifications, while the effect of the latter is temporary, i.e., restricted to an invocation of `solve`.

Let us now situate the solver in our incremental context.

Definition 5 *Let $(R_i)_{i\geq 0}$ and $(L_i)_{i\geq 0}$ be sequences of logic programs and literals, respectively, over $grd(\mathcal{A}) \cup \{\alpha_i \mid i \geq 0\}$. We define a solver as a pair consisting of the total functions*

$$\texttt{add} : R_i \mapsto S_i \;\; \textit{and}$$
$$\texttt{solve} : L_i \mapsto \chi\,, \;\textit{where}$$

$S_0 = R_0|_{head(R_0)}$, $S_i = S_{i-1} \cup R_i|_{head(S_{i-1}\cup R_i)}$ for $i \geq 1$, and $\chi \subseteq 2^{(grd(\mathcal{A})\cup\{\alpha_i \mid i\geq 0\})}$.

Note that only `add` affects a solver's state, where added programs are again subject to simplification. In fact, as with P_i for $i \geq 0$ in Definition 4, we assume that atoms not occurring as the head of any rule are eliminated. Even if such an atom becomes derivable later on when another program is added, it can thus not interact with the rules already present. The reason for this design decision is that, although operating in an open environment, the possible addition of knowledge or program slices, respectively, should not force the solver to continuously rebuild its existent data structures. Of course, this necessitates program slices to be provided in an bottom-up manner. The second function, `solve`, leaves the

[6] Within *clasp* (Gebser et al. 2007a), the heads of facts are subducted during preprocessing, making the later deletion of facts impossible. In order to support it, special treatment would be needed.

accumulated program slices (logically) unaffected, that is, the passed literals are only assumed locally within `solve`.

The objective of, once added, maintaining program slices also motivates the following definition of *soundness*.

Definition 6 *We define a solver as in Definition 5 as sound, if for all sequences $(R_i)_{i\geq 0}$ and $(L_i)_{i\geq 0}$ of logic programs and literals, respectively, over $grd(\mathcal{A}) \cup \{\alpha_i \mid i \geq 0\}$ and every $i \geq 0$, the following holds:*

$X \in \texttt{solve}(L_i)$ iff $L_i^+ \subseteq X \subseteq atom(S_i) \setminus L_i^-$ such that $X \models \bigcup_{0\leq j\leq i}(CF(R_j|_{Y_j}, head(R_j|_{Y_j})) \cup LF(R_j|_{Y_j}))$, where $Y_0 = head(R_0)$ and $Y_j = head(S_{j-1} \cup R_j)$ for $1 \leq j \leq i$.

First, observe that literals passed as assumptions in L_i must be respected by solutions X returned by a sound solver. Second, X must satisfy the completion and loop formulas individually for each program slice, thereby, restricting the attention to the respective head atoms. This conception allows the solver to build its data structures in a modular way, without sacrificing soundness, but it also relocates the responsibility to properly partition a program away from the solver. However, as Theorem 5 shows, modular domain descriptions (along with an adequate grounder) permit the construction of a program's completion and loop formulas locally for program slices, obtaining the same answer sets as with the entire program.

We now define the program slices to be added to the solver for the ground rules obtained from a domain description.

Definition 7 *Let $(B, P[k], Q[k])$ be a domain description, let `ground` be a grounder, and let $(P_i, O_i)_{i\geq 0}$ and $(Q_i, O_i')_{i\geq 1}$ as in Definition 4.*

If (P_0, O_0), (P_j, O_j), and (Q_j, O_j') are defined for all $1 \leq j \leq i$, we define a sequence $(R_i)_{i\geq 0}$ of logic programs and a sequence $(L_i)_{i\geq 0}$ of literals for $1 \leq j \leq i$ by:

$$\begin{array}{ll} R_0 = P_0 & R_j = P_j \cup Q_j(\alpha_j) \cup \{\{\alpha_j\} \leftarrow\} \cup \{\leftarrow \alpha_{j-1}\} \\ L_0 = \emptyset & L_j = \{\alpha_j\}\,, \end{array}$$

where $\alpha_{j-1}, \alpha_j \notin grd(\mathcal{A})$.

The difference between the cumulative rules in P_j and the volatile ones in Q_j is that an additional atom α_j is appended to the bodies of the latter. Moreover, choice rule $\{\alpha_j\} \leftarrow$ nominally permits the unconditional inclusion of α_j in an answer set. However, upon the invocation of `solve` in step j, literal α_j is passed as assumption, so that answer sets must necessarily contain α_j. In contrast, in step $j+1$, integrity constraint $\leftarrow \alpha_j$ is persistently added to the solver, forcing α_j to be false. Due to this, all rules in Q_j are deactivated in later steps. Notably, *clasp* eliminates such false atoms and rules with false bodies from its data structures, thus deleting the whole obsolete program Q_j.

Formally, however, no added rule is deleted later on, so we require an additional condition.

Definition 8 *We define a domain description $(B, P[k], Q[k])$ as separated, if for all $i \geq 1$ and $j > i$, $head(grd(Q[i])) \cap head(grd(P[j] \cup Q[j])) = \emptyset$.*

Separation can easily be achieved by using distinct predicates and parameter k in the heads of rules in $Q[k]$ as well as in respective body atoms. The domain descriptions given in (2) and (4), trivially, are separated.

Using an adequate grounder and a sound solver, we finally establish that our incremental solving strategy leads to the desired outcomes for modular domain descriptions.

Theorem 7 *Let $(B, P[k], Q[k])$ be a separated modular domain description, let `ground` be an adequate grounder, and let $(P_i, O_i)_{i\geq 0}$ and $(Q_i, O_i')_{i\geq 1}$ as in Definition 4. Furthermore, let $(\texttt{add}, \texttt{solve})$ be a sound solver, $(R_i)_{i\geq 0}$ and $(L_i)_{i\geq 0}$ as in Definition 7, and $S_j = \texttt{add}(R_j)$ for $j \geq 0$ as in Definition 5. If (P_0, O_0), (P_j, O_j), and (Q_j, O_j') are defined for all $1 \leq j \leq i$, we have the following for $i \geq 1$:*

$X \in \texttt{solve}(L_i)$ iff $(X \setminus \{\alpha_i\}) \in AS(\bigcup_{0\leq j\leq i} P_j \cup Q_i)$.

Comparing with the third item in Theorem 4 shows that our collective approach, comprising incremental grounding and solving, matches exactly the semantics of (programs induced by) separated modular domain descriptions (cf. Definition 2). In this context, the modularity condition allows us to largely reuse existing ASP technology, namely, the grounder *gringo* and the solver *clasp*, for implementing our approach. In fact, the data structures of *clasp* profoundly rely on a program's completion and loop formulas, so that their decomposability is a major benefit. Certainly, a more sophisticated incremental ASP system could drop the modularity assumption, but it would be likely to require intricate rectification work in-between incremental solving steps.

Algorithm Algorithm 1 combines our grounding and solving functions for successively computing the answer sets of programs induced by a domain description $(B, P[k], Q[k])$. To this end, `isolve` makes use of one instance of a grounder, denoted by GROUNDER, and one instance of a solver, viz., SOLVER. Programs B, $P[i]$, and $Q[i]$ are then gradually grounded by means of GROUNDER, starting from $i = 1$. Provided that GROUNDER can instantiate the given programs, i.e., if they fulfill any additional requirements GROUNDER might impose, the obtained ground programs are fed into SOLVER by way of its function `add`. In Line 7, 10, and 11 of Algorithm 1, cumulative and volatile program slices are handled according to the sequences of programs and assumptions, respectively, specified in Definition 7. Note that `isolve` terminates as soon as function `solve` of SOLVER reports some answer set. Otherwise, if no answer set is found in any step $i \geq 1$, `isolve` (in theory) loops forever.

Provided that GROUNDER is adequate and that SOLVER is sound, for a separated modular domain description $(B, P[k], Q[k])$ such that $P(\mathbb{Q}_i)$ (cf. Definition 2) has an answer set for some $i \geq 1$, `isolve` returns the answer sets of $P(\mathbb{Q}_i)$ for the least $i \geq 1$ for which $P(\mathbb{Q}_i)$ has some answer set.

Theorem 8 *Let $(B, P[k], Q[k])$ be a separated modular domain description, let* GROUNDER *be an adequate grounder, and let* SOLVER *be a sound solver. Let $(P_i, O_i)_{i\geq 0}$ and $(Q_i, O_i')_{i\geq 1}$ as in Definition 4 for `ground` = GROUNDER.`ground`, and let $(\mathbb{Q}_i)_{i\geq 1}$ as in Definition 2.*

If (P_0, O_0), (P_i, O_i), and (Q_i, O_i') are defined for all $i \geq 1$, we have $\texttt{isolve}((B, P[k], Q[k])) = AS(P(\mathbb{Q}_i))$ for the least $i \geq 1$ such that $AS(P(\mathbb{Q}_i)) \neq \emptyset$.

Note that the above result builds upon the assumption that

Algorithm 1: `isolve`

Input : A domain description $(B, P[k], Q[k])$.
Output : A nonempty set of answer sets.
Internal: A grounder GROUNDER and a solver SOLVER.

```
1  (P_0, O) ← GROUNDER.ground(B, ∅)
2  SOLVER.add(P_0)
3  i ← 0
4  loop
5      i ← i + 1
6      (P_i, O_i) ← GROUNDER.ground(P[i], O)
7      SOLVER.add(P_i)
8      O ← O ∪ O_i
9      (Q_i, O'_i) ← GROUNDER.ground(Q[i], O)
10     SOLVER.add(Q_i(α_i) ∪ {{α_i} ←} ∪ {← α_{i−1}})
11     χ ← SOLVER.solve({α_i})
12     if χ ≠ ∅ then return {X \ {α_i} | X ∈ χ}
```

$(B, P[k], Q[k])$ is modular. However, modularity is not enforced within `isolve`, so it could be fed with non-modular domain descriptions too. In such a case, if GROUNDER is able to instantiate B, $P[i]$, and $Q[i]$ for all $i \geq 1$, the result of SOLVER.`solve` are interpretations satisfying the "slice-wise" completion and loop formulas as required for the soundness of a solver (cf. Definition 6). But since compositionality in the sense of Theorem 5 is not guaranteed, the interpretations returned by SOLVER.`solve` and thus by `isolve` do not necessarily match the answer sets of the combined program accumulated within SOLVER. In order to avoid such peculiar behavior, users should take care to model their incremental problems in a modular way.

We next provide simple syntactic conditions under which B, $P[k]$, and $Q[k]$ assemble a modular domain description.

Proposition 9 *Let $(B, P[k], Q[k])$ be a domain description, and let $\mathbf{P} = \bigcup_{i\geq 1} P[i]$ and $\mathbf{Q} = \bigcup_{i\geq 1} Q[i]$. Then, $(B, P[k], Q[k])$ is modular if the following conditions hold:*

1. $atom(grd(B)) \cap (head(grd(\mathbf{P})) \cup head(grd(\mathbf{Q}))) = \emptyset$,
2. $atom(grd(\mathbf{P})) \cap head(grd(\mathbf{Q})) = \emptyset$, *and*
3. $\{head(grd(P[i])) \mid i \geq 1\}$ *is a partition of* $head(grd(\mathbf{P}))$.

Pragmatically, these conditions can be granted by using predicates not occurring in $B \cup P[k]$ for the heads of rules in $Q[k]$ and by including 0 as a parameter in every atom of B as well as parameter k in the head of every rule in $P[k]$. Of course, parameter 0 can also be omitted in atoms of B if the corresponding predicates are not used in the heads of rules in $P[k]$. Recalling the domain descriptions given in the introductory section and (4), one can observe that the respective programs B, $P[k]$, and $Q[k]$ fit into this scheme. Thus, we immediately conclude that both of them are modular.

For illustrating the proceeding of `isolve`, reconsider the example given in the introduction. Figure 1 shows the accumulation of ground rules within the solver during the formation of this answer set. While computing an answer set fails for rules added in $i = 0, 1$, we get the previous answer set from the entire set of rules. The other difference between both solving steps is that the first is done assuming α_1, while the second one is accomplished under the assumption of α_2.

i	Rules				L
0	B	$p(0)$	$\leftarrow$	$not\ \neg p(0)$	
		$\neg p(0)$	$\leftarrow$	$not\ p(0)$	
			$\leftarrow$	$p(0), \neg p(0)$	
1	$P[1]$	$a(1)$	$\leftarrow$	$not\ \neg a(1)$	
		$\neg a(1)$	$\leftarrow$	$not\ a(1)$	
		$p(1)$	$\leftarrow$	$a(1)$	
		$p(1)$	$\leftarrow$	$p(0), not\ \neg p(1)$	
		$\neg p(1)$	$\leftarrow$	$\neg p(0), not\ p(1)$	
			$\leftarrow$	$p(1), \neg p(1)$	
			$\leftarrow$	$a(1), \neg a(1)$	
	$Q[1](\alpha_1)$		$\leftarrow$	$not\ \neg p(0), \alpha_1$	α_1
			$\leftarrow$	$not\ p(1), \alpha_1$	
			$\leftarrow$	$not\ \neg a(1), \alpha_1$	
		$\{\alpha_1\}$	$\leftarrow$		
			$\leftarrow$	α_0	
2	$P[2]$	$a(2)$	$\leftarrow$	$not\ \neg a(2)$	
		$\neg a(2)$	$\leftarrow$	$not\ a(2)$	
		$p(2)$	$\leftarrow$	$a(2)$	
		$p(2)$	$\leftarrow$	$p(1), not\ \neg p(2)$	
		$\neg p(2)$	$\leftarrow$	$\neg p(1), not\ p(2)$	
			$\leftarrow$	$p(2), \neg p(2)$	
			$\leftarrow$	$a(2), \neg a(2)$	
	$Q[2](\alpha_2)$		$\leftarrow$	$not\ \neg p(0), \alpha_2$	α_2
			$\leftarrow$	$not\ p(2), \alpha_2$	
			$\leftarrow$	$not\ \neg a(2), \alpha_2$	
		$\{\alpha_2\}$	$\leftarrow$		
			$\leftarrow$	α_1	

Figure 1: Accumulation of ground rules within the solver.

The Incremental ASP System *iclingo*

We implemented our approach to incremental ASP solving within the system *iclingo*[7,8] by building on the ASP grounder *gringo*[9] (Gebser, Schaub, and Thiele 2007) and the ASP solver *clasp*[10] (Gebser et al. 2007a). Both systems are provided to *iclingo* as libraries and had to be extended for accommodating the additional functionality needed for supporting our incremental approach. The architecture of *iclingo* is depicted in Figure 2, concentrating on the interaction of *gringo* and *clasp* with Algorithm 1 (`isolve`).

In general, *gringo* accepts so-called λ-restricted programs, guaranteeing a finite equivalent ground instantiation. The input language was extended by a declaration statement of the form `#parameter k`, indicating that `k` is a parameter. The major extension of *gringo*, however, was the enhancement of *gringo*'s database system for dealing with successive grounder states (indicated by the boxed O in Figure 2). Roughly speaking, this database now maintains all heads of previously grounded rules; logically it corresponds to the content of variable O in Algorithm 1. A major difficulty in adapting *gringo* to an incremental setting was that its design is predicate-oriented and consequently not foresees a slice-wise generation of a predicate's ground

[7] `http://www.cs.uni-potsdam.de/iclingo`
[8] *iclingo* stands for *i*ncremental *cl*asp and gr*ingo*.
[9] `http://www.cs.uni-potsdam.de/gringo`
[10] `http://www.cs.uni-potsdam.de/clasp`

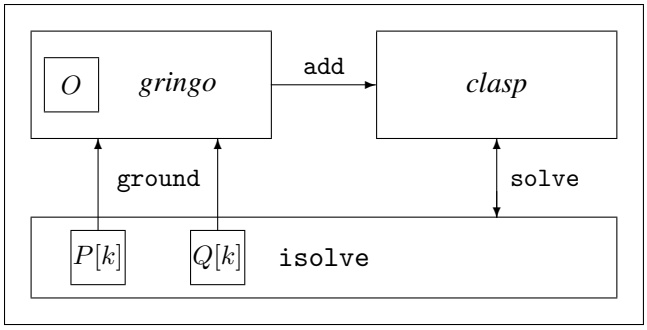


Figure 2: Architecture of *iclingo*.

instances. We addressed this problem in *iclingo* by dynamically viewing all predicates as completely instantiated after processing each program slice. Another consequence of the underlying predicate-oriented approach is that *iclingo* currently stipulates that the predicate symbols appearing in the heads of $P[k]$ and $Q[k]$ are disjoint (cf. the paragraph below Proposition 9). Procedurally, *iclingo* uses *gringo* as delineated in Algorithm 1. Notably, (the representations of) the parametrized rules in $P[k]$ and $Q[k]$ are kept *un*instantiated in *iclingo* (indicated by the boxed $P[k]$ and $Q[k]$ in Figure 2) and are only grounded on demand.[11] When `isolve` makes *gringo* ground a program slice, the result is automatically channeled to *clasp*. Interestingly, this is done in a rule-wise fashion instead of transferring an entire ground program (as in Algorithm 1). Once completed, `isolve` launches *clasp* with the appropriate assumptions and waits for the result.

As indicated in Theorem 5, the customization of *clasp* conceptually affects two components, namely, the treatment of a program's completion and loop formulas, respectively. While the latter is accomplished by providing an incremental construction of the positive dependency graphs needed by the unfounded set checker, the former must exempt assumptions from building corresponding completion formulas. To this end, the actual solver interface contains two further functions, viz., `freeze` and `unfreeze`. Logically, they amount to adding or removing, respectively, a choice rule, as done in Algorithm 1 via the addition of $\{\alpha_i\} \leftarrow$ and $\leftarrow \alpha_{i-1}$. Note that neither of these adaptations would be necessary in a SAT solver, since the underlying semantics does not rely on Clark's completion. Over time, *clasp* accumulates the ground program slices and, moreover, learns further constraints during solving. As a matter of fact, it is equipped with dynamic deletion and simplification techniques disposing of invalid and superfluous constraints. Hence, unfreezing a former assumption α_{i-1} automatically results in eliminating all constraints containing α_{i-1}. In this way, we are able to exploit *clasp*'s inherent constraint maintenance system for keeping only relevant information.

Experiments. We have conducted experiments on a variety of parametrized benchmark classes. We consider *iclingo* (including *gringo* 1.0.0 and *clasp* 1.0.5) in four settings: (1) keeping learned nogoods and heuristic values, (2) keeping learned nogoods only, (3) keeping heuristic values only, and (4) keeping neither over successive solving steps. We compare these variants with iterative deepening search using *clingo*, the direct combination of *gringo* (1.0.0) and *clasp* (1.0.5) via an internal interface, *gringo* (1.0.0) and *clasp* (1.0.5) communicating via a textual interface (using the output language of *lparse*), and finally, the combination of *lparse*[12] (1.1.1) and *smodels*[12] (2.32) via the same textual interface.

The benchmarks in Table 1 consider four different classes.[13]. The goal of the Blocksworld example is to reconstruct a tower of n blocks in inverse order, requiring a plan of length n. In the Queens example, we compute (at most) one answer set for each value of k, iterating from 1 to n. The Towers of Hanoi benchmarks are handmade instances, for which (n) indicates the number of steps needed for obtaining a solution. Finally, we consider a suite of Sokoban examples. Table 1 summarizes run-time results in seconds, taking the average of three runs per instance, each run limited to 1800s on a 3.4GHz PC; the lines marked with Σ show the sums of run-times over all instances of a benchmark class, with timeouts taken as 1800s.

On the Blocksworld and Queens examples, *iclingo* outperforms the other systems by one order of magnitude, which is primarily due to reduced grounding overhead. In fact, all of the considered solvers handle the Blocksworld problems without any search. However, all systems but *iclingo* need to repeat grounding and propagation in each iterative deepening step, working on ground programs of considerable size. For example, considering the Blocksworld problem with four blocks (viz. $n = 4$), *gringo* produces 167 ground rules for describing the first step and 234 ground rules for each further step. While *iclingo* adds this number of rules in each incremental step, resulting in $167 + (n-1) * 234 = 869$ ground rules for $n = 4$, the systems based on iterative deepening (using *gringo*) process $n * 167 + (n * (n-1)/2) * 234 = 2072$ ground rules before obtaining a solution. Of course, the ratio of ground rules processed by *iclingo* in comparison to the other systems gets even smaller as n increases, explaining the dramatic performance gains on Blocksworld examples observed in Table 1. On the Queens example, we observe a similar effect when comparing *iclingo* to systems using iterative deepening, but here the underlying solvers have to search for a solution for $n \geq 4$. Interestingly, for the smaller values of n, *iclingo (1)* is the fastest system, but from $n = 100$, *iclingo (2)* and *iclingo (3)* take the lead and exhibit increasingly significant gains compared to *iclingo (1)*. This somewhat suggests that the strategy of *iclingo (1)* keeping both learned nogoods and heuristic values in-between successive runs here tends to bias future runs too much, which makes sense because different Queens instances are largely independent from each other. Finally, note that lookahead used in *smodels* is likely to be a factor for the many timeouts in the last column, as it

[11] We omitted the base program B in Figure 2, as it is processed only once for providing atoms to O and rules to *clasp*.

[12] `http://www.tcs.hut.fi/Software/smodels`

[13] Taken from the web sites `http://asparagus.cs.uni-potsdam.de` and `http://www.ne.jp/asahi/ai/yoshio/sokoban/handmade`.

merely wastes time here.

Different from the simple Blocksworld and the combinatorial Queens example, Towers of Hanoi and Sokoban contain realistic instances, shifting the focus to search for a plan. In fact, all systems underlie non-deterministic heuristic effects and traverse the search space differently. Though all systems spend most of their run-time in the solving component, the savings in grounding are still noticeable for *iclingo* but smaller than with Blocksworld and Queens. Also note that slightly inferior performance of *clingo* compared to *gringo|clasp* is due to differences in the ordering of ground rules between the internal and the textual interface. With Towers of Hanoi, the differences between the systems are rather small (except for *lparse|smodels*), still, *iclingo (3)* keeping only heuristic values has some advantages. This might be explained by the fact that the problem is explicitly constrained only by the goal state, and learned nogoods depending on it can only be used in the step they were discovered. Finally, on Sokoban, we observe varying relative performance of the considered systems on individual instances, which is due to the elevated difficulty of the problem. Surprisingly, *iclingo (4)* keeping neither learned nogoods nor heuristic values exhibits the best overall performance on Sokoban. We conjecture that this is because constraints postulated via the goal state become obsolete after only one incremental step, which also makes learned nogoods derived from them useless for the following steps. Rather than encoding the goal state in query program Q, we plan to investigate alternative encodings working backwards (Eén and Sörensson 2003), that is, the starting state is part of the query program while the goal state remains fixed.

Discussion

We presented the first theoretical and practical account of incremental ASP solving. Our framework allows for tackling bounded problems in ASP in a so far unexampled manner paving the way for addressing more ambitious real-world applications. Our approach is driven by the desire to minimize redundancies while gradually treating program slices. However, fixing the incremental solving process necessitied the integration and adaption of manifold different concepts in a globally consistent way. To this end, we developed an incremental module theory guiding the formal setting of successive incremental grounding and solving steps by means of existing ASP grounders and solvers. Future work includes more elaborate incremental wrapping algorithms, allowing for non-elementary program slices while guaranteeing optimal solutions.

References

Baral, C.; Brewka, G.; and Schlipf, J., eds. 2007. *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Springer.

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

Brass, S., and Dix, J. 1999. Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic Programming* 40(1):1–46.

Brass, S.; Dix, J.; Freitag, B.; and Zukowski, U. 2001. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming* 1(5):497–538.

Clarke, E.; Biere, A.; Raimi, R.; and Zhu, Y. 2001. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 19(1):7–34.

Eén, N., and Sörensson, N. 2003. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science* 89(4).

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2004. Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, 87–99. Springer.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007a. clasp: A conflict-driven answer set solver. In Baral et al. (2007), 260–265.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007b. Conflict-driven answer set solving. In Veloso, M., ed., *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 386–392. AAAI Press/The MIT Press.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. GrinGo: A new grounder for answer set programming. In Baral et al. (2007), 266–271.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Transactions on Artificial Intelligence* 3(6):193–210.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1-2):49–104.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, 359–363. John Wiley & sons.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming*, 23–37. MIT Press.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, 92–106. Springer.

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

Oikarinen, E., and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, 412–416. IOS Press.

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

Syrjänen, T. Lparse 1.0 user's manual. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.

Name	n	*iclingo (1)*	*iclingo (2)*	*iclingo (3)*	*iclingo (4)*	*clingo*	*gringo|clasp*	*lparse|smodels*
Blocksworld	10	0.40	0.40	0.39	0.39	2.13	3.93	7.04
	15	2.05	2.05	2.02	2.03	16.18	29.21	49.30
	20	6.62	6.63	6.55	6.54	71.32	129.93	200.07
	25	16.81	16.86	16.66	16.67	224.36	406.15	594.49
	30	37.34	37.39	37.07	36.94	575.98	1059.71	1460.67
	35	72.30	72.70	71.93	71.62	1289.58	1800	1800
	Σ	135.52	136.03	134.63	134.20	2179.55	3428.94	4111.56
Queens	20	0.21	0.24	0.29	0.21	0.74	1.11	6.50
	25	0.40	0.50	0.62	0.41	1.76	2.67	759.69
	30	0.71	0.87	1.11	0.75	3.68	5.58	1800
	50	3.73	5.28	6.72	3.78	33.71	48.27	1800
	70	11.98	17.84	23.31	12.27	156.50	205.21	1800
	90	32.75	50.43	64.44	77.39	496.52	643.94	1800
	100	125.28	78.49	96.67	137.14	815.11	1040.30	1800
	110	232.21	117.25	146.11	246.96	1282.13	1640.20	1800
	120	400.26	169.85	224.14	446.24	1782.34	1800	1800
	Σ	807.52	440.74	563.41	925.15	4572.48	5387.65	13366.19
Towers	33	46.08	43.62	40.19	52.06	40.06	50.12	374.88
	34	71.60	54.47	44.39	34.82	69.10	52.05	1064.54
	36	112.34	91.34	88.22	104.11	145.72	112.99	960.60
	39	280.82	214.81	206.03	204.23	175.80	191.18	1800
	41	417.67	453.71	411.37	587.90	523.94	477.59	1800
	Σ	928.51	857.95	790.20	983.11	954.62	883.93	6000.03
Sokoban	17	2.39	2.68	2.82	2.83	7.12	8.64	1800
	13	0.71	0.67	1.35	0.95	3.06	4.21	65.82
	11	1.23	1.16	0.92	0.90	2.93	3.87	70.97
	11	5.51	4.65	4.27	3.65	9.50	12.55	1800
	16	478.67	579.70	503.70	716.57	896.85	593.29	1800
	12	33.88	45.40	41.46	39.28	73.79	58.17	1800
	12	7.40	8.38	6.70	7.02	16.16	28.25	1800
	13	10.02	8.15	15.05	14.09	16.97	17.49	1800
	16	183.90	183.83	383.83	363.23	388.26	377.50	1800
	16	116.87	362.27	321.10	206.10	269.36	298.06	1800
	18	248.70	229.07	232.10	251.47	508.48	493.03	1800
	16	49.59	32.99	37.17	32.48	64.72	91.03	1800
	17	1791.07	1619.40	1511.43	1067.61	1703.53	1755.63	1800
	14	221.30	372.93	252.37	205.77	208.88	256.47	1800
	14	108.27	65.39	98.88	118.20	108.04	103.42	1800
	15	28.43	38.30	19.67	22.50	27.54	38.40	1800
	18	11.54	16.34	13.41	11.49	16.85	24.44	1800
	14	3.26	2.84	3.66	3.60	9.71	11.39	1800
	13	7.77	14.50	11.07	15.35	30.17	24.17	1800
	17	300.23	208.00	209.47	201.50	203.12	166.25	1800
	14	82.18	49.29	85.25	71.79	143.30	172.59	1800
	Σ	3692.90	3845.93	3755.66	3356.37	4708.33	4538.84	34348.24
	$\Sigma\Sigma$	5564.44	5280.65	5243.09	5398.83	12414.98	14239.36	57826.02

Table 1: Benchmark results on a 3.4GHz PC under Linux; each run limited to 1800s time.

Defeasible Knowledge and Argumentative Reasoning for 3APL Agent Programming

Sebastian Gottifredi Alejandro J. Garcia Guillermo R. Simari
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Artificial Intelligence Research and Development Laboratory,
Department of Computer Science and Engineering - Universidad Nacional del Sur (UNS),
Bahía Blanca, ARGENTINA,
e-mail: {sg, ajg, grs}@cs.uns.edu.ar

Abstract

In this work we propose to integrate a defeasible argumentation logic programming formalism for representing beliefs and reasoning into 3APL thus extending its representational capabilities. Using this formalism the agent can represent tentative information in the form of weak rules. Since strong negation is allowed in the head of these rules, contradictory knowledge can be represented. The formalism allows the identification of the pieces of knowledge that are in contradiction and a dialectical argumentation process is used for deciding which information prevails. In particular, the argumentation based definition of the inference relation makes it possible to incorporate a treatment of preferences in an elegant way. This integration increases the capabilities of the components for knowledge representation and reasoning, aiming to the implementation of more sophisticated autonomous agents.

Introduction

The importance of using intelligent agents based on mental components like Beliefs, Desires, Commitments and Intentions to solve complex problems is well known in the literature, especially those agents based on BDI theory (Bratman, Israel, & Pollack). Nowadays, tools are needed to specify and program agents in terms of these components (Fisher *et al.* 2007). In particular, we are interested in agent development tools that provide not only a declarative way to specify agent mental components but also an argumentative mechanism for agent reasoning.

In this work we propose to integrate a defeasible argumentation logic programming formalism into 3APL. Thus, the agent can represent tentative information in the form of weak rules. Since strong negation is allowed in the head of these rules, contradictory knowledge can be represented. This formalism allows the identification of the pieces of knowledge that are in contradiction and a dialectical argumentation process is used for deciding which information prevails. In particular, the argumentation based definition of the inference relation enables to incorporate a treatment of preferences in an elegant way.

The contribution of this work is to provide 3APL agents with:

- a belief representation language that allows strong negation and default negation,
- an argumentative reasoning mechanism used to obtain the inferred beliefs,
- a programmable criteria used by the argumentative mechanism to decide between contradictory conclusions and
- a set of agent capabilities that are combined with the argumentative mechanism.

Thus, with this approach is that the programmer can focus on the rules that represent knowledge in a declarative way and not in the explicit interaction among these rules. Therefore, in order to add a rule the programmer does not need to be aware of the whole knowledge program. When a new rule is added, its interaction with the rest of the program is detected and handled by the argumentative inference mechanism.

When the belief representation language allows strong negation, contradictory information can be inferred. If contradictions occur, a decision criterion is needed to determine which information prevails. Previous approaches that increased the representational capabilities of 3APL belief language used a fixed decision criterion. Here, we propose a mechanism were the decision criterion is based on a programmable argument comparison criterion and changed dynamically without changing the belief program.

Argumentation is recognized in the literature as an interesting approach for reasoning with inconsistent information, based on the construction and the comparison of arguments (Bench-Capon & Dunne 2007; Dung 1995). Thus, argumentation provides a different perspective to nonmonotonic and defeasible reasoning, in which a claim is accepted or withdrawn on the basis of the arguments for and against it, and on whter these arguments can be attacked and defeated by others. There exist previous approaches that relate cognitive agents (especially those that follow the BDI theory) to argumentation frameworks (Rahwan & Amgoud 2006; Rotstein, Garcia, & Simari 2007). However, none of these approaches propose an agent programming language.

In the area of agent programming languages, 3APL (Dastani, van Riemsdijk, & Meyer 2005) is an interesting ap-

Partially supported by CONICET (PIP 5050), UNS and Agencia Nacional de Promocin Cientfica y Tecnolgica.

proach which allows a declarative way to implement cognitive agents. 3APL provides programming constructs for implementing an agent's beliefs, goals, basic capabilities such as belief updates or motor actions, and a set of practical reasoning rules through which the agent's goals can be updated or revised. The 3APL programs are executed by an interpreter that implements a deliberation cycle using such constructs. 3APL combines logic programming (for the specification of the agent mental components) and imperative programming (for the structure of plans). From the imperative programming viewpoint, 3APL inherits the full range of regular programming constructs, including recursive procedures and state-based computation. States of agents in 3APL, however, are belief (or knowledge) bases, which are different from the usual variable assignments of imperative programming. From the computational logic perspective, answers to belief-base queries of a 3APL agent are proofs in the logic programming sense (Fisher *et al.* 2007).

Next, a brief description of 3APL is given (for details see (Dastani, van Riemsdijk, & Meyer 2005; Dastani *et al.* 2003; Hindriks *et al.* 1999)). This description will focus on those issues that will be affected by the proposed approach. To program a 3APL agent means to specify its initial beliefs, goals, plans, capabilities, sets of goal planning rules and plan revision rules. The initial state of the shared environment is specified by a set of facts. A short description of the components of a 3APL agent is given below.

Belief Base (σ): Beliefs, represented by a first order domain language, describe the situation the agent is in. For example, in (Dastani, van Riemsdijk, & Meyer 2005), PROLOG is adopted as the belief representation language.

Goals Base (γ): Goals are used to describe the situation that the agent wants to achieve. In (Dastani, van Riemsdijk, & Meyer 2005), a goal is implemented by a conjunction of ground PROLOG atoms.

Reasoning Rules (PG and PR): Reasoning rules are the mechanisms used to reason about goals and plans. These rules are conditionalized by beliefs and can be divided into two types: goal planning rules (PG-Rules), which are used to generate plans in order to achieve goals and plan revision (PR-Rules), which are used to revise a selected plan.

Capabilities (Cap): Capabilities (also known as mental actions) are rules used to update the belief base. The effect of execution of a mental action is not a change in the world, but a change in the belief base of the agent.

Plan base (Π): Plans are compositions of basic actions using imperative program operators (for more details see (Dastani, van Riemsdijk, & Meyer 2005)). These basic actions are simple actions that can be divided into: mental actions, which are used to update the belief base of the agent; send actions, which are used to pass messages to other agents; external actions, which are used to act or sense the agent external environment; and test actions, which are used to block a plan if some condition is not met.

Although 3APL does not impose any particular knowledge representation language, the beliefs of an agent are expected to be formulas in some logical language. Currently, 3APL adopts PROLOG as such representation language (Dastani, van Riemsdijk, & Meyer 2005) and this choice implies some limitations for knowledge representation and reasoning. For example, no negative facts can be represented and rules can only use negation as failure.

Using DeLP as the Belief Language in 3APL

In this section we will describe how to integrate Defeasible Logic Programming (DeLP) into 3APL for belief representation and deliberation. This integration will be called 3APL-DeLP. The proposed approach will affect the agent reasoning and the way in which knowledge is represented.

Next, we give a brief summary of DeLP (for more details see (Garcia & Simari 2004)). The goal of this summary is to introduce the concepts of DeLP that will be used in this work. Then we will redefine the components of 3APL that will be affected by our proposal. In order to do so, we will introduce the following working example:

Example 1 *Consider an agent that has to move around a city where it can use different alternatives: bus, metro, car or walking. The agent can perceive different elements from its environment that it can use for taking decisions. For example, the agent can perceive if it is snowing, or if the city uses snow plows, or if its destination is downtown. Based on this perceived information the agent can reason which alternative is more convenient for its task. For example if it is snowing, roads may be closed and then the agent will prefer to use the metro. Finally, using its preferences the agent should be able to select one of the possible transport methods. In our case, the agent has the goal of reaching office. It has several possible plans: go walking if the destination is near to its current position and it is not snowing; go by car avoiding highways in rush hour, go by car if it is not snowing, go by bus or go by metro.*

DeLP is a formalism that combines results of Logic Programming and Defeasible Argumentation. DeLP provides the possibility of representing information in the form of rules in a declarative manner, and a defeasible argumentation inference mechanism for warranting the entailed conclusions. These rules are the key element for introducing defeasibility and they will be used to represent a relation between pieces of knowledge that could be defeated after all things are considered. Using these rules, common sense reasoning is defeasible in a way that is not explicitly programmed. Defeat should be the result of a global consideration of the corpus of knowledge of the agent performing the inference. Defeasible Argumentation provides the tools for doing this.

In a Defeasible Logic Program (or DeLP-program for short) knowledge can be represented using:

- *Facts*: ground literals representing atomic information or the negation of atomic information using strong negation "$\sim$" (e. g. $\sim have\_car$, or $closed\_roads$).
- *Defeasible Rules*: denoted $L_0 -\!\prec L_1, \ldots, L_n$, where L_0 is a ground literal and $\{L_i\}_{i>0}$ is a set of ground literals. (e. g. $\sim use\_metro(D) -\!\prec rush\_hour$).

A defeasible rule (d-rule) represents tentative information that may be used if nothing could be posed against it. A d-rule "$Head -\!\prec Body$" expresses that "*reasons to believe*

in the antecedent $Body$ give reasons to believe in the consequent $Head$". Defeasible rules are ground, however, following the usual convention, some examples will use "schematic rules" with variables. When required, the set of facts is denoted Ψ and the set of defeasible-rules Δ.

Strong negation could appear in facts or in the head of defeasible-rules and can be used to represent contradictory knowledge. Observe that from DeLP-program contradictory literals could be derived, however, the set Ψ (used to represent non-defeasible information) must be non-contradictory, *i. e.*no pair of contradictory literals can be derived from Ψ. Given a literal L, $\overline{L}$ represents the complement with respect to strong negation.

As we will define below, the belief base of a 3APL-DeLP agent is an extension of the belief base of a classical 3APL agent. This extended belief base can contain facts for representing positive or negated information, defeasible-rules for representing tentative information and also PROLOG clauses.

Definition 1 *The belief base of a 3APL-DeLP Agent will be a triplet $\mathcal{P}_\sigma$=(Ψ_σ, Δ_σ,$Prog$), where Ψ_σ is a non contradictory set of facts, Δ_σ is a set of defeasible-rules and Prog is a set of* PROLOG *rules.*

It is important to note that given a belief base $\mathcal{P}_\sigma$=(Ψ_σ, Δ_σ,$Prog$), the set of atoms A that can be derived from the PROLOG program $Prog$ are specially considered by the defeasible argumentation analysis performed by DeLP. Thus in order to avoid naming conflicts between DeLP Literals and PROLOG atoms we will assume that both of them are represented with separate names.

Example 2 *Fig. 1 shows part of the 3APL-DeLP implementation of the agent described in Ex. 1. The agent belief base ($\Psi_{\sigma 1}$, Δ_σ, $Prog$) contains facts, defeasible-rules and* PROLOG *clauses for* `distance(office,5)` *and* `near/1` *that succeeds if the agent is near the destination point D. The set $\Psi_{\sigma 1}$ = $\{\sim downtown(office)$, $have\_car$, $snow$, $\sim far\_metro\_station(office)$, $snow\_plow\}$. The set Δ_σ of defeasible-rules expresses reasons for and against using different transportation vehicles. For instance, by the first rule, if roads are closed then there is a reason for using the metro, and by the second rule there is a reason against using the metro when the destination point is far from a metro station. The goal base includes only one goal where the destination is $office$. Observe that only those elements that are relevant to the scope of the paper are included.*

Observe that the set Ψ_σ allows to represent positive and negative information. Since negation is explicit, it is also possible to represent uncertain information when neither an atom nor its negation is included in the belief base. As it will be explained next, for the agent of Ex. 2 the literals $snow$ and $\sim downtown(office)$ will be warranted. However, the agent will be *undecided* about $rush\_hour$ because there will be no warrant for $rush\_hour$ nor $\sim rush\_hour$.

Using the defeasible rules of the Δ_σ set, the agent can infer tentative information. These inferences will be called defeasible derivations and will be defined next.

Definition 2 *Let $\mathcal{P}_\sigma = (\Psi_\sigma, \Delta_\sigma, Prog)$ be a belief base and L a ground literal. A defeasible derivation of L from $\mathcal{P}_\sigma$, denoted $\mathcal{P}_\sigma \sim L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:*

a) L_i is a fact in Ψ_σ, or
b) L_i is a Prolog atom derived from $Prog$, or
c) there exists a defeasible rule R_i in Δ_σ with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i ($j < i$.)

`BeliefBase`
$snow$
$snow\_plow$
$\sim downtown(office)$
$have\_car$
$\sim far\_metro\_station(office)$
`distance(office,5).`
`near(D):-distance(D,X), X<10`

$use\_metro(D) \prec closed\_roads$
$\sim use\_metro(D) \prec far\_metro\_station(D)$
$closed\_roads \prec snow$
$\sim closed\_roads \prec \sim snow$
$\sim closed\_roads \prec snow, snow\_plow$
$use\_bus(D) \prec \sim closed\_roads$
$\sim use\_bus(D) \prec traffic\_jam(D)$
$traffic\_jam(D) \prec rush\_hour$
$\sim traffic\_jam(D) \prec rush\_hour, downtown(D)$
$\sim use\_car(D) \prec traffic\_jam(D)$
$\sim use\_car(D) \prec closed\_roads$
$use\_car(D) \prec have\_car$

`Goalbase`
$transport(office)$

`PG-Rules`
$transport(D) \leftarrow near(D), \sim snow \mid \{Walking\_plan(D)\}$
$transport(D) \leftarrow use\_car(D), not\ snow, rush\_hour \mid \{Car\_plan\_avoiding\_highways(D)\}$
$transport(D) \leftarrow use\_car(D), not\ snow \mid \{Car\_plan(D)\}$
$transport(D) \leftarrow use\_bus(D) \mid \{bus\_plan(D)\}$
$transport(D) \leftarrow use\_metro(D) \mid \{metro\_plan(D)\}$

`Capabilities`
$\{have\_car\}\ car\_broken\ \{\sim have\_car\}$
$\{received(A, inform, X)\}\ add\_preference\ \{X\}$

Figure 1: 3APL-DeLP Agent

Thus from a 3APL-DeLP agent belief base it is possible to defeasibly derive contradictory conclusions. For instance, using the defeasible-rules of Fig. 1, since $\{snow$,$snow\_plow\} \subseteq \Psi_{\sigma 1}$ then both $closed\_roads$ and $\sim closed\_roads$ can be defeasible derived using the third and the fifth d-rule respectively. In DeLP when contradictory literals are derived, a dialectical process is used for deciding which literal prevails. An *argument* for a literal L, denoted $\langle \mathcal{A}, L\rangle$, is a minimal non-contradictory set of defeasible-rules $\mathcal{A} \subseteq \Delta_\sigma$, that allows to derive L. Next we will define that structure

Definition 3 *Let h be a literal, and $\mathcal{P}_\sigma = (\Psi_\sigma, \Delta_\sigma, Prog)$ a belief base. We say that $\langle \mathcal{A}, h\rangle$ is an argument for h, if $\mathcal{A}$ is a set of defeasible rules of Δ_σ, such that:*

1. *there exists a defeasible derivation for h from $\mathcal{P}_\sigma$,*
2. *the set $\Psi \cup \mathcal{A}$ is non-contradictory, and*
3. *$\mathcal{A}$ is minimal: there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ such that $\mathcal{A}'$ satisfies conditions (1) and (2).*

an argument $\langle \mathcal{B}, q \rangle$ is a sub-argument of an argument $\langle \mathcal{A}, h \rangle$, iff $\mathcal{B} \subseteq \mathcal{A}$

For example, from the belief base of Fig.1 the following arguments can be constructed: $\mathcal{A}_1$=$\{closed\_roads \prec snow\}$ for $closed\_roads$, and $\mathcal{A}_2$=$\{\sim closed\_roads \prec snow, snow\_plow\}$ for $\sim closed\_roads$.

In DeLP, a literal L is *warranted* if there exists a non-defeated argument $\mathcal{A}$ supporting L. To establish if $\langle \mathcal{A}, L \rangle$ is a non-defeated argument, *defeaters* for $\langle \mathcal{A}, L \rangle$ are considered. A defeater is a counter-argument that is preferred to $\langle \mathcal{A}, L \rangle$ by some argument comparison criterion. Counter-arguments of $\langle \mathcal{A}, L \rangle$ are those arguments that *disagree* (are in contradiction) in some point with $\langle \mathcal{A}, L \rangle$.

Definition 4 *We say that $\langle \mathcal{A}_1, h_1 \rangle$* counter-argues*,* rebutts*, or* attacks *$\langle \mathcal{A}_2, h_2 \rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that h and h_1* disagree*.*

Following our example, $\mathcal{A}_2$ is a counter-argument for $\mathcal{A}_1$ (and viceversa) because both support contradictory conclusions.

Given an argument $\langle \mathcal{A}_1, h_1 \rangle$ and a counter-argument $\langle \mathcal{A}_2, h_2 \rangle$ for $\langle \mathcal{A}_1, h_1 \rangle$ these two arguments can be compared in order to decide which one prevails. This is done by the comparison criterion that defines a partial order among the arguments. It is important to note that in DeLP the argument comparison criterion is modular and thus, the most appropriate criterion for the domain that is being represented can be selected. The comparison criterion is defined by the following function

Definition 5 *Let $\mathcal{P}_\sigma$ be the belief base and $Args$ the set of arguments that can be obtained from $\mathcal{P}_\sigma$. The comparison criterion $\leq \subseteq Args \times Args$ and is any partial order on $Args$*

Next, we will use the comparison criterion called *generalized specificity* (Stolzenburg *et al.* 2003), a criterion that favors two aspects of an argument: it prefers (1) a *more precise* argument (*i. e.*with greater information content) or (2) a *more concise* argument (*i. e.*with less use of rules). Thus, following our example and using *generalized specificity* as the comparison criterion we got that $\mathcal{A}_1 \leq \mathcal{A}_2$ because $\mathcal{A}_2$ is more specific than $\mathcal{A}_1$. Below, in this section, we will describe how the criterion comparison criterion can be implemented.

A defeater $\mathcal{D}$ for an argument $\mathcal{A}$ using the comparison criterion can be *proper* if $\mathcal{D}$ is preferred to $\mathcal{A}$ or *blocking* (same strength) if neither argument is better, nor worse, than the other.

Definition 6 *Let $\langle \mathcal{A}_1, h_1 \rangle$ and $\langle \mathcal{A}_2, h_2 \rangle$ be two arguments and $\leq$ the comparison criterion. $\langle \mathcal{A}_1, h_1 \rangle$ is a* proper defeater *for $\langle \mathcal{A}_2, h_2 \rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that $\langle \mathcal{A}_1, h_1 \rangle$ counter-argues $\langle \mathcal{A}_2, h_2 \rangle$ at h, $\langle \mathcal{A}_1, h_1 \rangle \leq \langle \mathcal{A}, h \rangle$ and $\langle \mathcal{A}, h \rangle \not\leq \langle \mathcal{A}_1, h_1 \rangle$*

Definition 7 *Let $\langle \mathcal{A}_1, h_1 \rangle$ and $\langle \mathcal{A}_2, h_2 \rangle$ be two arguments and $\leq$ the comparison criterion. $\langle \mathcal{A}_1, h_1 \rangle$ is a* blocking defeater *for $\langle \mathcal{A}_2, h_2 \rangle$ at literal h, if and only if there exists a sub-argument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that $\langle \mathcal{A}_1, h_1 \rangle$ counter-argues $\langle \mathcal{A}_2, h_2 \rangle$ at h, and $\langle \mathcal{A}_1, h_1 \rangle$ is unrelated by the preference order to $\langle \mathcal{A}, h \rangle$,* i. e.*, $\langle \mathcal{A}_1, h_1 \rangle \leq \langle \mathcal{A}, h \rangle$, and $\langle \mathcal{A}, h \rangle \leq \langle \mathcal{A}_1, h_1 \rangle$.*

In our example, $\mathcal{A}_1 \leq \mathcal{A}_2$ and $\mathcal{A}_2 \not\leq \mathcal{A}_1$ since $\mathcal{A}_2$ is strictly more specific than $\mathcal{A}_1$, therefore, $\mathcal{A}_2$ is a proper defeater for $\mathcal{A}_1$. A defeater can attack the conclusion of an argument or an inner point of it. For example, consider

$$\mathcal{A}_3 = \left\{ \begin{array}{l} use\_metro(office) \prec closed\_roads \\ closed\_roads \prec snow \end{array} \right\}$$

The argument $\mathcal{A}_2$ that supports $\sim closed\_roads$ is a proper defeater for $\mathcal{A}_3$ that supports $use\_metro(office)$ because $\mathcal{A}_2$ attacks an inner point of $\mathcal{A}_3$ ($closed\_roads$).

Since defeaters are arguments, there may exist defeaters for them, and defeaters for these defeaters, and so on. Thus, a sequence of arguments called *argumentation line* can arise.

Definition 8 *Let $\mathcal{P}_\sigma$ be a belief base and $\langle \mathcal{A}_0, h_0 \rangle$ an argument obtained from $\mathcal{P}_\sigma$. An* argumentation line *for $\langle \mathcal{A}_0, h_0 \rangle$ is a sequence of arguments from $\mathcal{P}_\sigma$, denoted Λ= [$\langle \mathcal{A}_0, h_0 \rangle$, $\langle \mathcal{A}_1, h_1 \rangle$, $\langle \mathcal{A}_2, h_2 \rangle$, $\langle \mathcal{A}_3, h_3 \rangle$,...], where each element of the sequence $\langle \mathcal{A}_i, h_i \rangle$, $i > 0$, is a defeater of its predecessor $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$.*

Clearly, for a particular argument there might be more than one defeater. Therefore, many argumentation lines could arise from one argument, leading to a tree structure called *dialectical tree* (Garcia & Simari 2004). In a dialectical tree, every node (except the root) is a defeater of its parent, and leaves are non-defeated arguments.

Definition 9 *Let $\langle \mathcal{A}_0, h_0 \rangle$ be an argument from a belief base $\mathcal{P}_\sigma$. A dialectical tree for $\langle \mathcal{A}_0, h_0 \rangle$, denoted $\mathcal{T}\langle \mathcal{A}_0, h_0 \rangle$, is defined as follows:*

1. *The root of the tree is labeled with $\langle \mathcal{A}_0, h_0 \rangle$.*
2. *Let N be a non-root node of the tree labeled $\langle \mathcal{A}_n, h_n \rangle$, and Λ= $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ the sequence of labels of the path from the root to N. Let $\langle \mathcal{B}_1, q_1 \rangle$, $\langle \mathcal{B}_2, q_2 \rangle$, ..., $\langle \mathcal{B}_k, q_k \rangle$ be all the defeaters for $\langle \mathcal{A}_n, h_n \rangle$. For each defeater $\langle \mathcal{B}_i, q_i \rangle$ $(1 \leq i \leq k)$, such that, the argumentation line $\Lambda' = [\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}_i, q_i \rangle]$ is acceptable, then the node N has a child N_i labeled $\langle \mathcal{B}_i, q_i \rangle$. If there is no defeater for $\langle \mathcal{A}_n, h_n \rangle$ or there is no $\langle \mathcal{B}_i, q_i \rangle$ such that Λ' is acceptable, then N is a leaf.*

A dialectical tree provides a structure for considering all the possible acceptable argumentation lines that can be generated. In a dialectical tree every node can be marked as *defeated* (D) or *undefeated* (U): leaves are marked as undefeated nodes, and inner nodes are marked as defeated when there is at least a child marked as undefeated, or are marked as undefeated when all its children are marked as defeated.

Definition 10 *Let $\mathcal{P}_\sigma$ be a belief base of a 3APL-DeLP agent, $\leq$ the comparison criterion, $\langle \mathcal{A}, h \rangle$ be an argument*

from $\mathcal{P}_\sigma$ and $\mathcal{T}^*\langle\mathcal{A}, h\rangle$ its associated marked dialectical tree. The literal h is a warranted belief *from $\mathcal{P}_\sigma$ using the comparison criterion "$\leq$" (noted $(\mathcal{P}_\sigma, \leq) \mid\sim_w h$) iff the root of $\mathcal{T}^*\langle\mathcal{A}, h\rangle$ is marked as "U". We will say that $\mathcal{A}$ is a warrant for h.*

If it is the case that the literal L is not a warranted belief it will be noted as $\mathcal{P}_\sigma \not\mid\sim_w L$. Observe that facts from Ψ_σ and the atoms that are derived from the PROLOG program $Prog$ are always warranted, since no argument can attack a fact or a PROLOG atom.

Example 3 *Consider the agent of Ex. 2. From the belief base of Fig. 1 the warranted beliefs are: $use\_bus(office)$, $use\_car(office)$, $\sim closed\_roads$, $distance(office, 5)$, $near(office)$ and the elements of $\Psi_{\sigma 1}$. That is, in this situation using the bus or using the car are warranted options.*

Example 4 *Consider now that the agent of Ex. 2 is in a different situation and its set of facts is changed to the following set $\Psi_{\sigma 2}$ = { $snow$, $\sim far\_metro\_station(office)$, $rush\_hour$, $\sim snow\_plow$, $have\_car$, $\sim downtown(office)$ }. Since the situation has changed, the set of warranted belief differs from the ones of Ex. 3. From the belief base ($\Psi_{\sigma 2}$,Δ_σ,$Prog$) the warranted beliefs are: $use\_metro(office)$, $\sim use\_bus(office)$, $near(office)$, $distance(office, 5)$, $traffic\_jam(office)$, $closed\_roads$ and the elements of $\Psi_{\sigma 2}$. Therefore, in this situation the agent infers that using the metro is a warranted option.*

Since the comparison criterion is modular, different warrants can be obtained from a belief base if the comparison criterion is changed.

Example 5 *Consider for instance a new domain dependant comparison criterion that is based on generalized specificity but prefers arguments that use the literal $have\_car$ because of a special property of the city. That is, if $\langle\mathcal{A}_1, h_1\rangle$ and $\langle\mathcal{A}_2, h_2\rangle$ are two conflictive arguments then this new comparison criterion behaves as generalized specificity unless the literal $have\_car$ is used in $\langle\mathcal{A}_1, h_1\rangle$ or $\langle\mathcal{A}_2, h_2\rangle$İf $have\_car$ is used in $\langle\mathcal{A}_1, h_1\rangle$ then $\langle\mathcal{A}_2, h_2\rangle$ leq $\langle\mathcal{A}_1, h_1\rangle$. Consider now that the agent is in the situation described in the example 4 and the above comparison criterion is used. Since the comparison criterion has changed, the set of warranted belief differs from the ones of Ex. 4. From the belief base ($\Psi_{\sigma 2}$,Δ_σ,$Prog$) the warranted beliefs with this new criterion are: $use\_car(office)$ $use\_metro(office)$, $\sim use\_bus(office)$, $near(office)$, $distance(office, 5)$, $traffic\_jam(office)$, $closed\_roads$ and the elements of $\Psi_{\sigma 2}$. Therefore, in this situation the agent infers that using the metro or using the car are warranted options.*

In order to create a comparison criterion the programmer should implement the predicate is_better_than, that receives two arguments $\langle\mathcal{A}_1, h_1\rangle$ and $\langle\mathcal{A}_2, h_2\rangle$ returns true if $\langle\mathcal{A}_1, h_1\rangle$ is better or equal than $\langle\mathcal{A}_2, h_2\rangle$, or false otherwise. This predicate will be called by the DeLP engine when the comparison is needed.

In order to take advantage of the features of DeLP, next we will introduce the modifications to those components of 3APL that are affected by our proposed approach.

As stated above, 3APL provides PG-rules to reason about plans and goals. A PG-rule involves a goal, a guard and a plan. The plan specifies the actions needed to solve the goal and the guard is the precondition of the rule. In 3APL those preconditions are atoms or atoms with negation as failure that represent queries to the belief base. In our proposed approach, this guard will be a set of literals or extended literals (*i. e.*, a literal preceded by the symbol not).

Definition 11 *A goal planning rule (or PG-rule) R is an ordered triplet R=(κ, β, π), where: κ (header goal) is a conjunction of atoms representing the goal, π is a plan, and β=$\{G_1, \ldots, G_n, not\ C_1, \ldots, not\ C_m\}$ ($n \geq 0$ and $m \geq 0$) is a guard formed by a set of literals $\{G_1, \ldots, G_n\}$ representing preconditions for R and a set of extended Literals $\{not\ C_1, \ldots, not\ C_m\}$ representing restrictions for R. A PG-rule (κ, β, π) is also denoted $\kappa \leftarrow \beta \mid \{\pi\}$.*

The applicability of a PG-rule depends on the status of its guard β. The next definition indicates when β is considered warranted in our proposed approach.

Definition 12 *Let β = {p_1,..., p_k, not c_1,...,not c_s} and $\mathcal{P}_\sigma$ a belief base. β is a warranted set from $\mathcal{P}_\sigma$ (noted $\mathcal{P}_\sigma \mid\sim_w \beta$) iff $\forall p_i$, $i = 1..k$, $\mathcal{P}_\sigma \mid\sim_w p_i$ and $\forall$ c_j, $j = 1..s$, $\mathcal{P}_\sigma \not\mid\sim_w c_i$*

Consider an agent with a belief base $\mathcal{P}_\sigma$ and a goal base γ, a PG-rule R = (κ, β, π) will be applicable when κ is in γ and β is a warranted set from $\mathcal{P}_\sigma$. For instance, the agent of Fig. 1 has only one applicable PG-rule $transport(D) \leftarrow use\_bus(D) \mid \{bus\_plan(D)\}$. This is because $transport(office)$ is in γ and $use\_bus(office)$ is a warranted belief from $\mathcal{P}_\sigma$ (see Ex. 3). Also, observe that for example the first PG-rule is not applicable because $\sim snow$ is not a warranted belief. The operational semantics for PG-rules will be introduced in the next section. PR-Rules can be defined analogously to PG-Rules.

In 3APL, there are two types of actions that interact with the belief base: *mental actions* and *test actions*. Mental action rules are used to add or remove beliefs.

Definition 13 *A Mental Action rule (or MA-rule) is an ordered triplet Ma=(Pre, $Name$, Pos), where: $Name$ is an atom, Pre={p_1,..., p_n, not c_1,..., not c_m} ($n \geq 0$ and $m \geq 0$) is formed by a set of literals $\{p_1, \ldots, p_n\}$ representing preconditions and a set of extended literals $\{not\ c_1, \ldots, not\ c_m\}$ representing restrictions; Pos= {a_1,..., a_k, not d_1,..., not d_s} ($k \geq 0$ and $s \geq 0$) is formed by a set of non-contradictory literals $\{a_1, \ldots, a_n\}$ representing information to be added and a set of extended literals $\{not\ d_1, \ldots, not\ d_m\}$ representing information to be removed.*

As in 3APL, the set `Capabilities` will include all the MA-Rules of a 3APL-DeLP agent. In each MA-rule Ma=(Pre, $Name$, Pos) the atom $Name$ is used to apply Ma in a plan. The atom $Name$ can contain variables which will be treated as in-mode parameters.

Consider an agent with a belief base $\mathcal{P}_\sigma$, a MA-rule (Pre, $Name$, Pos) will be applicable when Pre is a warranted set from $\mathcal{P}_\sigma$ (Def.12). For instance, the agent of Fig. 1 has one applicable MA-rule

$\{have\_car\}\ car\_broken\ \{\sim have\_car\}$. This is because $have\_car$ is a warranted belief from $\mathcal{P}_\sigma$ (see Ex. 3). When a MA-rule is applied, its effects will modify the set Ψ_σ. Recall that Ψ_σ should be non-contradictory. In the next section we will describe the operational semantics of a MA-rule.

Since MA-rules can only add/remove literals, we introduce a new type of basic actions called extended mental actions that will be included in `Capabilities` and will be used to add and remove defeasible-rules.

Definition 14 *An Extended Mental Action rule (or EMA-rule) is a schematic ordered triplet EMa=(Pre, $Name$, Pos), where $Name$ and Pre are defined as in the MA-rules. Here, Pos= $\{R_1,\ldots, R_k, not\ R_{k+1},\ldots, not\ R_s\}$ ($k \geq 0$ and $s \geq k$) is formed by a set $\{R_1, \ldots, R_k\}$ representing defeasible-rules to be added, and a set $\{not\ R_{k+1}, \ldots, not\ R_s\}$ representing defeasible-rules to be removed, each R_i may be a d-rule or a rule-variable.*

EMA-Rules will be applicable when the set Pre is warranted from the belief base (Def.12). For instance, the agent of Fig. 1 has one EMA-rule, $\{received(A, inform, X)\}\ add\_preference\ \{X\}$, which is not applicable. This is because $received(A, inform, X)$ is not a warranted belief from $\mathcal{P}_\sigma$ (see Ex. 3). In order to add flexibility we allow to use rule-variables. Note that in the above rule X is a rule-variable. These variables will be dynamically instantiated with defeasible-rules when the EMA-rules are executed. In the next section we will give its semantics.

In order to allow the agent to change the comparison criterion dynamically we introduce a new type of basic actions called criterion change actions that will be included in `Capabilities`.

Definition 15 *An Criterion Change Action rule (or CCA-rule) is a schematic ordered triplet CCa=(Pre, $Name$, CC), where $Name$ and Pre are defined as in the MA-rules and CC can be a the name of a comparison criterion or a variable.*

CCA-Rules will be applicable when the set Pre is warranted from the belief base (Def.12). If CC is a variable, it will be dynamically instantiated with a comparison criterion when the CCa-rule is executed. In the next section we will give its semantics.

In 3APL, test actions are queries in the form of well formed formulas to the belief base. In our approach, test actions will be a set of literals or extended literals.

Definition 16 *A test action is a term $?(T)$ where T=$\{G_1, \ldots, G_n, not\ C_1, \ldots, not\ C_m\}$ is formed by a set of literals $\{G_1, \ldots, G_n\}$ representing preconditions and a set of extended literals $\{not\ C_1, \ldots, not\ C_m\}$ representing restrictions.*

Test actions are used to retrieve information or block the plan in which they are included. The effect of applying a test action rule will be given in the next section.

3APL-DeLP Operational Semantics

In previous sections we have described how to integrate DeLP into the agent belief base, how to obtain warranted information from the DeLP belief base, and how to modify the mental components in order to use that warranted information. In this section we will introduce the operational semantics of the 3APL-DeLP agents using the Transition System.

As in 3APL, to program a 3APL-DeLP agent means to specify its initial beliefs, the comparison criterion, goals, plans and capabilities, and to specify PG-rules and plan revision rules. The beliefs, goals, plans of individual agents and their environment can change during the execution of the agent, while the capabilities and the reasoning rules remain the same. Together with a substitution component, these changing components constitute a 3APL-DeLP agent configuration.

Definition 17 *A configuration of an individual 3APL-DeLP agent is a tuple $\langle \iota,\ \mathcal{P}_\sigma,\ \leq,\ \gamma,\ \Pi,\ \theta,\ \xi \rangle$ where ι is an agent identifier, $\mathcal{P}_\sigma$ is the belief base, $\leq$ is the comparison criterion, γ is the goal base, Π is the plan base, θ is a ground substitution that binds domain variables to domain terms, and ξ is the environment the agent interacts with, where ξ is a set of ground atoms. For any possible goal ϕ of γ it holds that ϕ is not a warrant belief of $\mathcal{P}_\sigma$.*

A transition is a transformation of one configuration into another and it corresponds to a single computation step. The transition rules can derive transitions transforming single-agent configurations. These derivation rules specify the semantics of the execution of plans and the application of reasoning rules. Next we will introduce derivation rules for the elements affected by our proposal, the rest of the derivation rules coincides with the ones presented for 3APL in (Dastani, van Riemsdijk, & Meyer 2005).

The first derivation rule specifies the execution of the plan base of a 3APL agent. The plan base of the agent is a set of plan-goal pairs. This set can be executed by executing one of the constituent plans. The execution of a plan can change the agent's configuration.

Definition 18 (plan base execution) *Let $\Pi = \{(\pi_1, k_1),\ \ldots,\ (\pi_i, k_i),\ \ldots,\ (\pi_n, k_n)\}$ and $\Pi' = \{(\pi_1, k_1),\ \ldots,\ (\pi'_i, k_i),\ \ldots,\ (\pi_n, k_n)\}$ be plan bases, θ, θ' be ground substitutions, and ξ, ξ' be environment specifications. Then, the derivation rule for the execution of a set of plans is specified in terms of the execution of individual plans as follows.*

$$\frac{\langle \iota,\mathcal{P}_\sigma,\leq,\gamma,(\pi_i,k_i),\theta,\xi\rangle \rightarrow \langle \iota,\mathcal{P}'_\sigma,\leq,\gamma',(\pi'_i,k_i),\theta',\xi'\rangle}{\langle \iota,\mathcal{P}_\sigma,\leq,\gamma,\Pi,\theta,\xi\rangle \rightarrow \langle \iota,\mathcal{P}'_\sigma,\leq,\gamma',\Pi',\theta',\xi'\rangle}$$

MA-rules can add or remove literals from the belief base. Their effects are given by a function $\mathcal{T}$ that should preserve consistency of the belief base. In this work we will define this function $\mathcal{T}$ next:

Definition 19 *Let Ma=(Pre, $Name$, Pos) be a MA-rule, where Pos= $\{a_1,\ldots, a_k, not\ d_1,\ldots, not\ d_s\}$, θ is a ground substitution, and the agent belief base $\mathcal{P}_\sigma$ = (Ψ_σ, Δ_σ, $Prog$). With the literals in Pos we define Del= $\{\ \overline{a_1}\ \ldots, \overline{a_k}, d_1\ \ldots, d_s\ \}$, and Add = $\{a_1,\ldots, a_k\}$. Then the mental action update function is: $\mathcal{T}(Ma, \theta, \mathcal{P}_\sigma) = ((\Psi_\sigma \setminus \{x_i\theta : x_i \in Del\}) \cup \{y_i\theta : y_i \in Add\}, \Delta_\sigma, Prog)$*

Definition 20 *Let Ma=(Pre, $Name$, Pos) be an applicable MA-rule, $\mathcal{T}$ be the mental action update function, and*

τ be a ground substitution, then the execution of a single MA-rule is specified as follows:

$$\frac{(\mathcal{P}_\sigma,\leq)\mid\!\sim_w Pre\theta\tau \;\&\; \mathcal{T}(Ma,\theta\tau,\mathcal{P}_\sigma)=\mathcal{P}'_\sigma \;\&\; \gamma\models k}{\langle\iota,\mathcal{P}_\sigma,\leq,\gamma,(Ma,k),\theta,\xi\rangle\rightarrow\langle\iota,\mathcal{P}'_\sigma,\leq,\gamma',(E,k),\theta,\xi\rangle}$$

where $\gamma' = \gamma \setminus \{\phi \in \gamma \mid (\mathcal{P}'_\sigma,\leq)\mid\!\sim_w \phi\}$

Note that the substitution θ is first applied to Pre, and then τ is the substitution returned by DeLP that should be used in order to bind the free variables of $Pre\theta$. Finally, $\theta\tau$ is used by the belief update function $\mathcal{T}$ to bind the corresponding free variables of the MA-rule postcondition. Also note that E is treated as an empty action, as it is in 3APL.

Example 6 *Consider the belief base and the capabilities of the agent of Fig. 1 (its warranted beliefs were described in Ex. 3). Suppose that the agent applies the MA-rule* car_broken *(note that it is applicable because* $have\_car$ *is warranted), then the fact* $have\_car$ *will be removed from the belief base and the fact* $\sim have\_car$ *will be introduced.*

In 3APL-DeLP MA-rules allow to add or remove literals from the belief base preserving the consistency of Ψ_σ, and argumentative reasoning is used in order to determine their applicability. Adding a new literal L to the belief base will allow to build new arguments, but it may also disable old arguments that are contradictory with $\Psi_\sigma \cup L$. Analogously, removing a literal may introduce or disable arguments.

EMA-rules can add or remove defeasible-rules from the belief base. Their effects are given by the function $\mathcal{U}$ defined next:

Definition 21 *Let* EMa*=(*Pre*,* $Name$*,* Pos*) be a EMA-rule, where* Pos*=* $\{R_1,\ldots, R_k, not\ R_{k+1},\ldots, not\ R_s\}$*. Let* θ *be a ground substitution,* λ *a set with an instantiation for each rule-variable, and* $\mathcal{P}_\sigma$ *=* $(\Psi_\sigma,\ \Delta_\sigma,\ Prog)$ *the agent belief base. We define* Del*=* $\{\ R_{k+1},\ldots,\ R_s\}$*, and* Add *=* $\{R_1,\ldots, R_k\}$*, where each* R_i *that is a rule-variable will be instantiated using* λ*. Then, the extended mental action update function is* $\mathcal{U}(EMa,\theta,\lambda,\mathcal{P}_\sigma) = (\Psi_\sigma,(\Delta_\sigma \setminus \{x_i\theta : x_i \in Del\}) \cup \{y_i\theta : y_i \in Add\},\ Prog)$*.*

Remark 1: If $R_i \in Del$ is a schematic rule, the function $\mathcal{U}$ will remove all the rules that match with R_i.
Remark 2: The rules that will instantiate the rule-variables of Del and Add are considered ground for the θ substitution.

Definition 22 *Let* EMa*=(*Pre*,* $Name$*,* Pos*) be a EMA-rule,* $\mathcal{U}$ *be the EMA-rule update function and* τ *be a ground substitution. Then, the execution of a EMA-rule is:*

$$\frac{(\mathcal{P}_\sigma,\leq)\mid\!\sim_w Pre\theta\lambda\tau \;\&\; \mathcal{U}(Ma,\theta\tau,\lambda,\mathcal{P}_\sigma)=\mathcal{P}'_\sigma \;\&\; \gamma\models k}{\langle\iota,\mathcal{P}_\sigma,\leq,\gamma,(EMa,k),\theta,\xi\rangle\rightarrow\langle\iota,\mathcal{P}'_\sigma,\leq,\gamma',(E,k),\theta,\xi\rangle}$$

where $\gamma' = \gamma \setminus \{\phi \in \gamma \mid (\mathcal{P}'_\sigma,\leq)\mid\!\sim_w \phi\}$

The set of instantiations λ is used to bind those free variables of $Pre\theta$ that are rule-variables in Pos. The substitution τ is used for the rest of the free variables in $Pre\theta\lambda$.

Example 7 *Consider now that the agent of Ex. 4 receives a message received(ag1, inform,* $(\sim use\_metro(D) \prec \sim downtown(D))$*) meaning that the metro stations outside downtown are becoming dangerous due to pick pockets. Thus, the agent will be able to apply the EMA-rule* $\{received(A, inform, P)\}$ $add\_preference\{P\}$ *because received(ag1, inform,* $(\sim use\_metro(D) \prec \sim downtown(D))$*) is warranted from the belief base. Note that the variable P will be instantiated with* $\sim use\_metro(D) \prec \sim downtown(D)$*. Then if the agent applies that EMA-rule, the d-rule* $\sim use\_metro(D) \prec \sim downtown(D)$ *will be added to the agent belief base.*

Using EMA-rules, new defeasible-rules can be added to the belief base, hence new arguments may be built. Recall that warranted literals are determined by the argumentation process. With new arguments, new warranted literals may arise. Nevertheless, the new arguments may represent defeaters for other arguments and therefore some literals may be not warranted. Thus, after applying the EMA-rules the set of warranted beliefs may change. For instance in the situation described in Ex.7, if the EMA-rule is applied, use_metro will not be a warranted belief anymore. Observe that defeasible-rules are simply added to the belief base. This is because no revision is needed, conflicts among defeasible-rules are solved by the DeLP argumentation mechanism.

CCA-rules can change the argumentation comparison criterion. Their semantics are defined as follows:

Definition 23 *Let* CCa*=(*Pre*,* $Name$*,* CC*) be a CCA-rule and* τ *be a ground substitution. Then, the execution of a CCA-rule is:*

$$\frac{(\mathcal{P}_\sigma,\leq)\mid\!\sim_w Pre\theta\tau \;\&\; \gamma\models k}{\langle\iota,\mathcal{P}_\sigma,\leq,\gamma,(CCa,k),\theta,\xi\rangle\rightarrow\langle\iota,\mathcal{P}_\sigma,CC\theta\tau,\gamma',(E,k),\theta,\xi\rangle}$$

where $\gamma' = \gamma \setminus \{\phi \in \gamma \mid (\mathcal{P}_\sigma, CCa\theta\tau)\mid\!\sim_w \phi\}$

Note that the Comparison Criterion is not changed when the warrants are calculated. This means that obtaining warrant for a literal L is an atomic operation. The CCA-Rules allow the agent to change the argument comparison criterion using an action during the execution of a plan. Thus, for instance, the agent will be able to change the comparison criterion to one that adapts to the environment conditions in which is currently involved, build plans to change the way in which the agent weights its beliefs or receive a comparison criterion from other agent and use it.

Next, we specify the derivation rule for the execution of a test action. A test action can bind the free variables that occur in the test formula.

Definition 24 *Let* β *=*$\{p_1,\ \ldots,\ p_n, not\ c_1,\ldots, not\ c_m\}$*, and* τ *be a ground substitution, then the test action execution is:*

$$\frac{(\mathcal{P}_\sigma,\leq)\mid\!\sim_w \beta\ \theta\tau \;\&\; \gamma\models k}{\langle\iota,\mathcal{P}_\sigma,\leq,\gamma,(\beta?,k),\theta,\xi\rangle\rightarrow\langle\iota,\mathcal{P}_\sigma,\leq,\gamma,(E,k),\theta\tau,\xi\rangle}$$

Composite plans containing conditions over the beliefs base are also affected by our proposal. However, as they are composed by test actions, they can be analogously defined using the previous definitions (see (Dastani *et al.* 2003)).

Next, we will define the transition rule for the PG-rules. A PG-rule $\kappa \leftarrow \beta \mid \{\pi\}$ specifies that the goal κ can be achieved by the plan π if the guard β is a warranted set from the belief base.

Definition 25 *Let* $\kappa \leftarrow \beta \mid \{\pi\}$ *be a PG-rule, where* $\beta = \{p_1, \ldots, p_n, not\ c_1, \ldots, not\ c_m\}$*,and* τ_1 *and* τ_2 *be a ground substitutions, then the PG-rule application is:*

$$\frac{\gamma \models k\tau_1 \ \&\ (\mathcal{P}_\sigma, \leq) \mid\sim_w \beta\, \tau_1\tau_2}{\langle \iota, \mathcal{P}_\sigma, \leq, \gamma, \Pi, \theta, \xi\rangle \rightarrow \langle \iota, \gamma, \mathcal{P}_\gamma, \leq, \{(\pi\tau_1\tau_2, k\tau_1)\} \cup \Pi, \theta, \xi\rangle}$$

The 3APL-DeLP PG-Rule guards are able to consult for negative and positive literals, combined with negation as failure. Also note that the deliberative cycle step in which the interpreter searches for applicable PG-Rule argumentative reasoning will be used thus making the agent reasoning capabilities more sophisticated.

Properties of 3APL-DeLP

The aim of this section is to study the properties that arise if integration between 3APL and DeLP proposed in this work.

In this proposal 3APL-DeLP agents are able to use strong negation, which increases the expressive power of the belief representation languages. However, using strong negation may lead to inconsistencies, which is an undesirable effect for an agent. Next, we will show two propositions that shows that the system works well with strong negation (*i. e.*the system does not reach inconsistencies).

The following property shows that in the 3APL-DeLP framework the strict knowledge always remains non-contradictory.

Proposition 1 *Given an 3APL-DeLP agent configuration* $\langle \iota, \mathcal{P}_\sigma, \leq, \gamma, \Pi, \theta, \xi\rangle$ *with* $\mathcal{P}_\sigma = (\Delta_\sigma, \Psi_\sigma, Prog)$ *were* Ψ_σ *is non-contradictory, after any of the possible transitions,* Ψ_σ *remains non-contradictory.*

Proof: Let $\langle \iota, \mathcal{P}_\sigma, \leq, \gamma, \Pi, \theta, \xi\rangle$ be the agent configuration with $\mathcal{P}_\sigma = (\Delta_\sigma, \Psi_\sigma, Prog)$ were Ψ_σ is non-contradictory. If the transition does not change the belief base, Ψ_σ remains non-contradictory. If the transition does change the belief base, it may be the transition of a EMA-Rule or a MA-Rule. If its a EMA-Rule then Δ_σ is changed, however it clear by definition 2 this does not introduce contradictions in the strict part of the program. If the transition is MA-Rule α then Ψ_σ is changed then:

- if α removes a literal L from Ψ_σ, then clearly Ψ_σ remains non-contradictory
- if α adds a literal L to Ψ_σ and $\overline{L} \notin \Psi_\sigma$, then clearly Ψ_σ remains non-contradictory
- if α adds a literal L to Ψ_σ and $\overline{L}\ in\ \Psi_\sigma$, then Ψ_σ remains non-contradictory because by definition 20 $\overline{L}$ will be removed from Ψ_σ

□

Using strong negation contradictory information can be inferred. In order to address this issue the argumentation mechanism will be used to decide which piece information prevail. Next, we will show that the set of warranted beliefs inferred by a 3APL-DeLP agent is a non-contradictory set (ie, the agent can not warrant a literal and its complement simultaneously).

Proposition 2 *Given a belief base* $\mathcal{P}_\sigma = (\Delta_\sigma, \Psi_\sigma, Prog)$ *were* Ψ_σ *is consistent, the set of warranted belief* W_b *obtained from* $\mathcal{P}_\sigma$ *is non-contradictory.*

Proof Sketch: Let W_b be the set of warranted beliefs from $\mathcal{P}_\sigma$ and $h_1 \in W_b$ thus, there exists an argument $\langle \mathcal{A}_1, h_1\rangle$ for h_1 which is undefeated in a dialectial process. Suppose $h_2 \in W_b$ and h_1 and h_2 are contradictory ($h_1 = \overline{h_2}$). As h_2 is warranted there exists an argument $\langle \mathcal{A}_2, h_2\rangle$ for h_2 which is undefeated in the argumentation process As h_1 and $h_2$are contradictory $\langle \mathcal{A}_1, h_1\rangle$ is a counterargument for $\langle \mathcal{A}_2, h_2\rangle$ and vice versa. Depending on the comparison criterion used, $\langle \mathcal{A}_2, h_2\rangle$ can be a proper defeat of $\langle \mathcal{A}_1, h_1\rangle$(or viceversa) or $\langle \mathcal{A}_2, h_2\rangle$ and $\langle \mathcal{A}_1, h_1\rangle$are blocking defeaters. Since $\langle \mathcal{A}_2, h_2\rangle$ is undefeated and a defeats (bloking or proper) $\langle \mathcal{A}_1, h_1\rangle$, h_1 is not warranted from $\mathcal{P}_\sigma$, that contradicts our hypothesis.□

The previous propositions shows that although the knowledge language allows strong negation, if an agent starts with a consistent Ψ_σ, regardless the action the agent executes or changes in the environment, the set of inferences of the agent will be non-contradictory. Thus, with these two properties we have shown that 3APL-DeLP agents can model their knowledge using strong negation and be safe of inconsistencies.

Since the belief representation language allows strong negation, contradictory information can be inferred. If contradictions occur, a decision criterion is needed to determine which information prevails. Previous approaches that increased the representational capabilities of 3APL belief language used a fixed decision criterion. Here, we have proposed a mechanism were the decision criterion is based on a programmable argument comparison criterion and that can be changed dynamically without changing the belief program. Since this criterion is programable, it adds an interesting degree of flexibility for the agent programmer. In this work for instance, we have shown two criterions. The first one, generalized specificity, is a general criterion and depends of the general structure of the arguments. The second one presented in example 5 is a domain dependant criterion and depend on particular domain elements that are part of the arguments. Other general comparison criterions could involve for instance, the quantity of literals, rule time-stamps, or rule position in the program. Other particular comparison criterions could involve for instance, preferences relations over the domain literals (Ferretti *et al.* 2008) or preferences relation over the domain rules using probability labels. As we shown in example 5 criterions can be combined to add more flexibility. The CCA-Rules allow the agent to change the argument comparison criterion using an action during the execution of a plan. Using these rules the agent will able, for instance, to change the comparison criterion to one that adapts to the environment conditions in which is currently involved, build plans to change the way in which the agent weights its beliefs or receive a comparison criterion from other agent and use it.

Conclusions and Related Work

In this work we have shown how to integrate a defeasible argumentation programming language into 3APL. The proposed approach allows to represent beliefs with a defeasible logic program. Thus, tentative information is represented

using defeasible-rules, negative information is represented by strong negation. In addition to this, a defeasible argumentation process is used to warrant the agent derived beliefs and a programmable criteria used by this process to decide between contradictory conclusions. DeLP features allow to model incomplete and potentially contradictory information thus extending representational capabilities of these agents. The inference mechanism of DeLP allows the agent to decide among contradictory conclusions, and allows to add or remove information in a dynamic way, without the need of changing every rule, which adds scalability and flexibility. Since the proposed approach affects the 3APL agent reasoning mechanism, new definitions for the agent mental components are given.

Thus, this approach, as a agent programming language, contains all the advantages of 3APL plus strong negation, argumentative reasoning, a programmable comparison criterion. Our aim with this proposal is to provide a highly declarative tool for agent programming, were the programmer can focus on the rules that represent knowledge in a declarative way and not in the explicit interaction among these rules. Therefore, in order to add a rule the programmer does not need to be aware of the whole knowledge program, he just add it and the argumentative mechanism solve the inferences.

The idea of using defeasible argumentation in the reasoning process cognitive agents is not new, and there exist previous approaches that relate cognitive agents to argumentation frameworks (Rahwan & Amgoud 2006; Rotstein, Garcia, & Simari 2007). However, none of these approaches propose an agent programming language. In (Rotstein, Garcia, & Simari 2007) a BDI agent architecture based on a DeLP was presented. As an architecture, several decision choices must be taken to allow the complete specification of agents. For example, they do not provide a model to represent agent actions or plans to solve desires. In contrast with their approach, we allow to use PROLOG rules and DeLP rules in the knowledge representation language. Also, we have defined a mechanism to update the agent belief base that combines 3APL capability rules with DeLP. Finally we give formal operational semantics in order to formally define the programming language constructs behavior.

In (Nigam & Leite 2006), a modification of 3APL to use Dynamic Logic Programming is introduced. There, they mention some limitations of 3APL regarding to the belief and goal representation language. Like us, they propose to use a more sophisticated belief representation language. However DLP and DeLP are different formalisms. DLP is based in Answer-Set programming and its aim is the update of the knowledge. DeLP is based on defeasible argumentation and its aim is to build arguments and reason with them. Thus our aim in this work is to provide 3APL with argumentative reasoning. In addition to that, in DLP the decision criterion between contradictory information is fixed, newer information is preferred. In DeLP a modular argument comparison criterion to decide between arguments that support contradictory conclusions is used. This comparison criterion is not fixed, so the agent programmer can use the best criterion depending on the context or domain application. For instance, the comparison criterion can be programmed to prefer those arguments that were built using newer rules, similar to DLP, or to prefer those arguments built from rules that have more literals. Thus with DeLP we provide a wider range of options to handle contradictory conclusions.

As future work, we plan to integrate the defeasible argumentation formalism into other components of a 3APL agent. For example, the shared environment, which is a set of atoms, can be extended to a defeasible logic program.

References

Bench-Capon, T. J. M., and Dunne, P. E. 2007. Argumentation in artificial intelligence. *Artif. Intell.* 171(10-15):619–641.

Bratman, M.; Israel, D.; and Pollack, M. Plans and resource-bounded practical reasoning. In *Philosophy and AI: Essays at the Interface*.

Dastani, M.; van Riemsdijk, B.; Dignum, F.; and Meyer, J. 2003. A programming language for cognitive agents: Goal-directed 3apl. In *First Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS03), Melbourne, Australia*.

Dastani, M.; Governatori, G.; Rotolo, A.; and van der Torre, L. W. N. 2005. Programming cognitive agents in defeasible logic. In *LPAR*, 621–636.

Dastani, M.; van Riemsdijk, M. B.; and Meyer, J.-J. C. 2005. Programming multi-agent systems in 3apl. In *Multi-Agent Programming*. 39–67.

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.

Ferretti, E.; Errecalde, M.; García, A.; and Simari, G. 2008. Decision rules and arguments in defeasible decision making. In *Proceedings of the 2nd International Conference on Computational Models of Arguments (COMMA)*, 171–182.

Fisher, M.; Bordini, R.; Hirsch, B.; and Torroni, P. 2007. Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence* 23(1):61–91.

Garcia, A., and Simari, G. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1-2):95–138.

Hindriks, K.; de Boer, F.; van der Hoek, W.; and Meyer, J.-J. C. 1999. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems* 2(4):357–401.

Nigam, V., and Leite, J. 2006. Adding knowledge updates to 3apl. In *PROMAS*, 165–181.

Rahwan, I., and Amgoud, L. 2006. An argumentation based approach for practical reasoning. In *AAMAS*.

Rotstein, N. D.; Garcia, A. J.; and Simari, G. R. 2007. Reasoning from desires to intentions: A dialectical framework. In *22nd. AAAI Conference on Artificial Intelligence*.

Stolzenburg, F.; Garcia, A. J.; Chesnevar, C. I.; and Simari, G. R. 2003. Computing generalized specificity. *Journal of Applied Non-Classical Logics* 13(1):87–.

Using Collaborations for Distributed Argumentation with Defeasible Logic Programming

Matthias Thimm
Faculty of Computer Science
Technische Universität Dortmund
Germany

Alejandro J. García
Department of Computer Science and Engineering
Universidad Nacional del Sur, Bahía Blanca
Argentina

Gabriele Kern-Isberner
Faculty of Computer Science
Technische Universität Dortmund
Germany

Guillermo R. Simari
Department of Computer Science and Engineering
Universidad Nacional del Sur, Bahía Blanca
Argentina

Abstract

In this paper, we extend previous work on distributed argumentation using Defeasible Logic Programming. There, several agents form a multi agent setting, in which they are able to generate arguments for a given query and counterarguments to the arguments of other agents. The framework is monitored by a moderator, which coordinates the argumentation process and can be seen as a judge overlooking the defender and accuser in a legal case. We extend this framework by allowing the agents to form alliances. We introduce a notion of cooperation for agents called *collaborations*, which allow the agents not only to argue with one another, but to share their beliefs in order to jointly generate new arguments. We give a declarative definition as well as an algorithmic characterization of the argument generation process and relate our framework with general Defeasible Logic Programming.

Introduction

Defeasible argumentation (Prakken & Vreeswijk 2002) deals with argumentative reasoning using uncertain knowledge. An instantiation of defeasible argumentation is Defeasible Logic Programming (DeLP) by García and Simari (García & Simari 2004) and is an approach for logical argumentative reasoning (Rahwan & Amgoud 2006; Besnard & Hunter 2000) based on defeasible logic. In DeLP the belief in literals is supported by arguments and in order to handle conflicting information a warrant procedure decides which information has the strongest grounds to believe in.

There are many approaches to realize multi agent argumentation and especially negotiation (Kraus 1997; Booth 2002) in multi agent systems. Whereas in (Amgoud, Dimopolous, & Moraitis 2007; Parsons, Sierra, & Jennings 1998) and especially in (Bench-Capon 2003), the focus lies on using argumentation for persuasion, here we use argumentation to reach a common conclusion of a group of agents. Considering a jury court it is reasonable to assume that there are jurors who are less competent in jurisdiction than others. However it is the main goal to reach an agreement regarding the given case rather than unifying the jurors beliefs.

In this paper, a distributed argumentation framework for cooperative agents is introduced in which agents may have independent or overlapping belief bases. Here, following (Thimm 2008; Thimm & Kern-Isberner 2008a), Defeasible Logic Programming is used for knowledge representation. Hence, agents belief bases will be sets of defeasible rules (García & Simari 2004) and agents may build argument using their local rules. Similar to (Móra, Alferes, & Schroeder 1998; de Almeida & Alferes 2006), we will define a notion of *collaboration* and a mechanisms that allow agents to cooperate for building arguments will be introduced.

In many different scenarios the cooperation of agents in a multi agent setting is desirable. Suppose that in a legal dispute a team of lawyers have to work together, acting as one accuser or defender. Or imagine a dispute between political parties, where each member tries to defend their party's interests. The simplest solution to these kinds of scenarios is to represent each whole team or party as one single agent, thus merging the beliefs of the members into one knowledge base. But from a knowledge representational point of view it is more realistic to represent each member of such a team as an individual agent and let these agents collaborate with each other. Another drawback of the first approach is a computational one. If the knowledge of many members of a team is joined, the computation of arguments can be expensive, as the whole knowledge base has to be searched. If a team is made up of many agents, each an expert in his field, the construction and evaluation of arguments can be divided upon them and only the agents, that can contribute, do so.

The framework proposed in (Thimm & Kern-Isberner 2008a) consists of several agents and a central moderator, which coordinates the argumentation process undertaken by the agents. The moderator accepts a query, consisting of a

single literal, and asks the agents to argue about the warrant status of it. That framework was motivated for modeling situations where participating agents have opposite views of the given query (e. g. a legal dispute, where agents take the roles of accuser and defender). Therefore, each agent build its own arguments using its local belief and it may attack or defend arguments of other agents. In this paper, we extend that framework by considering groups of agents who may collaborate in order to build better arguments using beliefs of other agents. A collaboration is basically a set of agents that form an alliance for argument construction. With the use of collaborations, we are able to derive more arguments than in the case with no collaborations. The key idea of computing collaborated arguments is similar to (Móra, Alferes, & Schroeder 1998) but uses another concept of a *partial argument*. As will be described below, a partial argument is some kind of an intermediate result when constructing an argument in a distributed manner. Partial arguments help collecting the rules that are necessary to derive a conclusion from the local belief bases.

The paper is organized as follows. In the next section, a brief introduction to Defeasible Logic Programming and the distributed framework of (Thimm & Kern-Isberner 2008a) is presented. We continue by introducing collaborations into the multi agent setting, that allow the agents to jointly build arguments. We proof soundness and completeness of the algorithmic representation of collaborated argument generation, followed by a comparison of our approach with Defeasible Logic Programming and other related work. Finally, we conclude we a summary and an outlook to further work.

Distributed Argumentation using DeLP

We give a brief introduction in the distributed argumentation framework ArgMAS (*Argumentation-based multi agent system*) proposed in (Thimm 2008; Thimm & Kern-Isberner 2008a; 2008b) adapted to our needs in this paper. The framework is based upon DeLP (Defeasible Logic Programming) (García & Simari 2004) and consists of several agents and a central moderator, which coordinates the argumentation process undertaken by the agents. An overview of such a system is depicted in figure 1. The moderator accepts a query, consisting of a single literal, and asks the agents to argue about the warrant status of it, i. e., whether the literal or its negation can be supported by an ultimately undefeated argument. Agents use the global belief base of the system, which contains strict knowledge, and their own local belief bases consisting of defeasible knowledge to generate arguments. Eventually the system returns an answer to the questioner that describes the final status of the literal based on the agents' individual beliefs.

We start our description of this framework by presenting the basic argumentative formalisms of DeLP(García & Simari 2004).

Defeasible Logic Programming

The basic elements of DeLP are facts and rules. Let $\mathcal{L}$ denote a set of ground literals, where a literal h is a ground atom A or a negated ground atom $\sim A$, where the symbol

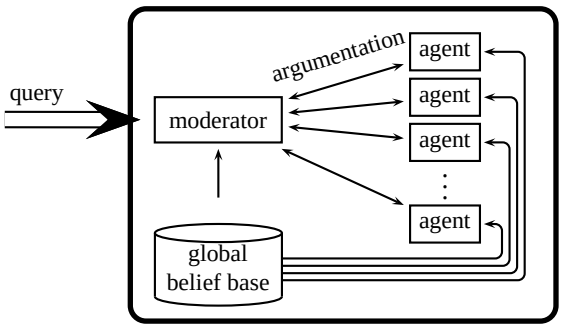


Figure 1: An argumentation-based multi agent system (ArgMAS)

$\sim$ represents the strong negation. Overlining will be used to denote the complement of a literal with respect to strong negation, i. e., it is $\overline{p} = \sim p$ and $\overline{\sim p} = p$ for a ground atom p. A literal $h \in \mathcal{L}$ is also called a *fact*.

The set of rules is divided into strict rules, i. e., rules encoding strict consequences, and defeasible rules which derive uncertain or defeasible conclusions. A *strict rule* is an ordered pair $h \leftarrow B$, where $h \in \mathcal{L}$ and $B \subseteq \mathcal{L}$. A *defeasible rule* is an ordered pair $h \prec B$, where $h \in \mathcal{L}$ and $B \subseteq \mathcal{L}$. A defeasible rule is used to describe tentative knowledge as in "birds fly". We use the functions $body/1$ and $head/1$ to refer to the head resp. body of a defeasible or strict rule. Strict and defeasible rules are ground. However, following the usual convention (Lifschitz 1996), some examples will use "schematic rules" with variables (denoted with an initial uppercase letter). Let DEF_X resp. STR_X be the set of all defeasible resp. strict rules, that can be constructed with literals from $X \subseteq \mathcal{L}$. We will omit the subscripts when referring to the whole set of literals $\mathcal{L}$, e. g. we write DEF for $\mathsf{DEF}_{\mathcal{L}}$.

Using facts, strict and defeasible rules, one is able to derive additional beliefs as in other rule-based systems. Let $X \subseteq \mathcal{L} \cup \mathsf{STR} \cup \mathsf{DEF}$ be a set of facts, strict rules, defeasible rules, and let furthermore $h \in \mathcal{L}$. A *(defeasible) derivation* of h from X, denoted $X \mathrel{|\!\sim} h$, consists of a finite sequence $h_1, \ldots, h_n = h$ of literals ($h_i \in \mathcal{L}$) such that h_i is a fact ($h_i \in X$) or there is a strict or defeasible rule in X with head h_i and body $b_1, \ldots, b_k$, where every b_l ($1 \leq l \leq k$) is an element h_j with $j < i$. If the derivation of a literal h only uses strict rules, the derivation is called a *strict* derivation. A set X is *contradictory*, denoted $X \mathrel{|\!\sim} \bot$, iff there exist defeasible derivations of two complementary literals from X. In difference to DeLP, the framework of ArgMAS divides the strict and defeasible knowledge into a global belief base and several local belief bases which constitute the individual beliefs of each agent.

Definition 1 (Belief bases)**.** A *global belief base* $\Pi \subseteq \mathcal{L} \cup \mathsf{STR}$ is a non-contradictory set of strict rules and facts. A set of defeasible rules $\Delta \subseteq \mathsf{DEF}$ is called a *local belief base*.

Given a set of agents $\mathfrak{A} = \{A_1, \ldots, A_n\}$ every agent A_i maintains a local belief base Δ_i ($1 \leq i \leq n$) which represents his own belief.

Example 1 ((García & Simari 2004), example 2.1)**.** Let a global belief base Π and a local belief base Δ be given by

$$\Pi = \left\{ \begin{array}{l} chicken(tina) \\ scared(tina) \\ penguin(tweety) \\ bird(X) \leftarrow chicken(X) \\ bird(X) \leftarrow penguin(X) \\ \sim flies(X) \leftarrow penguin(X) \end{array} \right\},$$

$$\Delta = \left\{ \begin{array}{l} flies(X) \prec bird(X) \\ \sim flies(X) \prec chicken(X) \\ flies(X) \prec chicken(X), scared(X) \\ nests\_in\_trees(X) \prec flies(X) \end{array} \right\}.$$

The global belief base Π contains the facts, that Tina is a scared chicken and that Tweety is penguin. The strict rules state that all chickens and all penguins are birds, and penguins cannot fly. The defeasible rules of the local belief base Δ express that birds normally fly, chickens normally do not fly (except when they are scared) and something that flies normally nests in trees.

As facts and strict rules describe strict knowledge, it is reasonable to assume Π to be non-contradictory, i. e., there are no derivations of complementary literals from Π only. But when considering several (or just one) local belief bases $\Delta_1, \ldots, \Delta_n$ of other agents, which may have different beliefs, then $\Pi \cup \Delta_1 \cup \ldots \cup \Delta_n$ can be contradictory.

Definition 2 (Argument, Subargument)**.** Let $h \in \mathcal{L}$ be a literal and let Π resp. Δ be a global resp. local belief base. $\langle \mathcal{A}, h \rangle$ is an *argument* for h, iff

- $\mathcal{A} \subseteq \Delta$,
- there exists a defeasible derivation of h from $\Pi \cup \mathcal{A}$,
- the set $\Pi \cup \mathcal{A}$ is non-contradictory, and
- $\mathcal{A}$ is minimal with respect to set inclusion.

The literal h will be called *conclusion* and the set $\mathcal{A}$ will be called *support* of the argument $\langle \mathcal{A}, h \rangle$. An argument $\langle \mathcal{B}, q \rangle$ is a *subargument* of an argument $\langle \mathcal{A}, h \rangle$, iff $\mathcal{B} \subseteq \mathcal{A}$. Let $\mathsf{ARG}_{\Pi,\Delta}$ be the set of all arguments that can be built from Π and Δ.

Two literals h and h_1 *disagree* regarding a global belief base Π, iff the set $\Pi \cup \{h, h_1\}$ is contradictory. Two complementary literals p und $\sim p$ disagree trivially, because for every Π the set $\Pi \cup \{p, \sim p\}$ is contradictory. But two literals which are not contradictory, can disagree as well. For $\Pi = \{(\sim h \leftarrow b), (h \leftarrow a)\}$ the literals a and b disagree, because $\Pi \cup \{a, b\}$ is contradictory.

We call an argument $\langle \mathcal{A}_1, h_1 \rangle$ a *counterargument* to an argument $\langle \mathcal{A}_2, h_2 \rangle$ at a literal h, iff there is a subargument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that h and h_1 disagree.

In order to deal with counterarguments to other arguments, a central aspect of defeasible argumentation becomes a formal comparison criterion among arguments. A possible preference relation among arguments is *Generalized Specificity* (Stolzenburg *et al.* 2003). According to this criterion an argument is preferred to another argument, iff the former one is more *specific* than the latter, i. e., (informally) iff the former one uses more facts or less rules. For example, $\langle \{c \prec a, b\}, c \rangle$ is more specific than $\langle \{\sim c \prec a\}, \sim c \rangle$.

For a formal definition and and desirable properties of preference criterions in general see (Stolzenburg *et al.* 2003; García & Simari 2004). For the rest of this paper we use $\succ$ to denote an arbitrary but fixed preference criterion among arguments. The preference criterion is needed to decide whether an argument defeats another or not, as disagreement does not imply preference.

Definition 3 (Defeater)**.** An argument $\langle \mathcal{A}_1, h_1 \rangle$ is a *defeater* of an argument $\langle \mathcal{A}_2, h_2 \rangle$, iff there is a subargument $\langle \mathcal{A}, h \rangle$ of $\langle \mathcal{A}_2, h_2 \rangle$ such that $\langle \mathcal{A}_1, h_1 \rangle$ is a counterargument of $\langle \mathcal{A}_2, h_2 \rangle$ at literal h and either $\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$ (*proper defeat*) or $\langle \mathcal{A}_1, h_1 \rangle \not\succ \langle \mathcal{A}, h \rangle$ and $\langle \mathcal{A}, h \rangle \not\succ \langle \mathcal{A}_1, h_1 \rangle$ (*blocking defeat*).

When considering sequences of arguments, the definition of defeat is not sufficient to describe a conclusive argumentation line. Defeat only takes an argument and its counterargument into consideration, but disregards preceeding arguments. But we expect also properties like *non-circularity* or *concordance* from an argumentation sequence. See (García & Simari 2004) for a more detailed description of acceptable argumentation lines.

Definition 4 (Acceptable Argumentation Line)**.** Let Π be a global belief base. Let $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_m, h_m \rangle]$ be a sequence of some arguments. Λ is called *acceptable argumentation line*, iff

1. Λ is a finite sequence,
2. every argument $\langle \mathcal{A}_i, h_i \rangle$ with $i > 1$ is a defeater of its predecessor $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ and if $\langle \mathcal{A}_i, h_i \rangle$ is a blocking defeater of $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ and $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ exists, then $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ is a proper defeater of $\langle \mathcal{A}_i, h_i \rangle$,
3. $\Pi \cup \mathcal{A}_1 \cup \mathcal{A}_3 \cup \ldots$ is non-contradictory (*concordance of supporting arguments*),
4. $\Pi \cup \mathcal{A}_2 \cup \mathcal{A}_4 \cup \ldots$ is non-contradictory (*concordance of interfering arguments*), and
5. no argument $\langle \mathcal{A}_k, h_k \rangle$ is a subargument of an argument $\langle \mathcal{A}_i, h_i \rangle$ with $i < k$.

Let SEQ denote the set of all sequences of arguments that can be built using rules from DEF, STR and facts from $\mathcal{L}$.

Let $+$ denote the concatenation of argumentation lines and arguments, e. g. $[\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle] + \langle \mathcal{B}, h \rangle$ stands for $[\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}, h \rangle]$.

In DeLP a literal h is *warranted*, if there is an argument $\langle \mathcal{A}, h \rangle$ which is non-defeated in the end. To decide whether $\langle \mathcal{A}, h \rangle$ is defeated or not, every acceptable argumentation line starting with $\langle \mathcal{A}, h \rangle$ has to be considered.

Definition 5 (Dialectical Tree)**.** Let Π be a global belief base and $\Delta_1, \ldots, \Delta_n$ be local belief bases. Let $\langle \mathcal{A}_0, h_0 \rangle$ be an argument. A *dialectical tree* for $\langle \mathcal{A}_0, h_0 \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, is defined as follows.

1. The root of $\mathcal{T}$ is $\langle \mathcal{A}_0, h_0 \rangle$.
2. Let $\langle \mathcal{A}_n, h_n \rangle$ be a node in $\mathcal{T}$ and let $\Lambda = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ be the sequence of nodes from the root to $\langle \mathcal{A}_n, h_n \rangle$. Let $\langle \mathcal{B}_1, q_1 \rangle, \ldots, \langle \mathcal{B}_k, q_k \rangle$ be the defeaters of $\langle \mathcal{A}_n, h_n \rangle$. For every defeater $\langle \mathcal{B}_i, q_i \rangle$ with $1 \leq i \leq k$ such that the argumentation line $\Lambda' =$

$[\langle \mathcal{A}_0, h_0\rangle, \ldots, \langle \mathcal{A}_n, h_n\rangle, \langle \mathcal{B}_i, q_i\rangle]$ is acceptable, the node $\langle \mathcal{A}_n, h_n\rangle$ has a child $\langle \mathcal{B}_i, q_i\rangle$. If there is no such $\langle \mathcal{B}_i, q_i\rangle$, the node $\langle \mathcal{A}_n, h_n\rangle$ is a leaf.

Let DIA denote the set of all dialectical trees with arguments that can be built using rules from DEF, STR and facts from $\mathcal{L}$.

In order to decide whether the argument at the root of a given dialectical tree is defeated or not, it is necessary to perform a *bottom-up*-analysis of the tree. Every leaf of the tree is marked "undefeated" and every inner node is marked "defeated", if it has at least one child node marked "undefeated". Otherwise it is marked "undefeated". Let $\mathcal{T}^*_{\langle \mathcal{A},h\rangle}$ denote the marked dialectical tree of $\mathcal{T}_{\langle \mathcal{A},h\rangle}$.

We call a literal h *warranted*, iff there is an argument $\langle \mathcal{A}, h\rangle$ for h such that the root of the marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A},h\rangle}$ is marked "undefeated". Then $\langle \mathcal{A}, h\rangle$ is a *warrant* for h. Observe that, if a literal h is a fact or has a strict derivation from the global belief base Π alone, then h is also warranted as there are no counterarguments for $\langle \emptyset, h\rangle$.

Formal Description of the Distributed Framework

We now describe the components of the distributed framework, namely the moderator and the agents, using a functional description of their intended behaviour. As the framework of ArgMAS is flexible, many different definitions of the functions to be presented can be thought of. But we restrain them on the notions of DeLP as described above, so we use the subscript "D" to denote the DeLP specific implementation.

When the moderator receives arguments from the agents, he builds up several dialectical trees and finally he has to evaluate them using the bottom-up evaluation method described above.

Definition 6 (Analysis function χ_{D})**.** The *analysis function* χ_{D} is a function $\chi_{\mathsf{D}} : \mathsf{DIA} \rightarrow \{0, 1\}$ such that for every dialectical tree $\upsilon \in \mathsf{DIA}$ it holds $\chi_{\mathsf{D}}(\upsilon) = 1$ iff the root argument of υ is undefeated.

Furthermore the evaluation of dialectical trees makes only sense, if the tree was built up according to the definition of an acceptable argumentation line. Hence, the moderator and the agents as well, have to check whether new arguments are valid in the current argumentation line.

Definition 7 (Acceptance function $\eta_{\mathsf{D},\succ}$)**.** For a given preference relation $\succ$ among arguments, the *acceptance function* $\eta_{\mathsf{D},\succ}$ is a function $\eta_{\mathsf{D},\succ} : \mathsf{SEQ} \rightarrow \{0, 1\}$ such that for every argument sequence $\Lambda \in \mathsf{SEQ}$ it holds $\eta_{\mathsf{D},\succ}(\Lambda) = 1$ iff Λ is acceptable according to Definition 4.

It is possible to assume different acceptance functions for different agents according to different definitions of an acceptable argumentation line (Thimm & Kern-Isberner 2008b). But in our multi agent system, we assume $\eta_{\mathsf{D},\succ}$ to be fixed and the same for the moderator and all agents by convention.

At the end of the argumentation process for a query h, the agents have produced a set of dialectical trees with root arguments for h or $\overline{h}$, respectively. As we have to distinguish several different cases, the moderator has to decide, whether the query h is warranted, the negation of h is warranted, or none of them are warranted in the framework. Let $\mathfrak{P}(S)$ denote the power set of a set S.

Definition 8 (Decision function μ_{D})**.** The *decision function* μ_{D} is a function $\mu_{\mathsf{D}} : \mathfrak{P}(\mathsf{DIA}) \rightarrow \{\text{YES,NO,UNDECIDED,UNKNOWN}\}$. Let $Q_{\dot{p}} \subseteq \mathsf{DIA}$ such that all root arguments of dialectical trees in $Q_{\dot{p}}$ are arguments for p or for $\overline{p}$, then μ_{D} is defined as

1. $\mu_{\mathsf{D}}(Q_{\dot{p}}) = \text{YES}$, if there is a dialectical tree $\upsilon \in Q_{\dot{p}}$ s. t. the root of υ is an argument for p and $\chi_{\mathsf{D}}(\upsilon) = 1$.
2. $\mu_{\mathsf{D}}(Q_{\dot{p}}) = \text{NO}$, if there is a dialectical tree $\upsilon \in Q_{\dot{p}}$ s. t. the root of υ is an argument for $\overline{p}$ and $\chi_{\mathsf{D}}(\upsilon) = 1$.
3. $\mu_{\mathsf{D}}(Q_{\dot{p}}) = \text{UNDECIDED}$, if $\chi_{\mathsf{D}}(\upsilon) = 0$ for all $\upsilon \in Q_{\dot{p}}$.
4. $\mu_{\mathsf{D}}(Q_{\dot{p}}) = \text{UNKNOWN}$, if p is not in the language ($p \notin \mathcal{L}$).

The function μ_{D} is well-defined, as it cannot be the case that both conditions 1. and 2. are simultaneously fulfilled, see for example (Thimm & Kern-Isberner 2008c).

The above functions are sufficient to define the moderator of the framework.

Definition 9 (Moderator)**.** For a given preference relation $\succ$ among arguments, the *moderator* is a tuple $(\mu_{\mathsf{D}}, \chi_{\mathsf{D}}, \eta_{\mathsf{D},\succ})$.

The agents of the framework provide two functionalities. First, they propose initial arguments for a given literal (or its negation) submitted by the moderator of the framework, which will be roots of the dialectical trees to be constructed. For a given query h it may be necessary to examine both, all dialectical trees with a root argument for h and all dialectical trees with a root argument for $\overline{h}$, as a query for h can only be answered with NO if there is a warrant for $\overline{h}$. Second, the agents propose counterarguments to arguments of other agents[1] that are valid in the given argumentation line. An agent is not obliged to return all his valid arguments for a given query or all his counterarguments for a given argument. Therefore, it is possible to model different kinds of argumentation strategies given different instantiations of the following argument functions.

Definition 10 (Root argument function)**.** Let Π be a global belief base and let Δ be a local belief base. A *root argument function* $\varphi_{\Pi,\Delta}$ relative to Π and Δ is a function $\varphi_{\Pi,\Delta} : \mathcal{L} \rightarrow \mathfrak{P}(\mathsf{ARG}_{\Pi,\Delta})$ such that for every literal $h \in \mathcal{L}$ the set $\varphi_{\Pi,\Delta}(h)$ is a set of arguments for h or for $\overline{h}$ from Π and Δ.

Definition 11 (Counterargument function)**.** Let Π be a global belief base and let Δ be a local belief base. A *counterargument function* $\psi_{\Pi,\Delta}$ relative to Π and Δ is a function $\psi_{\Pi,\Delta} : \mathsf{SEQ} \rightarrow \mathfrak{P}(\mathsf{ARG}_{\Pi,\Delta})$ such that for every argumentation sequence $\Lambda \in \mathsf{SEQ}$ the set $\psi_{\Pi,\Delta}(\Lambda)$ is a set of attacks from Π and Δ on the last argument of Λ and for every $\langle \mathcal{B}, h\rangle \in \psi_{\Pi,\Delta}(\Lambda)$ it holds that $\eta_{\mathsf{D},\succ}(\Lambda + \langle \mathcal{B}, h\rangle) = 1$.

[1] Furthermore the agents can possibly propose counterarguments to their own arguments, but here we will not consider this case explicitly.

Here we assume that the root argument and counterargument functions of all agents are the same and especially *complete*, i. e, they return all possible arguments for the given situation and do not omit one.

Given the above definitions an agent of the framework is defined as follows.

Definition 12 (Agent)**.** Let Π be a global belief base. An *agent* relative to Π is a tuple $(\Delta, \varphi_{\Pi,\Delta}, \psi_{\Pi,\Delta})$ with a local belief base Δ relative to Π, a root argument function $\varphi_{\Pi,\Delta}$ and a counterargument function $\psi_{\Pi,\Delta}$.

Finally, the definition of an argumentation-based multi agent system can be given as follows.

Definition 13 (Argumentation-based multi agent system)**.** An *argumentation-based multi agent system* (ArgMAS) is a tuple $(M, \Pi, \{A_1, \ldots, A_n\})$ with a moderator M, a global belief base Π and agents $A_1, \ldots, A_n$ relative to Π.

Given an ArgMAS T and a query h, the framework produces an answer to h as follows. First, the moderator of T asks all agents for initial arguments for h and for $\overline{h}$ and starts a dialectical tree with each of them as root arguments. Then for each of these arguments, the moderator asks every agent for counterarguments and incorporates them into the corresponding dialectical trees accordingly. This process is repeated for every new argument until no more arguments can be constructed. Eventually the moderator analyses the resulting dialectical trees and returns the appropriate answer to the questioner. A dialectical tree built via this process is called an *argumentation product*. The answer behaviour of an ArgMAS is determined by the decision function of its moderator. For a query $h \in \mathcal{L}$ and and ArgMAS T the *answer* of T on h is $\mu_{\mathsf{D}}(\{\upsilon_1, \ldots, \upsilon_n\})$, where μ_{D} is the decision function of the moderator of T and $\{\upsilon_1, \ldots, \upsilon_n\}$ is the set of all argumentation products of T for h.

We conclude this section with an example that illustrates the above definitions.

Example 2. Suppose an ArgMAS $T = (M, \Pi, \{A_1, A_2\})$ with two agents A_1, A_2. The global belief base Π and the local belief bases Δ_1 resp. Δ_2 of the agents A_1 resp. A_2 are given by

$$\begin{aligned} \Pi &= \{a, b\}, \\ \Delta_1 &= \{(d \prec a, c), (c \prec b)\}, \\ \Delta_2 &= \{(\sim c \prec a, b)\}. \end{aligned}$$

Assume *Generalized Specificity* as the preference relation among arguments and let d be the query under consideration. When the moderator passes this query to the agents, only the root argument function of A_1 returns a non-empty set of root arguments, namely the set that contains the one argument $X_1 = \langle\{(d \prec a, c), (c \prec b)\}, d\rangle$. The moderator starts the construction of one dialectical tree with X_1 as its root. Then he asks every agent for counterarguments on X_1 that are acceptable after the argumentation line $[X_1]$. There only the counterargument function of A_2 returns the only possible counterargument $\langle\{(\sim c \prec a, b)\}, \sim c\rangle$. After that, no more arguments can be constructed and the final dialectical tree, i. e., the one final argumentation product υ can be seen in Figure 2. After applying his analysis function, the moderator determines that the root argument of υ is marked "defeated" and as υ is the only argumentation product of T on d, the answer of the decision function of the moderator and thus the answer of the system on d is UNDECIDED.

$$\langle\{(d \prec a, c), (c \prec b)\}, d\rangle$$
$$\Big|$$
$$\langle\{(\sim c \prec a, b)\}, \sim c\rangle$$

Figure 2: The one argumentation product υ in Example 2.

Collaborations

The distributed argumentation framework described above serves well when modeling scenarios, where the agents are involved in some kind of a dispute and have opposite views of the given query, such as a legal dispute, where agents take the roles of accuser and defender (Thimm 2008). But the framework fails to model situations, in which the agents should cooperate in order to reach a common solution, because they cannot share their beliefs in order to construct arguments that cannot be constructed by one agent alone.

Example 3. Let $T = (M, \Pi, \{A_1, A_2\})$ be an ArgMAS with $\Pi = \{a, d\}$ and let Δ_1 resp. Δ_2 be the local belief bases of A_1 resp. A_2 with

$$\begin{aligned} \Delta_1 &= \{(b \prec a), (b \prec a, c)\} \quad \text{and} \\ \Delta_2 &= \{(\sim b \prec a), (c \prec d)\} \quad . \end{aligned}$$

Given the query b, T yields two argumentation products

$$\begin{aligned} &[\langle\{(b \prec a)\}, b\rangle, \langle\{(\sim b \prec a)\}, \sim b\rangle] \quad \text{and} \\ &[\langle\{(\sim b \prec a)\}, \sim b\rangle, \langle\{(b \prec a)\}, b\rangle] \quad . \end{aligned}$$

As the roots of both argumentation products will be marked "defeated", the answer of T on b is UNDECIDED.

Observe that there is the additional argument $\langle\{(b \prec a, c), (c \prec d)\}, b\rangle$, that could be constructed, if both agents share their beliefs. This argument cannot be defeated by $\langle\{(\sim b \prec a)\}, \sim b\rangle$, as the first is more specific than the second, and thus would be a warrant for b. So in this case, the answer of the system for query b should be YES instead of UNDECIDED.

In (Móra, Alferes, & Schroeder 1998) a framework for cooperating agents in a context very similar to that of an ArgMAS was introduced. There – in contrast to here – extended logic programs (Gelfond & Lifschitz 1991) were used to model an agent's belief. We follow the ideas of (Móra, Alferes, & Schroeder 1998) to define the notion of *collaboration* and the mechanisms that allow our agents to cooperate in an ArgMAS, but extend their framework according to our needs.

We begin by defining a *collaboration* which describes a coalition of several agents, like a team of lawyers or a political party. A collaboration describes a set of agents each obliged to one another to support them with necessary information.

Definition 14 (Collaboration)**.** Let $T = (M, \Pi, \{A_1, \ldots, A_n\})$ be an ArgMAS. A *collaboration* C of T is a set of agents with $C \subseteq \{A_1, \ldots, A_n\}$.

A collaboration is basically a set of agents that form an alliance for argument construction. With the use of collaborations, we are able to derive more arguments than in the case with no collaborations. Observe, that we do not impose any conditions on collaborations. Although it might be appropriate to enforce the agents in a collaboration (for example) to have non-conflicting beliefs, we do not restrain the above definition to stay simple in our presentation. Furthermore, if an agent has conflicting beliefs with its partners in a collaboration, this does not affect the conjoint construction of arguments, since not all rules of each agents have to be in conflict.

Definition 15 (Collaborated argument)**.** Let $T = (M, \Pi, \mathfrak{A})$ be an ArgMAS and $C = \{A_1, \ldots, A_l\} \subseteq \mathfrak{A}$ a collaboration of T. If $\Delta_1, \ldots, \Delta_l$ are the local belief bases of $A_1, \ldots, A_l$, then an argument $\langle \mathcal{A}, h \rangle$ is a *collaborated argument* of the collaboration C iff $\langle \mathcal{A}, h \rangle \in \mathsf{ARG}_{\Pi, \Delta_1 \cup \ldots \cup \Delta_l}$, i. e., $\langle \mathcal{A}, h \rangle$ is an argument regarding the global belief base Π with $\mathcal{A} \subseteq \Delta_1 \cup \ldots \cup \Delta_l$.

Example 4. Consider again the ArgMAS of example 3. Suppose agents A_1 and A_2 are members of a collaboration C, i. e., $C = \{A_1, A_2\}$. Then

$$\langle \{(b \prec a, c), (c \prec d)\}, b \rangle$$

is a collaborated argument of C.

We call $\langle \mathcal{A}, h \rangle$ a *strict* collaborated argument of the collaboration C iff it is a collaborated argument of C and it can not be constructed by any agent alone, i. e., it is $\mathcal{A} \not\subseteq \Delta$ for every local belief base Δ of an agent in C. For instance, the argument in example 4 is a strict collaborated argument. In the upcoming algorithm, we do not intend to generate only strict collaborated arguments. This means, that the algorithm will also generate arguments, that could have been generated by an agent alone. A modification of the algorithm to suppress the generation of non-strict collaborated arguments is straightforward, but loses simplicity and clarity.

We can describe the intended behaviour of the distributed framework including collaborations by introducing meta agents, each representing a collaboration, and then subsuming the extended case with collaborations by the simple framework described in the last section. Without considering these meta agents, the generation of collaborated arguments must be done completely autonomously by the agents of a collaboration alone. We do not address this issue in the present work, but leave it open for future research. For now, assume that φ_C^{coll} resp. ψ_C^{coll} is a root argument resp. counterargument function that generates collaborated arguments of the collaboration C. We will give a formal definition of these functions and an operational description of their computation in the next subsection.

Definition 16 (Associated meta agent)**.** Let $T = (M, \Pi, \mathfrak{A})$ be an ArgMAS, $C = \{A_1, \ldots, A_l\} \subseteq \mathfrak{A}$ be a collaboration of T with $\Delta_1, \ldots, \Delta_l$ being the local belief bases of agents $A_1, \ldots, A_l \in \mathfrak{A}$ and η be an acceptance function. The agent $(\emptyset, \varphi_C^{coll}, \psi_C^{coll}, \eta)$ is called the *meta agent associated to the collaboration* C with functions $\varphi_C^{coll} : \mathcal{L} \to \mathfrak{P}(\mathsf{ARG}_{\Pi, \Delta_1 \cup \ldots \cup \Delta_l})$ and $\psi_C^{coll} : \mathsf{SEQ} \to \mathfrak{P}(\mathsf{ARG}_{\Pi, \Delta_1 \cup \ldots \cup \Delta_l})$.

Definition 17 (Collaborative ArgMAS)**.** A tuple $T = (M, \Pi, \{A_1, \ldots, A_n\}, \{C_1, \ldots, C_m\})$ is a *collaborative ArgMAS* if $T' = (M, \Pi, \{A_1, \ldots, A_n, A_{C_1}, \ldots, A_{C_m}\})$ is an ArgMAS with A_{C_i} being the meta agent associated to the collaboration C_i (for $1 \leq i \leq m$).

The above definition does not impose, that an agent cannot belong to more than one collaboration, but in the following we only consider the case, where $C_1, \ldots, C_m$ are disjoint.

Before turning to the operational aspects of computing collaborated arguments, we give a small example to illustrate collaborations.

Example 5. Let $\Pi = \{(h \leftarrow a, b), c, d\}$ and two local belief bases Δ_1, Δ_2 of two agents A_1, A_2 given by

$$\begin{aligned} \Delta_1 &= \{(a \prec c), (g \prec d)\}, \\ \Delta_2 &= \{(b \prec f), (f \prec g)\}. \end{aligned}$$

Let $C = \{A_1, A_2\}$ be a collaboration and A_C the corresponding meta agent. When asked for an argument for h the two agents alone A_1 and A_2 can obviously not return any. But when combing their beliefs, the meta agent A_C is able to generate the argument

$$\langle \{(a \prec c), (b \prec f), (f \prec g), (g \prec d)\}, h \rangle \quad ,$$

which makes also use of the strict rule $h \leftarrow a, b$.

Generating collaborated arguments

The key idea of computing collaborated arguments is similar to (Móra, Alferes, & Schroeder 1998) but uses another characterization of a *partial argument*. While Móra et al. impose a partial argument to be a partial derivation with no intermediate rules missing, we define a partial argument declaratively as an argument with some additional facts missing. For both, a partial argument is some kind of an intermediate result when constructing an argument in a distributed manner.

Definition 18 (Partial argument)**.** Let Π be a global belief base and $R \subseteq \mathsf{DEF}$ a set of defeasible rules. A tuple $\langle \mathcal{A}, h \rangle$ is a *partial argument* for a literal h regarding Π and R, iff $\mathcal{A} \subseteq R$ and there is a set of literals $F \subseteq \mathcal{L}$ such that $\langle \mathcal{A}, h \rangle \in \mathsf{ARG}_{\Pi \cup F, R}$, i. e., $\langle \mathcal{A}, h \rangle$ is an argument in $(\Pi \cup F, R)$. The smallest sets F (regarding set inclusion) satisfying this condition are called *free sets*. The set of all free sets is denoted $free(\langle \mathcal{A}, h \rangle)$ for a partial argument $\langle \mathcal{A}, h \rangle$. Let $\mathsf{PAR}_{\Pi, R}$ be the set of all partial arguments for the global belief base Π and a set of defeasible rules $R \subseteq \mathsf{DEF}$.

Example 6. Let $\Pi = \{(h \leftarrow a), (h \leftarrow b)\}$. Then $\langle \emptyset, h \rangle$ is a partial argument (regarding DEF), since there is a set of literals, namely $\{a\}$, such that $\langle \emptyset, h \rangle$ is an argument regarding $\Pi' = \{(h \leftarrow a), (h \leftarrow b), a\}$. The same is true for the set $\{b\}$, so the free sets of $\langle \emptyset, h \rangle$ regarding Π are given by

$$free(\langle \emptyset, h \rangle) = \{\{a\}, \{b\}\} \quad .$$

Example 7. Let $\Pi = \{(h \leftarrow a, b), b\}$ and $\mathcal{A} = \{(a \prec c)\}$. Then $\langle \mathcal{A}, h\rangle$ is a partial argument (regarding DEF), since $\{c\}$ is a free set of $\langle \mathcal{A}, h\rangle$ regarding Π:

$$free(\langle \mathcal{A}, h\rangle) = \{\{c\}\}$$

Observe, that every argument $\langle \mathcal{A}, h\rangle$ is also a partial argument (with $free(\langle \mathcal{A}, h\rangle) = \emptyset$), as well as $\langle \emptyset, h\rangle$ for any h.

Our approach to compute collaborated arguments is a top-down approach that starts with the empty set and iteratively adds defeasible rules until the given conclusion can be derived. For this purpose we equip every agent with a function that extends a given partial argument as much as possible.

Definition 19 (Partial argument function)**.** Let Π be a global belief base, A be an agent and Δ_A its local belief base. A *partial argument function* κ_A for agent A is a function $\kappa_A : \mathsf{PAR}_{\Pi,\mathsf{DEF}} \rightarrow \mathfrak{P}(\mathsf{PAR}_{\Pi,\mathsf{DEF}})$ and is defined as

$$\kappa_A(\langle \mathcal{A}, h\rangle) = \{\langle \mathcal{A}', h\rangle \in \mathsf{PAR}_{\Pi,\mathcal{A}\cup\Delta_A} \mid \mathcal{A}' \supset \mathcal{A}\}$$

Example 8. Let $\Pi = \{(d \leftarrow e)\}$ be a global belief base and A be an agent with a local belief base

$$\Delta = \{(g \prec c, d), (e \prec f)\} \quad .$$

Then it is

$$\kappa_A(\langle \emptyset, g\rangle) = \{\langle \mathcal{A}_1, g\rangle, \langle \mathcal{A}_2, g\rangle\}$$

with

$$\begin{aligned} \mathcal{A}_1 &= \{(g \prec c, d)\}, \\ \mathcal{A}_2 &= \{(g \prec c, d), (e \prec f)\}. \end{aligned}$$

Furthermore it is $free(\langle \mathcal{A}_1, g\rangle) = \{\{c, d\}, \{c, e\}\}$ and $free(\langle \mathcal{A}_2, g\rangle) = \{\{c, f\}\}$

Using the partial argument functions of the agents in a collaboration, the associated meta agent is able to compute the collaborated arguments for a given literal h with Algorithm 1. The algorithm `CollaboratedArguments` takes as input a global belief base Π, a collaboration of agents $\{A_1, \ldots, A_l\}$ and a literal h, and it returns the set of all collaborated arguments of $\{A_1, \ldots, A_l\}$ in a backward chaining manner.

```
1  CollaboratedArguments(Π, {A_1, ..., A_l}, h)
2    iArgs := {⟨∅, h⟩}
3    cArgs := ∅
4    while iArgs ≠ ∅
5      remove a tuple ⟨A, h⟩ from iArgs
6      if free(⟨A, h⟩) = ∅ then
7        cArgs := cArgs ∪ ⟨A, h⟩
8      else
9        for i from 1 to l
10         iArgs := iArgs ∪ κ_{A_i}(⟨A, h⟩)
11   return cArgs
```

Algorithm 1: Construction of collaborated arguments

First, the algorithm initializes the set of partial arguments `iArgs` with the trivial partial argument $\langle \emptyset, h\rangle$ (line 2). As long as there are partial arguments available, the algorithm removes one of them from `iArgs` and extends it in every possible way, i. e., by eliminating free literals from any free set by every agents' partial argument function. When an argument is complete, i. e., $free(\langle \mathcal{A}, h\rangle) = \emptyset$, the argument can be added to the result set (line 6,7).

Example 9. Let Π be a global belief base with

$$\Pi \quad = \quad \{(g \leftarrow c), (d \leftarrow f), a, b\}$$

and let A_1, A_2 be two agents with local belief bases Δ_1, Δ_2, respectively, given by

$$\begin{aligned} \Delta_1 &= \{(c \prec a), (c \prec h), (d \prec e)\} \quad \text{and} \\ \Delta_2 &= \{(g \prec d), (e \prec b)\} \quad . \end{aligned}$$

Let g be a query and consider the following exemplary execution of `CollaboratedArguments` on the call `CollaboratedArguments(`Π`,`$\{A_1, A_2\}$`,`g`)`.

First, the set `iArgs` is initialized with the partial argument $X_1 = \langle \emptyset, g\rangle$. As $free(X_1) = \{\{g\}, \{c\}\} \neq \emptyset$ the algorithm continues at line 10. There, $\kappa_{A_1}(X_1)$ is called yielding $\{X_2, X_3\}$ as the set of possible extensions to X_1 with $X_2 = \langle (c \prec a), g\rangle$ and $X_3 = \langle (c \prec h), g\rangle$. Then $\kappa_{A_2}(X_1)$ is called yielding $X_4 = \langle (g \prec d), g\rangle$ as a possible extension to X_1. So back at line 4 we have `iArgs`$= \{X_2, X_3, X_4\}$.

Let then X_3 be chosen at line 5. As $free(X_3) = \{\{h\}\}$ the algorithm continues at line 10. Neither agent can extend X_3 because neither has a defeasible rule with head h nor is there a strict rule with head h, so it is $\kappa_{A_1}(X_3) = \kappa_{A_2}(X_3) = \emptyset$. Back at line 4 we have `iArgs`$= \{X_2, X_4\}$.

Let then X_2 be chosen at line 5. As $free(X_2) = \emptyset$ the algorithm continues at line 7 and X_2 is added to the result set `cArgs`.

Now it is `iArgs`$= \{X_4\}$ and $free(X_4) = \{\{d\}\}$. Continuing at line 10, $\kappa_{A_1}(X_4)$ is called yielding $X_5 = \langle (g \prec d), (d \prec e), g\rangle$ and $\kappa_{A_2}(X_4)$ is called yielding no further extension to X_4. So back at line 4 we have `iArgs`$= \{X_5\}$ with $free(X_5) = \{\{e\}\}$. Finally, agent A_2 completes X_5 yielding $X_6 = \langle (g \prec d), (d \prec e), (e \prec b), g\rangle$ with $free(X_6) = \emptyset$.

So, `CollaboratedArguments(`Π`,`$\{A_1, A_2\}$`,`g`)` returns the set `cArgs`$= \{X_2, X_6\}$.

Using the `CollaboratedArguments` algorithm we are now able to define the root argument and counterargument functions of the associated meta agents.

Definition 20 (Root argument function φ_C^{coll})**.** Let Π be a global belief base, $C = \{A_1, \ldots, A_l\}$ a collaboration with Δ_i being the local belief base of agent A_i ($1 \leq i \leq l$) and h a literal. The function $\varphi_C^{coll} : \mathcal{L} \rightarrow \mathfrak{P}(\mathsf{ARG}_{\Pi,\Delta_1\cup\ldots\cup\Delta_l})$ is defined as

$$\begin{aligned} \varphi_C^{coll}(h) \quad = \quad & \texttt{CollaboratedArguments(}\Pi\texttt{,}C\texttt{,}h\texttt{)} \cup \\ & \texttt{CollaboratedArguments(}\Pi\texttt{,}C\texttt{,}\sim h\texttt{)} \end{aligned}$$

Definition 21 (Counterargument function ψ_C^{coll})**.** Let Π be a global belief base, C $C = \{A_1, \ldots, A_l\}$ a collaboration with Δ_i being the local belief base of agent A_i ($1 \leq i \leq l$) and h a literal. Let λ be an argumentation sequence. The function $\psi_C^{coll} : \mathsf{SEQ} \rightarrow \mathfrak{P}(\mathsf{ARG}_{\Pi,\Delta_1\cup\ldots\cup\Delta_l})$ is defined as

$$\begin{aligned} \psi_C^{coll}(\lambda) = \{\langle \mathcal{A}, h\rangle \in \mathsf{ARG}_{\Pi,\Delta_1\cup\ldots\cup\Delta_l} \mid \exists h : \langle \mathcal{A}, h\rangle \in \\ \texttt{CollaboratedArguments(}\Pi\texttt{,}C\texttt{,}h\texttt{)} \wedge \\ \lambda + \langle \mathcal{A}, h\rangle \text{ is acceptable } \} \end{aligned}$$

Example 10. We continue Example 5. So let $\Pi = \{(h \leftarrow a, b), c, d\}$ and two local belief bases Δ_1, Δ_2 of two agents A_1, A_2 given by

$$\begin{aligned} \Delta_1 &= \{(a \prec c), (g \prec d)\}, \\ \Delta_2 &= \{(b \prec f), (f \prec g)\}. \end{aligned}$$

Let there be a collaboration $C = \{A_1, A_2\}$ and A_C the corresponding meta agent. Given the query h, the agent A_C would use his root argument function φ_C^{coll} and thus the algorithm `CollaboratedArguments` in order to generate a root argument for or against h. In the algorithm `CollaboratedArguments` for the literal h the set `iArgs` is initialized with $\{\langle\emptyset, h\rangle\}$ with $free(\langle\emptyset, h\rangle) = \{\{h\}, \{a, b\}\}$. This means, that the partial argument $\langle\emptyset, h\rangle$ can be completed by either an argument for h or by arguments for both a and b using the strict rule $(h \leftarrow a, b)$. Observe that there is no possibility to complete $\langle\emptyset, h\rangle$ without the use of the strict rule $(h \leftarrow a, b)$, as no agent has a defeasible rule with h as its head.

Next, suppose, that A_C asks agent A_1 to extend the partial argument $\langle\emptyset, h\rangle$. As A_1 has a defeasible rule for a, he can extend $\langle\emptyset, h\rangle$ to $\langle\mathcal{A}, h\rangle$ with $\mathcal{A} = \{(a \prec c)\}$ and $free(\langle\mathcal{A}, h\rangle) = \{\{b\}\}$. Agent A_2 can then extend $\langle\mathcal{A}, h\rangle$ to $\langle\mathcal{A}', h\rangle$ with $\mathcal{A}' = \{(a \prec c), (b \prec f), (f \prec g)\}$ and $free(\langle\mathcal{A}', h\rangle) = \{\{g\}\}$ and finally agent A_1 can extend $\langle\mathcal{A}', h\rangle$ to $\langle\mathcal{A}'', h\rangle$ with $\mathcal{A}'' = \{(a \prec c), (b \prec f), (f \prec g), (g \prec d)\}$ and $free(\langle\mathcal{A}'', h\rangle) = \emptyset$.

Observe, that the algorithm `CollaboratedArguments` generates this argument also on other ways than the described above, e. g. by first using the partial argument $\langle(b \prec f), h\rangle$ provided by A_2.

Soundness and completeness

We will now show, that the algorithm `CollaboratedArguments` is sound and complete. The soundness and completeness of the root argument function φ_C^{coll} and the counterargument function ψ_C^{coll} then follow directly.

We start by showing soundness, i. e., that every argument $\langle\mathcal{A}, h\rangle$ that is returned by `CollaboratedArguments(`Π`,`$\{A_1, \ldots, A_n\}$`,`h`)` is indeed a collaborated argument of C with conclusion h.

Theorem 1 (Soundness)**.** *Let Π be a global belief base, $C = \{A_1, \ldots, A_n\}$ be a collaboration of agents $A_1, \ldots, A_n$ with local belief bases $\Delta_1, \ldots, \Delta_n$ respectively. If h is a literal and*

$$\langle\mathcal{A}, h'\rangle \in \texttt{CollaboratedArguments(}\Pi\texttt{,}C\texttt{,}h\texttt{)},$$

then $h = h'$ and $\langle\mathcal{A}, h'\rangle$ is a collaborated argument of C.

Proof. It is clear due to line 7 of Algorithm 1 that $h = h'$. So it remains to show, that $\langle\mathcal{A}, h\rangle$ is a collaborated argument of C, i. e., that $\langle\mathcal{A}, h\rangle$ is an argument of (Π, Δ') with $\Delta' = \Delta_1 \cup \ldots \cup \Delta_n$.

1. Clearly it is $\mathcal{A} \subseteq \Delta'$ because the partial argument function of an agent A_i ($1 \leq i \leq n$) only adds defeasible rules to the argument that belong to $\Delta_i \subseteq \Delta'$.
2. $\langle\mathcal{A}, h\rangle$ defeasibly derives h, because it is $free(\langle\mathcal{A}, h\rangle) = \emptyset$ (line 6 in Algorithm 1).
3. $\mathcal{A}$ is non-contradictory, because the partial argument functions of the agents only return partial arguments due to definition. A partial argument must be non-contradictory, as there must be an extension (possibly an extension by $\emptyset$ as in the last completion step of a partial argument) that is an argument and hence non-contradictory.
4. $\langle\mathcal{A}, h\rangle$ is minimal using the same argumentation as above.

□

Furthermore our algorithm is complete in the sense, that if $\langle\mathcal{A}, h\rangle$ is a collaborated argument of a collaboration C with respect to a global belief base Π, then $\langle\mathcal{A}, h\rangle$ will be returned by the algorithm `CollaboratedArguments`.

Theorem 2 (Completeness)**.** *Let Π be a global belief base, $C = \{A_1, \ldots, A_n\}$ be a collaboration of agents $A_1, \ldots, A_n$ with local belief bases $\Delta_1, \ldots, \Delta_n$ respectively. If $\langle\mathcal{A}, h\rangle$ is a collaborated argument of C then it is*

$$\langle\mathcal{A}, h\rangle \in \texttt{CollaboratedArguments(}\Pi\texttt{,}C\texttt{,}h\texttt{)} \quad .$$

Proof. We have to show, that $\langle\mathcal{A}, h\rangle$ is added to the set `cArgs` at line 7 of Algorithm 1. If h can be strictly derived from Π, i. e., it is $\mathcal{A} = \emptyset$, then it is $free(\langle\mathcal{A}, h\rangle) = \emptyset$ and $\langle\mathcal{A}, h\rangle$ is added at line 7 of Algorithm 1. Otherwise, as $\langle\mathcal{A}, h\rangle$ is a collaborated argument of C, there is a defeasible rule $r \in \mathcal{A}$ with $head(r) \in K$ for some $K \in free(\langle\emptyset, h\rangle)$. Let $r \in \Delta_k$ for some $k \in \{1, \ldots, n\}$, then agent A_k will extend the partial argument $\langle\emptyset, h\rangle$ with at least rule r in line 10 of Algorithm 1. Inductively it follows that there is always an extension of this argument by rules of $\mathcal{A}$. As $\langle\mathcal{A}, h\rangle$ is an argument, it is $free(\langle\mathcal{A}, h\rangle) = \emptyset$ and so $\langle\mathcal{A}, h\rangle$ is added to `cArgs` in line 7 of Algorithm 1. □

Related work and comparison

In (Thimm & Kern-Isberner 2008b) it has been shown, that the distributed framework without collaborations subsumes ordinary DeLP, as every defeasible logic program can be translated into an equivalent distributed framework with the same answer behaviour. The other way round is not always possible, as there are distributed settings, where there is no equivalent single defeasible logic program that models the same situation. With the use of collaborations we are now able to establish an equivalence between a special case of the distributed framework with collaborations and ordinary DeLP. For the special case of a collaborative ArgMAS with one collaboration involving all agents, the answer behaviour is the same as when considering a defeasible logic program which is built upon the union of all local belief bases.

Many other proposals exist for introducing argumentative capabilities into distributed systems and especially negotiation systems, see for example (Kraus 1997; Amgoud, Dimopolous, & Moraitis 2007; Bench-Capon 2003; Rueda, Garcia, & Simari 2002; Karunatillake *et al.* 2005).

There are especially two other approaches, that have similarities with the approach proposed in this paper. The framework of (Móra, Alferes, & Schroeder 1998; de Almeida & Alferes 2006) uses extended logic programs to model an

agent's belief and defines a notion of distributed argumentation using these extended logic programs. The framework uses the argumentation semantics from (Prakken 1997) and defines a notion of cooperation, that allows the agents to share their beliefs in order to construct new arguments. As this framework uses extended logic programs as the underlying representation formalism, it has a declarative semantics in contrast to the dialectical semantics of DeLP used here.

Black et al. (Black 2007; Black & Hunter 2007) also use defeasible logic programming as the underlying representation formalism to model distributed argumentation. Complementary to the proposal in this paper, the focus of (Black 2007) is on modeling communication protocols and strategies for successful argumentation between agents. They introduce two kinds of inquiry dialogues, one to generate combined arguments and one for the actual argumentation.

Conclusion

Usually, argumentation is considered as a dialectical process which involves two parties, a proponent and an opponent who generate arguments in order to evaluate reasons in favor of or against claims. Argumentation might even reflect deliberations taking place within one single agent.

In this paper, we study argumentation in distributed scenarios in which the pro and con parties consist of several collaborating agents, each agent possessing its own subjective beliefs but sharing strict knowledge with all other agents. As a proper framework to realize such distributed argumentation, we choose DeLP (García & Simari 2004) since it allows a distinction between strict, commonly known world knowledge, on one side, and subjective and defeasible beliefs, on the other. Via collaborations, the agents may produce more and better arguments as any of them might bring forward when only using its own belief base. For each collaboration, we introduce a meta agent that organizes the generation of arguments and counterarguments from the rule reservoir of each agent in a dialogue. As a crucial concept for handling fragments of arguments effectively to build complete arguments, we defined *partial arguments* by modifying an idea from (Móra, Alferes, & Schroeder 1998).

For the operational part of our approach, we present an algorithm to generate collaborated arguments, and prove its soundness and completeness. Finally, we show that the results in this paper generalize the approach proposed in (Thimm & Kern-Isberner 2008a), and compare our work to related approaches.

As part of our ongoing work, we explore different applications of our collaborative argumentation framework. One particularly appealing scenario is to realize negotiations in a multi-agent system under confidentiality constraints. In such scenario, each agent tries to hide its subjective beliefs as well as possible, while at the same time being interested in making as much information available as necessary to reach a good negotiation result.

Acknowledgments The authors thank the reviewers for their very helpful comments to improve the original version of this paper.

References

Amgoud, L.; Dimopolous, Y.; and Moraitis, P. 2007. A unified and general framework for argumentation-based negotiation. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems, AAMAS'2007*.

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. 2000. Towards a logic-based theory of argumentation. In *Proc. of the 17th American Nat. Conf. on Artif. Intelligence (AAAI'2000)*, 411–416.

Black, E., and Hunter, A. 2007. A generative inquiry dialogue system. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07)*. IEEE Press.

Black, E. 2007. *A Generative Framework for Argumentation-Based Inquiry Dialogues*. Ph.D. Dissertation, University College London.

Booth, R. 2002. Social contraction and belief negotiation. In *Proceedings of the Eighth Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, 375–384.

de Almeida, I. C., and Alferes, J. J. 2006. An argumentation-based negotiation for distributed extended logic programs. In *Proceedings of CLIMA VII*, 191–210.

García, A., and Simari, G. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1-2):95–138.

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

Karunatillake, N. C.; Jennings, N. R.; Rahwan, I.; and Norman, T. J. 2005. Argument-based negotiation in a social context. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 1331–1332. New York, NY, USA: ACM.

Kraus, S. 1997. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence* 94(1-2):79–97.

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

Móra, I. A.; Alferes, J. J.; and Schroeder, M. 1998. Argumentation and cooperation for distributed extended logic programs. In *Nonmonotonic Reasoning Workshop'98*.

Parsons, S.; Sierra, C.; and Jennings, N. 1998. Agents that reason and negotiate by arguing. *Journal of Logic and Computation* 8(3):261–292.

Prakken, H., and Vreeswijk, G. 2002. Logical systems for defeasible argumentation. In Gabbay, D., and Guenthner, F., eds., *Handbook of Philosophical Logic*, volume 4. Dordrecht: Kluwer Academic Publishers, 2 edition. 219–318.

Prakken, H. 1997. Dialectical proof theory for defeasible argumentation with defeasible priorities (preliminary report). In *ModelAge Workshop*, 202–215.

Rahwan, I., and Amgoud, L. 2006. An argumentation-based approach for practical reasoning. In Weiss, G., and Stone, P., eds., *5th International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS'2006*, 347–354.

Rueda, S. V.; Garcia, A.; and Simari, G. R. 2002. Argument-based negotiation among BDI agents. *Journal of Computer Science and Technology* 2(7).

Stolzenburg, F.; García, A.; Chesnevar, C. I.; and Simari, G. 2003. Computing generalized specificity. *Journal of Non-Classical Logics* 13(1):87–113.

Thimm, M., and Kern-Isberner, G. 2008a. A distributed argumentation framework using defeasible logic programming. In Besnard, P.; Doutre, S.; and Hunter, A., eds., *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA'08)*, number 172 in Frontiers in Artificial Intelligence and Applications, 381–392. Toulouse, France: IOS Press.

Thimm, M., and Kern-Isberner, G. 2008b. A distributed argumentation framework using defeasible logic programming (extended version). Technical report, Technische Universität Dortmund.

Thimm, M., and Kern-Isberner, G. 2008c. On the relationship of defeasible argumentation and answer set programming. In Besnard, P.; Doutre, S.; and Hunter, A., eds., *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA'08)*, number 172 in Frontiers in Artificial Intelligence and Applications, 393–404. Toulouse, France: IOS Press.

Thimm, M. 2008. *Verteilte logikbasierte Argumentation: Konzeption, Implementierung und Anwendung im Rechtswesen*. VDM Verlag Dr. Müller.

GIDL: A Grounder for FO$^+$

Johan Wittocx[*] and **Maarten Mariën** and **Marc Denecker**
Department of Computer Science, K.U. Leuven, Belgium
{johan,maartenm,marcd}@cs.kuleuven.be

Abstract

In this paper, we present GIDL, a grounder for FO$^+$. FO$^+$ is a very expressive extension of first-order logic with several constructs such as inductive definitions, aggregates and arithmetic. We describe the input and output language of GIDL, and provide details about its architecture. In particular, the core grounding algorithm implemented in GIDL is presented. We compare GIDL with other FO$^+$ grounders and with grounders for Answer Set Programming.

Introduction and Motivation

The ambition of *declarative problem solving* is, in a nutshell, that a human expert represents his knowledge as a precise logic specification in terms of a vocabulary formalizing relevant objects and concepts of the problem domain, and solves computational tasks within this domain by applying suitable forms of logical inference on the logic specification. The success of a declarative problem solving framework depends on three main factors: the quality of the logic as a specification language, the flexibility of the logical inference to solve a broad class of computational problems, and the availability of efficient solvers.

An important and flexible logical inference task is finite (Herbrand) model generation. Indeed, in many real-life computational problems, one searches for objects of a complex nature, e.g., plans, schedules, assignments, etc. Such objects are often represented as (finite) structures. Model generation serves to explicitly construct such a structure, given an implicit description of it by means of a logic theory. The idea of a declarative problem solving framework based on computing "solutions" as the models of a theory was presented for the first time in (Marek & Truszczyński 1998) in the context of Answer Set Programming (ASP). Earlier, SAT-solvers had been used in this spirit, for example in Kautz and Selman's blackbox approach to planning problems (Kautz & Selman 1996). And, as recently pointed out in (Mitchell & Ternovska 2008), problem solving in Constraint Programming (CP) systems often amounts to computing models of first-order logic (FO) specifications.

In (Mitchell & Ternovska 2005), a declarative framework based on *model expansion* (MX) was presented. MX for a logic $\mathcal{L}$, denoted MX($\mathcal{L}$), extends model generation: it takes as input not only an $\mathcal{L}$-theory T over a vocabulary Σ and finite domain D, but also a structure I_σ with domain D, interpreting a subvocabulary $\sigma \subseteq \Sigma$. It searches to expand I_σ into a Σ-model of T. The input interpretation I_σ presents a convenient way to store *data* of a problem.

From a computational point of view, an interesting aspect of finite model generation and MX is that its complexity remains in NP for every logic for which the model checking problem is in P. This is the case for, e.g., first-order logic (FO) and many extensions of it, which are languages par excellence for describing many real-life computational problems. In this paper, we consider MX for such a logic, namely full first-order logic extended with aggregates, inductive definitions, arithmetic, partial functions and ordered sorts. We denote this logic by FO$^+$. Clearly, FO$^+$ is an expressive language, convenient for modelling a broad class of domains.

An important result in (Mitchell & Ternovska 2005) states that in the context of MX, FO is sufficient to solve all problems in NP. More precisely, for every NP decision problem on finite σ-structures, there exists a vocabulary $\Sigma \supseteq \sigma$ and a theory T over Σ such that a σ-structure I_σ is accepted iff there exists a model of T expanding I_σ. Hence, the class of problems that can be *represented* in MX(FO$^+$) and MX(FO) is exactly the same. In practice however, new language primitives, such as the ones in FO$^+$, may seriously ease the modelling task and enlarge the class of problems that can be *solved* by practical implementations. As an example, consider the concept of *reachability* in a graph, which is often needed to model, e.g., planning or scheduling problems. This concept can be expressed in MX(FO), but not in a simple and natural manner: it requires a non-trivial encoding of an iterative fixpoint construction in FO. To allow for a direct, natural representation, one can consider MX for FO(ID), an extension of FO with *inductive definitions* (Denecker 2000). Besides making the modelling task easier, the resulting MX problem can be solved more efficiently, at least by the current generation of solvers. A similar argument applies for other language primitives, such as aggregates and arithmetic.

Currently, most model generation systems, and hence also MX solvers, consist of two components: a *grounder* and a *propositional solver*. The grounder transforms the input to an equivalent propositional theory, whose models are then

[*]Research assistant of the *Fonds voor Wetenschappelijk Onderzoek - Vlaanderen* (FWO Vlaanderen)

computed by the propositional solver. Several grounders for (fragments of) MX(FO$^+$) are being developed. MXIDL (Mariën, Wittocx, & Denecker 2006), the first implemented MX(FO(ID)) grounder, works by translating its input into an equivalent normal logic program, according to the transformation described in (Mariën, Gilis, & Denecker 2004), and then calls a (slightly adapted) grounder for ASP. MXIDL can handle full many-sorted FO(ID), extended with arithmetic. The first native grounding algorithm for MX(FO(ID)) was described in (Patterson *et al.* 2007), and partially implemented in the MXG system (Mitchell *et al.* 2006). MXG allows function-free FO, cardinality aggregates and a very restricted form of inductive definitions as input.

In this paper, we present GIDL, a new MX grounder, designed to handle a very expressive input language. It is tightly coupled with the propositional solvers MIDL (Mariën, Wittocx, & Denecker 2007) and MINISAT(ID) (Mariën *et al.* 2008), developed in our group. GIDL's input language is full FO$^+$: full order-sorted FO(ID), extended with cardinality, sum and product aggregates, partial functions and arithmetic. We present this input language in detail and describe GIDL's architecture. In particular, we present the core grounding algorithm, which is different from the one in MXG. We compare GIDL to MXIDL and MXG, showing that it is currently the fastest MX grounder. We also compare GIDL to grounders for ASP and to PSGRND (East *et al.* 2006), a grounder for the logic of propositional schemata (East & Truszczynski 2006a).

Preliminaries

In this section, we present many-sorted FO and FO(ID) and formally define the concepts of model expansion and grounding. We assume the reader is familiar with standard FO.

Many-Sorted First-Order Logic with Equality

A *vocabulary* Σ consists of a set Σ_S of sorts, and of variables, constant, predicate and function symbols. Variables and constant symbols are denoted by lowercase letters, predicate and function symbols by uppercase letters. Sets and tuples of variables are denoted by $\overline{x}, \overline{y}, \ldots$. Each variable x and constant symbol c has an associated sort $\mathfrak{s}(x)$, respectively $\mathfrak{s}(c) \in \Sigma_S$, each predicate symbol P with arity n an associated tuple of sorts $\mathfrak{s}(P) \in \Sigma_S^n$, and each function symbol F with arity n an associated tuple $\mathfrak{s}(F) \in \Sigma_S^{n+1}$.

A *term* over a vocabulary Σ is inductively defined as follows:

- A variable x of Σ is a term of sort $\mathfrak{s}(x)$.
- A constant c of Σ is a term of sort $\mathfrak{s}(c)$.
- If F is a function symbol of Σ with $\mathfrak{s}(F) = (s_1, \ldots, s_n, s_{n+1})$, and $t_1, \ldots, t_n$ are terms over Σ of sort respectively $s_1, \ldots, s_n$, then $F(t_1, \ldots, t_n)$ is a term of sort s_{n+1}.

The sort of a term t is denoted by $\mathfrak{s}(t)$. A (well-sorted) FO formula over Σ is inductively defined by:

- If P is a predicate symbol with $\mathfrak{s}(P) = (s_1, \ldots, s_n)$ and $t_1, \ldots, t_n$ are terms of sort respectively $s_1, \ldots, s_n$, then $P(t_1, \ldots, t_1)$ is a formula.
- If t_1 and t_2 are two terms of the same sort, then $t_1 = t_2$ is a formula.
- If φ and ψ are formulas and x is a variable, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\exists x\ \varphi$ and $\forall x\ \varphi$ are formulas.

An *atom* is a formula of the form $P(\bar{t})$ or $t_1 = t_2$. A *literal* is an atom or the negation of an atom. An occurrence of a formula φ as subformula in a formula ψ is *positive* (*negative*) if it occurs in the scope of an even (odd) number of negations. For a formula φ, we often write $\varphi[\overline{x}]$ to indicate that $\overline{x}$ are its free variables. The formula $\varphi[x/c]$ is the formula obtained by replacing in φ all free occurrences of the variable x by the constant symbol c. This notation is extended to tuples of variables and constant symbols of the same length. A *sentence* is a formula without free variables.

A Σ-*interpretation* I consists of

- a domain s^I for each sort $s \in \Sigma$;
- a domain element $x^I \in s^I$ for each variable x with $\mathfrak{s}(x) = s$;
- a domain element $c^I \in s^I$ for each constant c with $\mathfrak{s}(c) = s$;
- a relation $P^I \in s_1^I \times \ldots \times s_n^I$ for each predicate symbol P with $\mathfrak{s}(P) = (s_1, \ldots, s_n)$;
- a function $F^I : s_1^I \times \ldots \times s_n^I \to s_{n+1}^I$ for each function symbol F with $\mathfrak{s}(F) = (s_1, \ldots, s_{n+1})$.

A Σ-*structure* is an interpretation of only the sorts, constant, relation and function symbols of Σ. The restriction of a Σ-interpretation I to a vocabulary $\sigma \subseteq \Sigma$ is denoted by $I|_\sigma$. For a variable x and domain element d, $I[x/d]$ is the interpretation that assigns d to x and corresponds to I on all other symbols. This notation is extended to variables and domain elements of the same length.

The value t^I of a term t in an interpretation I, and the satisfaction relation $\models$ are defined as usual (see, e.g., (Enderton 1972)).

Inductive Definitions and FO(ID)

FO(ID) (Denecker 2000) is an extension of FO with inductive definitions. It can be viewed as an integration of FO with logic programming.

A *definition* over a vocabulary Σ is a finite set of rules of the form

$$\forall \overline{x}\ (P(\bar{t}) \leftarrow \varphi[\overline{y}]),$$

where φ is an FO formula over Σ, $\overline{y} \subseteq \overline{x}$, P is a predicate in Σ and $\bar{t}$ a tuple of terms such that $P(\bar{t})$ is well-sorted. Also, the set of variables occurring in $\bar{t}$ is a subset of $\overline{x}$. $P(\bar{t})$ is called the *head* of the rule, φ the *body*. The connective $\leftarrow$ is called *definitional implication* and is to be distinguished from material implication $\supset$. A predicate appearing in the head of a rule of a definition Δ is called a *defined* predicate of Δ, any other predicate symbol and each constant and function symbol is called an *open symbol* of Δ. The set of open symbols of Δ is denoted by $\mathrm{Open}(\Delta)$, the set of defined predicates by $\mathrm{Def}(\Delta)$. An occurrence of a formula φ

in a rule body is positive (negative) if it occurs in the scope of an odd (even) number of negations.

A Σ-interpretation I is said to satisfy a definition Δ over Σ, denoted $I \models \Delta$, if $I|_{\mathrm{Def}(\Delta)}$ is the well-founded model of Δ extending $I|_{\mathrm{Open}(\Delta)}$. The definition of well-founded model can be found in (Van Gelder, Ross, & Schlipf 1991; Denecker & Ternovska 2004).

An FO(ID) theory T is a finite set of FO sentences and definitions. An interpretation is a model of T iff it satisfies all sentences and definitions of T.

Model Expansion

Model expansion for a logic $\mathcal{L}$, abbreviated MX($\mathcal{L}$), was first presented as a declarative problem solving paradigm in (Mitchell & Ternovska 2005). For representation theorems, like the *capturing NP* property mentioned in the introduction, and for a comparison with other paradigms, we refer the reader to that paper.

Definition 1 (MX($\mathcal{L}$)). Given an $\mathcal{L}$ theory over a vocabulary Σ, a vocabulary $\sigma \subseteq \Sigma$ with the same set of sorts, and a finite σ-structure I_σ, the *model expansion search problem* for input $\langle \Sigma, T, \sigma, I_\sigma \rangle$ is the problem of finding models M of T that expand I_σ, i.e., $M|_\sigma = I_\sigma$. The *MX($\mathcal{L}$) decision problem* is the problem of deciding whether such a model exists.

The vocabulary σ is called the *instance vocabulary* of the problem, the vocabulary $\Sigma \setminus \sigma$ the *expansion vocabulary*. I_σ is called the *instance structure*.

Observe that if T is a theory over a vocabulary Σ containing no function symbols, Herbrand model generation for T can be simulated by MX. Indeed, let σ be the set of constants in Σ, the domain of I_σ the Herbrand universe and $c^{I_\sigma} = c$ for every constant $c \in \sigma$.

On the other hand, when $\sigma = \Sigma$ solving the MX decision problem boils down to model checking.

The following are two examples of MX(FO(ID)) representations of well-known computational problems.

Example 1 (Graph Colouring). The instance vocabulary consists of two sorts, Vtx and $Colour$, representing respectively the vertices of the given graph and the colours. It also contains a predicate symbol $Edge$ with sort (Vtx, Vtx), representing the edges of the given graph. The expansion vocabulary consists of a single function symbol $Colouring$ of sort $(Vtx, Colour)$, representing the solution. The only sentence in the theory is $\forall v_1, v_2 \ (Edge(v_1, v_2) \supset Colouring(v_1) \neq Colouring(v_2))$.

For the instance structure I_σ given by $Vtx^{I_\sigma} = \{a; b; c\}$, $Colour^{I_\sigma} = \{blue; red\}$ and $Edge^{I_\sigma} = \{a, b; b, c\}$, a sample solution to the MX search problem is the structure M, expanding I_σ with $Colouring^M(a) = blue$, $Colouring^M(b) = red$ and $Colouring^M(c) = blue$.

Example 2 (Hamiltonian Path). The instance vocabulary contains a sort Vtx, a predicate symbol $Edge$ of sort (Vtx, Vtx) and a constant $Start$ of sort Vtx, which represents the first vertex in the path. The expansion vocabulary contains a predicate In and $Reached$, where $\mathfrak{s}(In) = (Vtx, Vtx)$ and $\mathfrak{s}(Reached) = Vtx$. In represents the edges that are in the path. The theory is given by

$$\forall v_1, v_2 \ (In(v_1, v_2) \supset Edge(v_1, v_2)).$$
$$\forall v_1, v_2, v_3 \ (In(v_1, v_2) \wedge In(v_1, v_3) \supset v_2 = v_3).$$
$$\forall v_1, v_2, v_3 \ (In(v_1, v_3) \wedge In(v_2, v_3) \supset v_1 = v_2).$$
$$\forall v \ \neg In(v, Start).$$
$$\forall v \ Reached(v).$$
$$\left\{ \begin{array}{l} \forall v \ Reached(v) \leftarrow v = Start. \\ \forall v \ Reached(v) \leftarrow Reached(w) \wedge In(w, v). \end{array} \right\}$$

Grounding

Solving the MX(FO(ID)) search or decision problem for input $\langle \Sigma, T, \sigma, I_\sigma \rangle$ can be done by creating an "equivalent" propositional theory T_g using T and I_σ and subsequently calling a model generator (in case of the search problem) or satisfiability checker (in case of the decision problem) for the propositional fragment of FO(ID). For solving the MX(FO(ID)) decision problem, it suffices that T_g is satisfiable iff T has a model expanding I_σ. For solving the search problem, a one-to-one correspondence between the models of T_g and the models of T expanding I_σ is required. Because GIDL is meant to be a grounder for the search problem, we consider the latter, stronger type of equivalence in this paper.

We now define grounding formally. Let σ be a subvocabulary of Σ with the same set of sorts and let I_σ be a σ-structure. Denote by Σ^{I_σ} the vocabulary Σ, extended with a new constant symbol $\mathbf{d}$ for every $d \in s^{I_\sigma}$, $s \in \Sigma_S$. We call these new constants *domain constants* and denote the set of all domain constants by $D(I_\sigma)$. For a Σ-structure M expanding I_σ, denote by $M^{D(I_\sigma)}$ the structure expanding M to Σ^{I_σ} by interpreting every $\mathbf{d} \in D(I_\sigma)$ by the corresponding domain element d. A formula is in *ground normal form* (GNF) if it contains no quantifiers and all its atomic subformulas are of the form $P(\mathbf{d}_1, \ldots, \mathbf{d}_n)$, $F(\mathbf{d}_1, \ldots, \mathbf{d}_n) = \mathbf{d}$, $c = \mathbf{d}$ or $\mathbf{d}_1 = \mathbf{d}_2$, where P, F and c are respectively a predicate, function and constant symbol of Σ, and $\mathbf{d}_1, \ldots, \mathbf{d}_n, \mathbf{d}$ are domain constants of the appropriate sorts. Observe that a GNF formula is essentially propositional.

A rule is in GNF if its body is in GNF and its head is of the form $P(\mathbf{d}_1, \ldots, \mathbf{d}_n)$, where $\mathbf{d}_1, \ldots, \mathbf{d}_n$ are domain constants.

Definition 2 (Grounding). Let T be a theory over Σ, $\sigma \subseteq \Sigma$ and I_σ a σ-structure. A *grounding for T with respect to I_σ* is a theory T_g over Σ^{I_σ} such that all sentences and rules occurring in T_g are in GNF and for every Σ-structure M expanding I_σ, $M \models T$ iff $M^{D(I_\sigma)} \models T_g$. T_g is called *reduced* if it contains no symbols of σ.

Input and Output Language

In this section, the input and output language of GIDL are described. The input language is called FO$^+$ and is an extension of FO(ID) with partial functions, subsorts, arithmetic and aggregates. The concrete syntax accepted by the system is basically an ASCII version of the input language as described below and can be found in the user manual of the system (Wittocx & Mariën 2008). The manual also describes the output syntax.

Basic Input

The input for GIDL reflects the input of an MX search problem. I.e., it consists of a declaration of an instance vocabulary σ, a sorted expansion vocabulary $\Sigma \setminus \sigma$, a theory T over Σ and a finite σ-structure. These four parts are separated by different headers and can be placed in different files if necessary. GIDL supports full FO(ID), i.e., T can contain arbitrary definitions, the same predicate can be defined in multiple definitions, terms can be nested arbitrarily deep, etc.

The variables occurring in T do not have to be declared. Their associated sort can be specified at the moment they are used in T. Moreover, GIDL contains a sort inference mechanism that derives the sort of a variable automatically if there is one and only one possibility for its sort such that a well-sorted formula is obtained[1].

The declaration of the expansion vocabulary can be split in a set of auxiliary symbols and a set of symbols whose interpretation is relevant to the solution of the problem. This information is passed to the propositional solver, such that it can report to the user only the interpretation of the latter symbols in the models it finds.

FO$^+$

We now describe the extensions of FO(ID) included in FO$^+$.

Partial Functions In standard FO and FO(ID), all functions are total. Besides total functions, one can also declare and use *partial* functions in GIDL. When declaring a partial function, it is possible to specify a domain where it is total.

In general, arbitrary use of partial function symbols creates an ambiguity problem. E.g., consider the formula $P(F(\bar{t}))$, where F is a partial function symbol. This formula can be interpreted in two different ways, as illustrated by the following non-ambiguous rewritings of it:

$$\exists y\ (F(\bar{t}) = y \wedge P(y)) \quad (1)$$

$$\forall y\ (F(\bar{t}) = y \supset P(y)) \quad (2)$$

Here, the atoms $F(\bar{t}) = y$ should be interpreted as $G_F(\bar{t}, y)$, where G_F denotes the graph of F. When F is total, both rewritings are equivalent, but this is not the case when F is partial. Indeed, for an interpretation I such that $\bar{t}^I$ is not in the domain of F^I, $I \not\models$ (1), but $I \models$ (2).

A simple solution to this ambiguity problem is to impose the syntax restriction that a partial function symbol F can only occur in atoms of the form $F(t_1, \ldots, t_n) = t_{n+1}$, where $t_1, \ldots, t_{n+1}$ are terms containing no partial function symbols. For such formulas, there is no ambiguity problem. GIDL does not impose this syntax restriction. Instead, it interprets positive occurrences of atoms $P(F(\bar{t}))$ by (2) and negative occurrences by (1). In other words, it assumes the interpretation where the truth of the sentences in T is maximized, while the truth of the rule bodies is minimized. In case this does not reflect the intended interpretation, a user has to write the sentences and definitions of T in a non-ambiguous form.

Partial functions can be declared by the user but are required also for a logically correct treatment of functions declared over subsorts and of partial arithmetic functions such as $\div$ and mod.

Subsorts In the vocabulary declaration part of an input for GIDL, one can specify that a sort s_1 is a direct subsort of at most one other sort s_2. In that case, the domain $s_1^{I_\sigma}$ of s_1 in the instance structure I_σ has to be a subset of $s_2^{I_\sigma}$. The corresponding hierarchy of sorts must be a collection of trees. The root of a tree in the hierarchy is called a *base sort*. By $\mathrm{base}(s)$, we denote the root of the tree where s occurs, i.e., the base sort above s.

In a context where subsorts are used, a formula is well sorted if the following hold:

- for each term $F(t_1, \ldots, t_n)$ where $\mathfrak{s}(F) = (s_1, \ldots, s_{n+1})$, $\mathrm{base}(s_i) = \mathrm{base}(\mathfrak{s}(t_i))$ for $1 \leq i \leq n$;
- for each atom $P(t_1, \ldots, t_n)$ where $\mathfrak{s}(P) = (s_1, \ldots, s_n)$, $\mathrm{base}(s_i) = \mathrm{base}(\mathfrak{s}(t_i))$ for $1 \leq i \leq n$;
- for each atom $t_1 = t_n$, $\mathrm{base}(\mathfrak{s}(t_1)) = \mathrm{base}(\mathfrak{s}(t_2))$.

A rule with head $P(t_1, \ldots, t_n)$ and $\mathfrak{s}(P) = (s_1, \ldots, s_n)$ is well-sorted if its body is well-sorted and $\mathfrak{s}(t_i) = \mathfrak{s}(s_i)$ for $1 \leq i \leq n$.

A function with sort $(s_1, \ldots, s_{n+1})$ is treated as a partial function whenever one of the input sorts $s_1, \ldots, s_n$ is not a base sort. For an interpretation I and an atom $P(t_1, \ldots, t_n)$ with $\mathfrak{s}(P) = (s_1, \ldots, s_n)$, we define $I \not\models P(t_1, \ldots, t_n)$ if for at least one i, $t_i^I \notin s_i^I$. This fixes the semantics for inputs with subsort declarations.

Whenever a variable x occurs in two positions with a different sort, e.g in $P(x)$ and in $Q(x)$, where $\mathfrak{s}(P) \neq \mathfrak{s}(Q)$, GIDL does not automatically derive a sort for x, as this can lead to unexpected situations. Instead, the user is then forced to declare the sort of the variable.

Arithmetic Besides the vocabulary specified by the user, the instance vocabulary σ of a GIDL input implicitly contains a sort int and the arithmetic functions $+$, $-$, $\cdot$, $\div$, $abs(\cdot)$ and mod. In every instance structure over σ, int is interpreted by the integers $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$, $+$ by addition on $\mathbb{Z}$, $-$ by subtraction, $\cdot$ by multiplication, $\div$ by integer division, abs by the absolute value and mod by the remainder. Note that $\div$ and mod are partial functions on $\mathbb{Z}$ with domain $\mathbb{Z} \setminus \{0\}$. Terms of the form $t_1 + t_2$, $t_1 \cdot t_2$, etc, are of sort int.

To ensure that the grounding produced by GIDL is finite, the use of int is restricted, both in the vocabulary declaration and the theory. In the input and expansion vocabulary declaration, a sort can be declared to be a subsort of int and a variable may have sort int. On the other hand, predicate or function declarations with sort $(\ldots, int, \ldots)$ are not allowed. If in the theory, a variable x of sort int is universally, respectively existentially quantified, it should occur as $\forall x\ (\varphi \supset \ldots)$, respectively $\exists x\ (\varphi \wedge \ldots)$ where φ is a formula for which there exists a finite interval such that $M[x/d] \not\models \varphi$ for any model M of the theory and d outside that interval.

[1] Some of the language extensions described below allow for situations where there is more than one possibility to obtain a well-sorted formula.

We call φ a *bound* for x. GIDL requires that the bounds have a very simple form. E.g., an atom $P(\ldots, x, \ldots)$ is a bound. A formula $t_1 \leq x \leq t_2$ is a bound if t_1, respectively t_2, is a term for which there exists an $n_1 \in \mathbb{Z}$, resp. $n_2 \in \mathbb{Z}$ such that for each model M of the theory $n_1 \leq t_1^M$, respectively $n_2 \geq t_2^n$. Etc. Besides occurrences of bounds φ as $\forall x\ (\varphi \supset \ldots)$ or $\exists x\ (\varphi \wedge \ldots)$, GIDL also accepts syntactically equivalent forms like $\forall x\ (\ldots \vee \neg\varphi \vee \ldots)$ or $\exists x\ (\ldots \wedge \varphi \wedge \ldots)$.

Aggregates Aggregates are functions that have a set as argument. GIDL supports three aggregates: cardinality, sum and product. Concretely, the following are terms with sort int in the input language of GIDL: $card\{\overline{y} \mid \varphi[\overline{y}, \overline{z}]\}$, $sum\{x, \overline{y} \mid \varphi[x, \overline{y}, \overline{z}]\}$ and $prod\{x, \overline{y} \mid \varphi[x, \overline{y}, \overline{z}]\}$. The variables $\overline{z}$ are free in the aggregate term, while x and $\overline{y}$ are local to the term. The sort of x must be a subsort of int. Given an interpretation I, these terms are interpreted by

- $(card\{\overline{y} \mid \varphi[\overline{y}, \overline{z}]\})^I$ is the number of $\overline{d}$ such that $I[\overline{y}/\overline{d}] \models \varphi$;
- $(sum\{x, \overline{y} \mid \varphi[x, \overline{y}, \overline{z}]\})^I = \sum_{I[x/d_x, \overline{y}/\overline{d}_y] \models \varphi} d_x$;
- $(prod\{x, \overline{y} \mid \varphi[x, \overline{y}, \overline{z}]\})^I = \prod_{I[x/d_x, \overline{y}/\overline{d}_y] \models \varphi} d_x$;

Aggregates can be used everywhere in sentences or rule bodies where a term with a subsort of int can occur. The semantics for definitions containing recursion involving aggregates is the one presented in (Pelov, Denecker, & Bruynooghe 2005)

Example 3. In a machine scheduling problem, the constraint that at each timepoint t, the sum of the capacities c of the machines m that are not in maintenance must exceed 100 can be expressed by the sentence $\forall t\ (sum\{c, m \mid Capacity(m) = c \wedge \neg Maintenance(m, t)\} \geq 100)$.

Output Language

The output language of GIDL is an extension with rules and aggregates of the CNF format for SAT solvers and is called *extended CNF* (ECNF). It is the input format for the propositional solvers MIDL (Mariën, Wittocx, & Denecker 2007) and MINISAT(ID) (Mariën *et al.* 2008). Details about the syntax and semantics of the ECNF format is available at `www.cs.kuleuven.be/~dtai/krr/software.html`.

Translation Information

In an ECNF file, each propositional atom has a number, but not a name. In order to construct human readable solutions, GIDL also passes a *translation table* to the propositional solver, defining a mapping from each number that occurs in its ECNF output to a name. To avoid an exhaustive table mapping each number to its corresponding name, first all sort names and their domain elements are listed. Then all predicates with their corresponding sorts are listed, and are assigned a number. An atom $P(d_1, d_2)$ then corresponds to the number $n_P + (i_1 - 1) \cdot |s_2| + (i_2 - 1)$, where $\mathfrak{s}(P) = (s_1, s_2)$, n_P is the number assigned to P, d_1 the i_1th domain element of sort s_1, d_2 the i_2th domain element in s_2 and $|s_2|$ the size of the domain of s_2. The offsets n_P are chosen such that the numbers associated to atoms of different predicates do not overlap.

True and Arbitrary Atoms

Atoms that do not occur in an ECNF file are standard considered to be false by solvers. However, it is often desirable to also leave out the atoms that are discovered to be true in every model and the ones whose truth value can be arbitrarily chosen. GIDL passes a list of left out true and arbitrary atoms to the solver.

System Architecture

Given an input $\langle \Sigma, T, \sigma, I_\sigma \rangle$, GIDL constructs a grounding for T with respect to I_σ in six phases. In this section, a short description of each of the phases is given. The actual grounding algorithm (phase 5) is described in more detail in the next section.

Parser In the first phase, the input $\langle \Sigma, T, \sigma, I_\sigma \rangle$ is parsed. The parser of GIDL is implemented using *flex* and *bison*, which makes it easy to include future extensions of the input language.

Rewrite and Analyze In this phase, T is transformed into an internal normal form: negations are pushed inside until they are directly in front of atoms, $\supset$ is translated in terms of $\neg$ and $\vee$, functions are brought in the form $F(\overline{x}) = y$ and then, these atoms are replaced by $G_F(\overline{x}, y)$, where G_F is a new predicate representing the graph of F. Constraints are added to ensure that each G_F is a graph of a function. Also, all definitions are merged into a single definition Δ.

The dependency graph of Δ is constructed and analyzed to discover which defined predicates do not depend on open expansion predicates. The interpretation of these predicates is the same in every model of T expanding I_σ and can efficiently be computed. Also, a good grounding order for the rules of Δ is computed.

Pre-grounder The pre-grounder calculates the interpretation of the defined predicates that do not depend on open expansion predicates or on aggregates by evaluating their rules. The evaluation algorithm is a generalized version of the semi-naive technique (Ullman 1988) and can handle recursion over negation. The predicates whose interpretation is calculated are from then on considered to be part of the instance structure σ. I_σ is extended by assigning the computed relations to these predicates.

Approximation In this phase, an *approximation* for each subformula in T is computed, using the anytime algorithm described in (Wittocx, Mariën, & Denecker 2008). The computed approximations are used to both reduce grounding size and time.

Formally, an approximation for a formula $\varphi[\overline{x}]$ is a pair of formulas $(\varphi_{ct}[\overline{y}], \varphi_{cf}[\overline{z}])$ over σ such that $\overline{y} \subseteq \overline{x}$, $\overline{z} \subseteq \overline{x}$, $T \models \forall \overline{x}\ (\varphi_{ct} \supset \varphi)$ and $T \models \forall \overline{x}\ (\varphi_{cf} \supset \neg\varphi)$. Intuitively, the formula φ_{ct} provides a lower bound on the set of instances $\varphi[\overline{x}/\overline{\mathbf{d}}]$ of φ that are true in every model of T. The grounding algorithm can then safely replace instances $\varphi[\overline{x}/\overline{\mathbf{d}}]$ in this lower bound by $\top$, leading to a smaller

grounding. Vice versa, φ_{cf} provides a lower bound on the set of $\varphi[\overline{x}/\overline{\mathbf{d}}]$ that are false in every model of T. Instances $\varphi[\overline{x}/\overline{\mathbf{d}}]$ in this lower bound can be replaced by $\bot$. Observe that $(\bot, \bot)$ is an approximation for every formula, called the *trivial approximation*.

Example 2 (Continued). In the Hamiltonian path example, $(\bot, v_2 = Start \vee \neg Edge(v_1, v_2))$ is an approximation for the subformula $In(v_1, v_2)$ and $(Edge(v_1, v_2), \neg Edge(v_1, v_2))$ is an approximation for $Edge(v_1, v_2)$.

The maximal running time of the approximation algorithm, as well as the maximal size of the derived bounds can be specified by the user. Experiments in (Wittocx, Mariën, & Denecker 2008) showed that the default settings work well in most cases.

In the implementation, the approximations are represented and simplified using binary decision diagrams for FO as defined in (Goubault 1995). We extended the simplification algorithm of that paper with rules to cope with arithmetic. Also, parts of approximations that contain no free variables are evaluated out using the instance structure I_σ. This evaluation is the only part of the approximation algorithm that depends on I_σ.

Grounder Using the computed approximations for each subformula, an ECNF theory, equivalent to the input T and I_σ is constructed.

Translate Finally, the translation information and the list of true and arbitrary atoms is written to the output.

Grounding

The actual grounding component in GIDL accomplishes two tasks. It instantiates variables by domain elements and at the same time transforms complex formulas and rules into the ECNF format by applying the Tseitin transformation (Tseitin 1968). In this section, we present the grounding algorithm for the FO part of the input. The algorithm for grounding the rules is similar.

Procedure `Ground` gets as input a formula $\varphi[\overline{x}]$ and outputs a GNF theory, equivalent to the theory containing the single sentence $\bigwedge_{\overline{\mathbf{d}}} \varphi[\overline{x}/\overline{\mathbf{d}}]$. I.e., it outputs a grounding for the sentence $\forall \overline{x}\ \varphi[\overline{x}]$. Here, φ is assumed to be in the internal normal form of GIDL, i.e., the negations are in front of the atoms and function symbols F only occur in atoms of the form $F(\overline{y}) = z$. $(\varphi_{ct}, \varphi_{cf})$ denotes the approximation of φ.

The procedure `output` writes a single ground formula or rule to the output.

The function `getLit` implements the Tseitin transformation. It gets as input a formula $\varphi[\overline{x}]$, outputs a definition Δ_φ in GNF and returns a literal $P_\varphi \in \mathrm{Def}(\Delta_\varphi)$ such that in every model of Δ_φ, the truth value of P_φ equals the truth value of $\bigvee_{\overline{\mathbf{d}}} \varphi[\overline{x}/\overline{\mathbf{d}}]$. Our actual implementation of `getLit` involves some bookkeeping to make sure Δ_φ is written only once, even if `getLit(`φ`)` is called multiple times.

The purpose of line 15 of procedure `Ground`, is to compute all values $\overline{\mathbf{d}}$ such that $\bigvee_i \psi_i[\overline{x}/\overline{\mathbf{d}}]$ is not certainly true, i.e., to compute the answers of the conjunctive formula $\bigwedge_i \neg(\psi_i)_{ct}[\overline{x}]$ in I_σ. GIDL uses the backjumping algorithm of (Leone, Perri, & Scarcello 2004). The original algorithm was designed for computing answers to conjunctions of *literals* but, since the answers of the formulas $\neg(\psi_i)_{ct}[\overline{x}]$ can be easily computed as a table, it is easy to extend the algorithm.

To obtain a grounding of the FO part of T, `Ground` is applied on all sentences of T.

Procedure `ground(`φ`)`

```
1  if φ_cf ≡ ⊤ then
2      output ⊥; return ;
3  if φ_ct ≡ ⊤ then
4      output ⊤; return ;
5  Let x̄ be the free variables of φ;
6  switch φ do
7      case φ is a literal
8          for all d̄ such that I_σ ⊭ φ_ct[x̄/d̄] do
9              if I_σ ⊨ φ_cf[x̄/d̄] then
10                 output ⊥; return ;
11             else output φ[x̄/d̄];
12     case φ ≡ ⋀_{1≤i≤n} ψ_i
13         for 1 ≤ i ≤ n do ground(ψ_i)
14     case φ ≡ ⋁_{1≤i≤n} ψ_i
15         for all d̄ such that I_σ ⊭ ⋁_{1≤i≤n} (ψ_i)_ct[x̄/d̄] do
16             V := ∅;
17             for 1 ≤ i ≤ n do
18                 if I_σ ⊭ (ψ_i)_cf[x̄/d̄] then
19                     add getLit(ψ_i[x̄/d̄]) to V;
20             output ⋁_{L∈V} L;
21     case φ ≡ ∀y ψ
22         ground(ψ);
23     case φ ≡ ∃y ψ
24         for all d̄ such that I_σ ⊭ φ_ct[x̄/d̄] do
25             if I_σ[x̄/d̄] ⊨ φ_cf then
26                 output ⊥; return ;
27             else
28                 V := ∅;
29                 for all d′ such that
                   I_σ ⊭ ψ_cf[x̄/d̄][y/d′] do
30                     if I_σ ⊭ ψ_ct[x̄/d̄][y/d′] then
31                         add getLit(ψ[x̄/d̄][y/d′]) to
                           V;
32                 output ⋁_{L∈V} L;
```

Complexity of `Ground`

When all subformulas of a formula φ are assigned the trivial approximation $(\bot, \bot)$, applying `Ground` to φ consists

```
Function getLit(φ)
1  Let x̄ be the free variables of φ;
2  switch φ do
3      case φ is a literal
4          V := ∅;
5          for all d̄ such that I_σ ⊭ φ_cf[x̄/d̄] do
6              if I_σ ⊭ φ_ct[x̄/d̄] then
7                  add φ[x̄/d̄] to V;
8          if V is a singleton {P} then return P;
9          else
10             Let P be a new propositional atom;
11             output P ← ⋁_{L∈V} L;
12             return P;
13     case φ ≡ ∃y ψ
14         return getLit(ψ);
15     ⋮ // Other cases
```

of simply substituting the variables of φ by all possible domain constants of the appropriate sorts. Hence in this case, computing $\texttt{Ground}(\varphi)$ takes time $\mathcal{O}(\prod_{s\in\Sigma_S}|s|^{n_s})$, where n_s is the number of variables of sort s in φ and $|s|$ the size of the domain of s^{I_σ} of s.

In the case arbitrary approximations are assigned to the subformulas of φ, the result of $\texttt{Ground}(\varphi)$ will become smaller. On the other hand, the worst-case time complexity of computing $\texttt{Ground}(\varphi)$ is then $\mathcal{O}(\prod_{s\in\Sigma_S}|s|^{n_s+d_s})$, where d_s is the number of variables of sort s that occur non-free in an approximation of a subformula of φ. This shows that grounding in the presence of non-trivial approximations may increase the complexity. In practice however, the approximations computed by the algorithm of (Wittocx, Mariën, & Denecker 2008) almost never slow down grounding. Instead, experiments in that paper show that they often lead to a dramatic speed-up.

Example 2 (Continued). Let φ be the formula $\neg In(x,y)$. If the approximation for φ is $(\bot,\bot)$, then $\texttt{Ground}(\varphi,\varphi)$ takes time $\mathcal{O}((Vtx^{I_\sigma})^2)$. If the approximation is $(y = Start \vee \neg Edge(x,y), \bot)$, it takes only time $\mathcal{O}(|Edge^{I_\sigma}|)$.

Related Work

MXidL

A non-native approach to grounding MX(FO(ID)) consists of applying the algorithm presented in (Mariën, Gilis, & Denecker 2004) to transform an MX(FO(ID)) input into an equivalent normal logic program under the well-founded semantics. Then, a (slightly adapted) grounder for Answer Set Programming can be used to ground the logic program. This is the approach taken by MXIDL, the first implemented MX(FO(ID)) grounder. MXIDL supports full many-sorted FO(ID) and arithmetic, but no aggregates, subsorts and partial functions. Experiments with MXIDL were reported on in (Mariën, Wittocx, & Denecker 2006).

MXG

The first native grounding algorithm for MX(FO) and MX(FO(ID)) was described in (Patterson *et al.* 2006; 2007) and works on a table-by-table basis. I.e., to construct a grounding of a sentence φ, it proceeds by taking joins, projections, complements,... of the tables in the instance structure, ending up with a full grounding of φ. The algorithm in GIDL on the other hand, proceeds on a tuple-by-tuple basis. For every variable, it tries all the (relevant) substitutions by domain constants, and it outputs part of the grounding of φ as soon as possible.

An implementation of the grounding algorithm of (Patterson *et al.* 2006) was reported on in (Mitchell *et al.* 2006) and is called MXG. The MXG system implements only part of FO(ID). It allows only for definitions that do not depend on open expansion predicates and that do not involve recursion over negation. It does not support functions, subsorts or arithmetic.

Psgrnd

PSGRND (East *et al.* 2006) is a grounder for the extended logic of propositional schemata (East & Truszczynski 2006a). This logic is a restricted fragment of function-free FO, extended with cardinality aggregates. Also, it has restricted support for inductive definitions: each theory may contain one definition, and all rule bodies must be conjunctions of atoms.

PSGRND keeps the grounding in memory and performs unit propagation each time a clause is added to the grounding. As a post-processing step, it does a limited amount of forward checking on the grounding.

Answer Set Programming

Answer Set Programming (ASP) is a framework for declarative problem solving that is closely related to MX(FO^+). Answer set programs can be transformed into FO^+ theories in a modular way (East & Truszczyński 2006b). Moreover, the *structure* of ASP theories is the same as that of FO^+ theories and there are a lot of similarities between the methodology of modelling in ASP and in MX(FO^+) (Mariën, Gilis, & Denecker 2004; Mariën, Wittocx, & Denecker 2006).

On the other hand, there are several differences in the input languages for GIDL and ASP systems. E.g., GIDL allows for arbitrary FO^+ sentences and definitions, while an answer set program is basically one big definition, in which rule bodies are restricted to conjunctions of literals. An FO sentence φ is modelled in ASP by a rule with an empty head and body $\neg\varphi$, an open predicate can be modelled by defining it with a *choice rule*. Finally, the instance structure of an MX(FO^+) problem corresponds to a series of facts in an answer set program.

ASP systems work by grounding and propositional solving. Three ASP grounders are LPARSE (Syrjänen 1998), GrinGo (Gebser, Schaub, & Thiele 2007) and the grounding component of DLV (Dell'Armi *et al.* 2004). The algorithm in LPARSE works table-by-table, the algorithms in the other two grounders tuple-by-tuple.

Table 1: Impact of approximation (time)

	no approx.	4/4	5/8	6/64
15puzzle	4.89	3.70	3.63	9.00
bounded spanningtree	256.93	10.88	8.41	21.21
clique	1.33	2.36	2.45	###
blocked n-queens	22.44	3.50	3.51	3.51
algebraic groups	7.20	7.38	7.86	###
hamiltonian path	21.37	0.04	0.03	0.19
sokoban	0.49	0.24	0.26	0.31
schur numbers	12.49	0.56	1.32	2.52
sudoku	1.00	0.70	1.08	###

Table 2: Impact of approximation (size)

	no approx.	4/4	5/8	6/64
15puzzle	1461007	1219751	1219751	1219375
bounded spanningtree	8857075	2255522	2255522	2255522
clique	353800	353800	353800	###
blocked n-queens	923822	15822	15822	15822
alg. groups	4001420	3931659	3870081	###
ham. path	8404074	5701	5701	5701
sokoban	95279	74878	74878	74878
schur numbers	64300	62369	51454	47733
sudoku	319795	178828	109267	###

Table 3: Impact of domain atoms

	GIDL	PSGRND	GrinGo	LPARSE	DLV
Ham. circuit (time)	0.54	0.52	17.41	5.25	21.27
Ham. circuit (size)	1.00	1.00	30.80	26.00	24.76

Table 4: MX(FO$^+$) problems (time)

	GIDL	MXG	MXIDL
25-queens	0.16	0.76	0.93
50-queens	1.72	7.90	22.42
75-queens	8.03	33.39	165.71
algebraic groups (size 8)	0.86	3.11	3.27
algebraic groups (size 10)	3.40	11.65	12.12
algebraic groups (size 12)	10.94	34.96	37.48
graph colouring (64980 nodes, 4 colours)	8.82	11.84	###
graph colouring (64980 nodes, 6 colours)	13.32	18.94	###
tower of hanoi (8 discs)	0.87	2.26	145.37
latin square (dim 30)	3.40	9.65	8.28
social golfer (24 players, 6 groups, 8 weeks)	0.47	1.68	1.88

Experiments

In this section, we evaluate the impact of the approximation phase and compare GIDL's performance to other grounders.

All experiments in this section were run on a C2D 3GHz machine with 2GB RAM. All times are in seconds and are averaged over five runs. There was a time-out (###) of 600 seconds for each run. To measure the size of a ground theory, we counted the number of propositional atoms in it.

When comparing to other grounders, we used the standard parameters for the approximation phase of GIDL (see below). The times for GIDL include the time needed for the pre-grounding and the approximation phase. More detailed information, including the used problem encodings, is available at `www.cs.kuleuven.be/~dtai/krr/software/gidl.html`.

Impact of Approximation

Impact of different settings To evaluate the impact of the approximation algorithm, we ran GIDL with different settings. The resulting grounding times and sizes are shown in Tables 1 and 2. In these tables, the first number of the setting is the number of times an approximating formula can be refined. The second number is a measure for the maximum size of the approximating formulas. Increasing these numbers makes the approximation process more expensive and the computed approximating formulas larger and potentially more precise. Due to the increased precision of these formulas, the subsequent grounding phase will produce smaller groundings. This phase may be faster or slower depending on whether the gain due to the smaller grounding dominates the cost of evaluating the larger approximating formulas. The default setting of GIDL is 4/4.

The tables show that the use of approximation yields (often drastically) better times and sizes, even with few refinements and small formula sizes. Only in two cases, grounding without approximation is slightly faster. As for the impact of the size of the parameters, we observe that the most precise setting (6/64) produces a substantially smaller grounding in only one case while it has two time-outs. These and other experiments showed that GIDL's default setting 4/4 provides a good trade-off.

Domain Atoms In general, encoding problems for ASP solvers involves carefully adding (semantically redundant) *domain atoms* to obtain fast grounding times and small grounding sizes. Due to the approximation algorithm, this is not needed when encoding problems for GIDL. Instead, adding redundant domain atoms to GIDL's input rather increases the running time. Because of its unit propagation, the same observation holds for PSGRND. This is illustrated by Table 3 which, in case of a Hamiltonian circuit problem, shows the ratios of the grounding time and size for an encoding without to that for an encoding with redundant domain atoms.

Comparison to MX(FO$^+$) grounders

In this section, we compare GIDL to the other existing grounders for fragments of MX(FO$^+$): MXG (version 0.16)[2] and MXIDL. The ASP grounder used as back-end for MXIDL in the experiments is an adaption of GrinGo (version 0.0.1). The encodings of the problems in the first category are exactly the same for each of the three systems. Most of them were taken from `www.cs.sfu.ca/research/groups/mxp/examples/index.html`. The grounding times are shown in Table 4. GIDL consistently outperforms the other MX(FO$^+$) grounders. Table 5 shows the number of literal instances in the resulting ground files. In general, GIDL produces the smallest groundings, MXIDL the largest ones.

[2]There exists a newer version of MXG, but it was not available at the time of the submission deadline.

Table 5: MX(FO$^+$) problems (size)

	GIDL	MXG	MXIDL
24-queens	50850	50850	52500
50-queens	411700	411700	418125
75-queens	1395050	1395050	1409375
algebraic groups (size 8)	783209	1048119	1054835
algebraic groups (size 10)	3171771	3998889	4014163
algebraic groups (size 12)	9852733	11941763	11971803
graph colouring (64980 nodes, 4 colours)	4141440	2590560	###
graph colouring (64980 nodes, 6 colours)	6991920	4665600	###
tower of hanoi (8 discs)	517570	610904	7152033
latin square (dim 30)	1545113	2430400	2514530
social golfer (24 players, 6 groups, 8 weeks)	366144	534144	510378

Comparison to PSGRND and ASP grounders

In this section, we compare GIDL to PSGRND (7 jul 2005[3]), LPARSE (1.0.17[4]), GrinGo (1.0.0) and DLV (11 oct 2007).

The grounding times of GIDL, PSGRND and the ASP grounders are shown in Table 6, the sizes in Table 7. The problems were chosen such that their encodings cover a wide range of different formulas and language constructs. Because the grounders take different input languages, it is not possible to compare their performance in an entirely objective manner. To nevertheless obtain an as fair as possible comparison, the encodings are similar for the different grounders (i.e., as far as possible, they are straightforward translations of each other) except that domain atoms are added to the ASP encodings where needed to avoid the excessively bad grounding times and sizes mentioned above. Only the encoding of the sokoban puzzle differs considerably among the grounders, because it involves complex statements with alternating quantifiers which are not directly expressible in ASP. For the n-queens instances, we tried the grounders on two different encodings. The first one contains an explicit definition of the concept of a diagonal on a chess board. Due to the use of arithmetic in the second encoding, it could not easily be translated to the input language for DLV.

For each of the problems, GIDL ranks first or second in grounding time. Only on the first version of n-queens and the smallest instance of the Hamiltonian circuit, one of the ASP grounders is (slightly) faster. PSGRND outperforms GIDL on 3 of the seven problems, being at most 7 times faster. It is also faster on the *even/odd* problem but cannot handle large instances of it. On the remaining 3 problems, GIDL outperforms PSGRND. It is at least 30 times faster on the first version of the n-queens problem and on the largest instance of the magic series.

The good results of GIDL and DLV on the first version of n-queens is due to their pre-grounding phase using the semi-naive evaluation technique. The other three grounders use a combination of grounding and unit propagation to compute the diagonals on the chess board. On the *even/odd* problems, the semi-naive evaluation seems less efficient.

[3]We switched the forward checking phase of PSGRND off, as this phase is not incorporated in the other grounders. In all problems selected here, grounding with lookahead leads to slower grounding times.

[4]The newest version of LPARSE (1.1.1) appears to be a lot slower than version 1.0.17 and exhibits the same segmentation faults on our experiments

Table 6: Comparison to PSGRND and ASP grounders (time)

	GIDL	PSGRND	GrinGo	LPARSE	DLV
100-queens-v1	8.83	###	16.67	15.09	9.31
125-queens-v1	19.51	###	35.92	35.58	18.78
150-queens-v1	38.24	###	74.61	71.96	34.37
250-queens-v2	1.38	0.21	1.46	###	-
500-queens-v2	5.66	0.82	6.93	###	-
750-queens-v2	12.78	2.03	19.78	###	-
graph col. (4 colours)	4.72	1.78	9.02	seg. fault	9.12
graph col. (6 colours)	6.14	2.50	12.28	seg. fault	15.05
graph col. (8 colours)	7.56	3.32	15.58	seg. fault	22.62
magic series (size 250)	0.62	7.02	###	###	###
magic series (size 500)	2.49	71.45	###	###	###
magic series (size 750)	5.73	###	###	###	###
Ham. circuit (500 nodes)	0.67	0.80	2.69	0.55	0.83
Ham. circuit (1000 nodes)	2.19	3.99	11.80	2.20	2.53
Ham. circuit (1500 nodes)	4.88	11.11	34.45	6.51	5.67
even/odd (0..10^6)	9.10	3.37	10.17	seg. fault	###
even/odd (0..$2 \cdot 10^6$)	18.96	error	20.34	seg. fault	###
even/odd (0..$3 \cdot 10^6$)	29.06	error	30.23	seg. fault	###
sokoban (20 steps)	3.28	2.44	11.66	8.08	9.39
sokoban (40 steps)	7.00	5.03	23.26	16.29	19.72
sokoban (60 steps)	10.76	7.64	36.04	24.54	30.31

Table 7: Comparison to PSGRND and ASP grounders (size)

	GIDL	PSGRND	GrinGo	LPARSE	DLV
100-queens-v1	1333400	###	1990200	2688100	2050100
125-queens-v1	2604250	###	3890875	5241375	3984500
150-queens-v1	4500100	###	6727800	9047150	6862650
250-queens-v2	252488	250496	250746	###	###
500-queens-v2	1004988	1000996	1001496	###	###
750-queens-v2	2257488	2251496	2252246	###	###
graph col. (4 col.)	1810800	1810800	2069644	###	3630240
graph col. (6 col.)	2716200	2716200	2975046	###	7004880
graph col. (8 col.)	3621600	3621600	3880448	###	11419200
magic series (250)	315005	15939253	###	###	###
magic series (500)	1255005	126253503	###	###	###
magic series (750)	2820005	###	###	###	###
Ham. circ. (500)	97036	128464	121073	124573	192572
Ham. circ. (1000)	242033	320967	302067	309067	481066
Ham. circ. (1500)	483059	641441	603119	613619	961618
even/odd (0..10^6)	0	0	2000003	###	###
even/odd (0..$2 \cdot 10^6$)	0	###	4000003	###	###
even/odd (0..$3 \cdot 10^6$)	0	###	6000003	###	###
sokoban (20 steps)	1940913	1153859	3155238	5679469	3085574
sokoban (40 steps)	4085433	2425099	6430318	11357089	6363194
sokoban (60 steps)	6229953	3696339	9705398	17034709	9640814

The good result of GIDL on the magic series problems stems from its output format, which allows to define a single ground set and use it in multiple aggregate expressions. The other grounders write out a set for each aggregate expression, yielding cubic grounding size instead of GIDL's quadratic size in case of the magic series problem.

For each of the problems, either PSGRND or GIDL has the smallest grounding size. This is due to their rich output languages, enabling compact representations of, e.g., aggregate expressions, and to, respectively, the unit propagation and approximation algorithm. The zero grounding size in the *even/odd* problems stems from the fact that both PSGRND and GIDL write true atoms in the translation information, and not in the actual grounding.

Conclusions

We presented a grounder for an extension of FO, in the context of model expansion. An important contribution of the system is that it supports a very rich input language, extending full FO with ordered sorts, inductive definitions, aggregates, arithmetic and partial functions. The input language

and core algorithm of the grounder were described. Despite its rich language, which makes GIDL the most complete MX grounder of the moment, our experiments show that GIDL is the fastest MX grounder for (extensions of) FO and is more robust and often faster compared to ASP grounders.

Acknowledgments

The input language syntax of GIDL was designed in collaboration with David Mitchell and Eugenia Ternovska. The adapted version of GrinGo was written by Sven Thiele.

References

Dell'Armi, T.; Faber, W.; Ielpa, G.; Leone, N.; Perri, S.; and Pfeifer, G. 2004. System description: Dlv with aggregates. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, 326–330. Springer.

Denecker, M., and Ternovska, E. 2004. A logic of nonmonotone inductive definitions and its modularity properties. In Lifschitz, V., and Niemelä, I., eds., *Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'7)*.

Denecker, M. 2000. Extending classical logic with inductive definitions. In Lloyd et al., J., ed., *First International Conference on Computational Logic (CL'2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, 703–717. Springer.

East, D., and Truszczynski, M. 2006a. Predicate-calculus-based logics for modeling and solving search problems. *ACM Trans. Comput. Log.* 7(1):38–83.

East, D., and Truszczyński, M. 2006b. Predicate-calculus based logics for modeling and solving search problems. *ACM Transactions on Computational Logic (TOCL)* 7(1):38 – 83.

East, D.; Iakhiaev, M.; Mikitiuk, A.; and Truszczynski, M. 2006. Tools for modeling and solving search problems. *AI Commun.* 19(4):301–312.

Enderton, H. B. 1972. *A Mathematical Introduction To Logic*. Academic Press.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, 266–271. Springer.

Goubault, J. 1995. A bdd-based simplification and skolemization procedure. *Logic Journal of IGPL* 3(6):827–855.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *AAAI/IAAI, Vol. 2*, 1194–1201.

Leone, N.; Perri, S.; and Scarcello, F. 2004. Backjumping techniques for rules instantiation in the DLV system. In *NMR*, 258–266.

Marek, V. W., and Truszczyński, M. 1998. Stable models and an alternative logic programming paradigm. *CoRR* cs.LO/9809032.

Mariën, M.; Wittocx, J.; Denecker, M.; and Maurice, B. 2008. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In *Proceedings of the 11th conference on Theory and Applications of Satisfiability Testing, SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*, 211–224. Springer.

Mariën, M.; Gilis, D.; and Denecker, M. 2004. On the relation between ID-Logic and Answer Set Programming. In Alferes, J. J., and Leite, J. A., eds., *JELIA'04*, volume 3229 of *Lecture Notes in Computer Science*, 108–120. Springer.

Mariën, M.; Wittocx, J.; and Denecker, M. 2006. The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*, 19–34.

Mariën, M.; Wittocx, J.; and Denecker, M. 2007. MidL: A SAT(ID) solver. In *4th Workshop on Answer Set Programming: Advances in Theory and Implementation*, 303–308.

Mitchell, D., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *AAAI'05*, 430–435. AAAI Press/MIT Press.

Mitchell, D. G., and Ternovska, E. 2008. Expressive power and abstraction in ESSENCE. *Constraints* 13(3).

Mitchell, D.; Ternovska, E.; Hach, F.; and Mohebali, R. 2006. Model expansion as a framework for modelling and solving search problems. Technical Report TR2006-24, Simon Fraser University.

Patterson, M.; Liu, Y.; Ternvska, E.; and Gupta, A. 2006. Grounding for model expansion in k-guarded formulas. In *Proceedings of 21st IEEE Symposium on Logic in Computer Science (LICS06)*.

Patterson, M.; Liu, Y.; Ternovska, E.; and Gupta, A. 2007. Grounding for model expansion in k-guarded formulas with inductive definitions. In Veloso, M. M., ed., *IJCAI*, 161–166.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2005. Well-founded and stable semantics of logic programs with aggregates. *CoRR* abs/cs/0509024.

Syrjänen, T. 1998. Implementation of local grounding for logic programs with stable model semantics. Technical Report B18, Digital Systems Laboratory, Helsinki University of Technology.

Tseitin, G. S. 1968. On the complexity of derivation in propositional calculus. In Slisenko, A. O., ed., *Studies in Constructive Mathematics and Mathematical Logic II*, volume 8 of *Seminars in Mathematics: Steklov Mathem. Inst.* New York: Consultants Bureau. 115–125.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

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

Wittocx, J., and Mariën, M. 2008. The IDP system. Obtainable via `www.cs.kuleuven.be/~dtai/krr/software.html`.

Wittocx, J.; Mariën, M.; and Denecker, M. 2008. Grounding with bounds. In *AAAI'08*, 572–577. AAAI Press.

Special Session on Preferences

Preferences constitute a natural and effective way of resolving conflicts and choosing best solutions from a large set of candidates. The problem of preference handling and its applications received extensive interest in artificial intelligence, including non-monotonic reasoning, logic programming, constraint programming, planning, and decision making.

Session Chairs

Hans Tompits, Vienna University of Technology, Austria
Kewen Wang, Griffith University, Australia

Program Committee

Ronen I. Brafman, Ben-Gurion University, Israel
Gerhard Brewka, University of Leipzig, Germany
Jan Chomicki, University at Buffalo, USA
James P. Delgrande, Simon Fraser University, Canada
Jon Doyle, North Carolina State University, USA
Judy Goldsmith, University of Kentucky, USA
Katsumi Inoue, National Institute of Informatics, Japan
Ulrich Junker, ILOG, France
Jerome Lang, IRIT, Toulouse, France
Sheila A. McIlraith, University of Toronto, Canada
Francesca Rossi, University of Padova, Italy
Torsten Schaub, University of Potsdam, Germany
Tran Cao Son, New Mexico State University, USA
Toby Walsh, University of New South Wales, Australia

Distributed Defeasible Reasoning in Multi-Context Systems

Antonis Bikakis and **Grigoris Antoniou**
Institute of Computer Science, FO.R.T.H.
Vassilika Vouton, P.O. Box 1385, GR 71110, Heraklion, Greece
{bikakis,antoniou}@ics.forth.gr

Abstract

Multi-Context Systems (MCS) are logical formalizations of distributed context theories connected through a set of mapping rules, which enable information flow between different contexts. Reasoning in MCS introduces many challenges that arise from the heterogeneity of contexts with respect to the language and inference system that they use, and from the potential conflicts that may arise from the interaction of context theories through the mappings. This study proposes a P2P reasoning model for MCS, which represents contexts as peer theories in a P2P system, mapping rules as defeasible rules (rules that can be defeated in the existence of adequate contrary evidence), and uses a preference relation (which, e.g., expresses trust information) to resolve the potential conflicts. It also provides a reasoning algorithm for query evaluation, analyzes its formal properties, and discusses alternative methods for conflict resolution, which differ in the type of information that they use to resolve the conflicts.

Motivation and Background

A Multi-Context System consists of a set of *contexts* and a set of inference rules (known as *mapping* or *bridge* rules) that enable information flow between different contexts. A context can be thought as a logical theory - a set of axioms and inference rules - that models local context knowledge. Different contexts are expected to use different languages and inference systems, and although each context may be locally consistent, global consistency cannot be required or guaranteed. Reasoning with multiple contexts requires performing two types of reasoning; (a) *local reasoning*, based on the individual context theories; and (b) *distributed reasoning*, which combines the consequences of local theories using the mappings. The most critical issues of contextual reasoning are; (a) the *heterogeneity* of contexts with respect to the language and inference system that they use; and (b) the potential conflicts that may arise from the interaction of the different contexts through the mappings. Our study mainly focuses on the second issue, by modeling the different contexts as peers in a P2P system, and performing some type of defeasible reasoning on the distributed peer theories.

The notions of *context* and *contextual reasoning* were first introduced in AI by McCarthy in (McCarthy 1987), as an approach for the problem of *generality*. In the same paper, he argued that the combination of non-monotonic reasoning and contextual reasoning would constitute an adequate solution to this problem. Since then, two main formalizations have been proposed to formalize context: the propositional logic of context (*PLC* (Buvac and Mason 1993; McCarthy and Buvač 1998)) , and the Multi-Context Systems introduced in (Giunchiglia and Serafini 1994), which later became associated with the Local Model Semantics proposed in (Ghidini and Giunchiglia 2001). The second formalism was the basis of two recent studies that were the first to deploy non-monotonic reasoning approaches in MCS: (a) the non-monotonic rule-based MCS framework, which supports default negation in the mapping rules allowing to reason based on the absence of context information, proposed in (Roelofsen and Serafini 2005); and (b) the multi-context variant of Default Logic (Brewka, Roelofsen, and Serafini 2007). The latter models the bridge relations between different contexts as *default rules*, and has the additional advantage that is closer to implementation due to the well-studied relation between Default Logic and Logic Programming. However, the authors do not provide specific reasoning algorithms (e.g. for query evaluation), and their model does not include the notion of priority, which we use for conflict resolution.

Our study also relates to several recent studies that focus on formal models and methods for reasoning in peer data management systems. A key issue in formalizing data-oriented P2P systems is the semantic characterization of *mappings* (bridge rules). One approach, followed in (Bernstein et al. 2002; Halevy et al. 2003), is the first-order logic interpretation of P2P systems. (Calvanese et al. 2004) identified several drawbacks with this approach, regarding modularity, generality and decidability, and proposed new semantics based on epistemic logic. A common problem of both approaches is that they do not model and thus cannot handle inconsistency. Franconi *et al.* in (Franconi et al. 2003) extended the autoepistemic semantics to formalize local inconsistency. The latter approach guarantees that a locally inconsistent database base will not render the entire knowledge base inconsistent. A broader extension, proposed in (Calvanese et al. 2005), is based on non-monotonic epistemic logic, and enables isolating local inconsistency, while also handling peers that may provide mutually inconsistent data. The proposed query evaluation algorithm assumes that all peers share a common alphabet of constants, and does not model *trust* or *priorities* between the peers. The proposi-

tional P2P inference system proposed in (Chatalic, Nguyen, and Rousset 2006) deals with conflicts caused by mutually inconsistent information sources, by detecting them and reasoning without them. The main problem is the same, once again: To perform reasoning, the conflicts are not actually resolved using some external trust or priority information; they are rather isolated.

The reasoning model that we propose represents contexts as peer theories in a P2P system. Specifically, it considers peers that have independent knowledge, and that interact with existing, *neighboring* peers to exchange information. The internal knowledge is expressed in terms of rules, and knowledge is imported from other peers through *mapping rules*. Even if it is assumed that the theory of each peer is locally consistent, the same assumption will not necessarily hold for the global knowledge base. The unification of the local context theories may result in inconsistencies that are caused by the mapping rules. For example, a context theory A may import context knowledge from two different contexts B and C, through two competing mapping rules. In this case, even if the three different contexts are locally consistent, their unification through the mappings defined by A may contain inconsistencies. To deal with with this type of inconsistencies (*global conflicts*), we follow a nonmonotonic approach; mapping rules are expressed as *defeasible rules* (rules that may be defeated in the existence of adequate contrary evidence), and priorities between conflicting rules are determined by the level of trust that each peer has in the other system peers.

The P2P reasoning model captures the three fundamental dimensions of contextual reasoning, as these were formulated in (Benerecetti, Bouquet, and Ghidini 2000), namely *partiality*, *approximation*, and *perspective*:

- *Partiality*. Each peer may not have immediate access to all available information, so a peer theory can be thought as a partial representation of *the world*.
- *Approximation* Each peer theory differs at the level of detail at which a portion of *the world* is represented.
- *Perspective* Each peer theory encodes a different point of view on *the world*.

Furthermore, the P2P paradigm enables us to model:

- Information flow between different contexts as message exchange between the system peers.
- Context changes using the dynamics of a P2P system.
- Confidence on the different context theories as trust between the system peers.

The rest of the paper is structured as follows: First, we formalize the problem. Then, we describe the reasoning algorithm and study its formal properties. Finally, we discuss three alternative approaches for conflict resolution, and conclude with the plans of our future work.

Our Reasoning Approach

Our approach models a multi-context framework as a P2P system P, which is a collection of peer context theories:

$$P = \{P_i\}, i = 1, 2, ..., n$$

Each system peer has a proper distinct vocabulary V_{P_i} and a unique identifier i. Each local theory is a set of rules that contain only local literals (literals from the local vocabulary). These rules are of the form:

$$r_i^l : a_i^1, a_i^2, ...a_i^{n-1} \rightarrow a_i^n$$

where i denotes the peer identifier. Local rules express strict (sound) knowledge and are interpreted in the classical sense: whenever the literals in the body of a local rule $(a_i^1, a_i^2, ...a_i^{n-1})$ derive as consequences of the local theory, then so does the conclusion of the rule (a_i^n). Strict rules with empty body are used to express factual knowledge.

Each peer also defines mappings that associate literals from its own vocabulary (*local literals*) with literals from the vocabulary of other peers (*foreign literals*). The acquaintances of peer P_i, $ACQ(P_i)$ are the set of peers that at least one of P_i's mappings involves at least one of their local literals. Mappings are modeled as defeasible rules (rules that can be defeated in the existence of adequate contrary evidence) of the form::

$$r_i^m : a_i^1, a_j^2, ...a_k^{n-1} \Rightarrow a_i^n$$

The above mapping rule is defined by P_i, and associates some of its own local literals with some of the literals defined by P_j, P_k and other system peers. Literal a_i^n is a local literal of P_i.

Finally, each peer P_i defines a trust level order T_i, which includes a subset of the system peers, and expresses the trust that P_i has in the other system peers. This is of the form:

$$T_i = [P_k, P_l, ..., P_n]$$

A peer P_k is considered more trusted by P_i than peer P_l if P_k precedes P_l in this list. The peers that are not included in T_i are less trusted by P_i than those that are part of the list.

We assume that the context theories are locally consistent, but this is not necessarily true for the global theory, which derives from the unification of local theories and mappings. The inconsistencies result from interactions between local theories and are caused by mappings. To resolve them, we use the available trust information from the system peers.

The $P2P\_DR$ Algorithm

$P2P\_DR$ is a distributed algorithm for query evaluation in Multi-Context Systems following the model that we described in the previous section. The specific reasoning problem that it deals with is: *Given a MCS P, and a query about literal x_i issued to peer P_i, find the truth value of x_i considering P_i's local theory, its mappings and the context theories of the other system peers.* The algorithm may return two different answers: (a) $Ans_{x_i} = Yes$ means that it has computed a positive truth value for x_i, while (b) $Ans_{x_i} = No$ is meant for a negative truth value. The algorithm parameters are:

x_i: the queried literal
P_0: the peer that issues the query
P_i: the local peer
SS_{x_i}: the Supportive Set of x_i (a set of literals that is initially empty)

CS_{x_i}: the Conflicting Set of x_i (a set of literals that is initially empty)

$Hist_{x_i}$: the list of pending queries ($[x_1, ..., x_i]$)

Ans_{x_i}: the answer returned for x_i (initially empty)

The algorithm proceeds in four main steps. In the first step, the algorithm determines if the queried literal, x_i, or its negation $\neg x_i$ are consequences of P_i's local rules. To do that it calls a local reasoning algorithm ($local\_alg$, described later in this section), which returns a positive answer, in case x_i derives from the local rules, or a negative answer in any other case. Below, we denote as $R_s(x_i)$ the set of rules that support x_i (as their conclusion); and as $R_c(x_i)$, the set of rules that contradict x_i (those that support $\neg x_i$)

P2P_DR$(x_i, P_0, P_i, SS_{x_i}, CS_{x_i}, Hist_{x_i}, Ans_{x_i}, T_i)$

if $\exists r_i^l \in R_s(x_i)$ **then**
  $localHist_{x_i} \leftarrow [x_i]$
  call $local\_alg(x_i, localHist_{x_i}, localAns_{x_i})$
  if $localAns_{x_i} = Yes$ **then**
    return $Ans_{x_i} = localAns_{x_i}$ and terminate
  end if
end if
if $\exists r_i^l \in R_c(x_i)$ **then**
  $localHist_{x_i} \leftarrow [x_i]$
  call $local\_alg(\neg x_i, localHist_{x_i}, localAns_{\neg x_i})$
  if $localAns_{\neg x_i} = Yes$ **then**
    return $Ans_{x_i} = \neg localAns_{\neg x_i}$ and terminate
  end if
end if

If Step 1 fails, the algorithm collects, in the second step, the local and mapping rules that support x_i. To check which of these rules can be applied, it checks the truth value of the literals in their body by issuing similar queries (recursive calls of the algorithm) to P_i or to the appropriate neighboring peers $P_j \in ACQ_{P_i}$. To avoid cycles, before each new query, it checks if the same query has been issued before, during the same algorithm call (using $Hist$). For each applicable supportive rule r_i, the algorithm builds its Supportive Set SS_{r_i}. The Supportive Set of a rule derives from the unification of the set of the *foreign literals* (literals that are defined by peers that belong in $ACQ(P_i)$) that are contained in the body of r_i, with the Supportive Sets of the local literals that belong in the body of the same rule. In the end, in case there is no applicable supportive rule ($SR_{x_i} = \{\}$, where SR_{x_i} is the set of applicable rules that support x_i) , the algorithm returns a negative answer for x_i and terminates. Otherwise, it computes the Supportive Set of x_i, SS_{x_i}, as the *strongest* of the Supportive Sets of the applicable rules that support x_i, and proceeds to the next step. The *strongest* Supportive Set is computed using the $Stronger$ function (described later in this section), which applies the preference relation defined by P_i, T_i, on the given sets.

$SR_{x_i} \leftarrow \{\}$
for all $r_i^{lm} \in R_s(x_i)$ **do**
  $SS_{r_i} \leftarrow \{\}$
  for all $b_t \in body(r_i^{lm})$ **do**
    if $b_t \in Hist_{x_i}$ **then**
      stop and check the next rule
    else
      $Hist_{b_t} \leftarrow Hist_{x_i} \cup b_t$
      call $P2P\_DR(b_t, P_i, P_t, SS_{b_t}, CS_{b_t}, Hist_{b_t}, Ans_{b_t}, T_t)$
      if $Ans_{b_t} = No$ **then**
        stop and check the next rule
      else if $Ans_{b_t} = Yes$ *and* $b_t \notin V_i$ **then**
        $SS_{r_i} \leftarrow SS_{r_i} \cup b_t$
      else
        $SS_{r_i} \leftarrow SS_{r_i} \cup SS_{b_t}$
      end if
    end if
  end for
  if $SR_{x_i} = \{\}$ *or* $Stronger(SS_{r_i}, SS_{x_i}, T_i) = SS_{r_i}$ **then**
    $SS_{x_i} \leftarrow SS_{r_i}$
  end if
  $SR_{x_i} \leftarrow SR_{x_i} \cup r_i^{lm}$
end for
if $SR_{x_i} = \{\}$ **then**
  return $Ans_{x_i} = No$ and terminate
end if

In the third step, in the same way with the previous step, the algorithm collects the rules that contradict x_i and builds the conflicting set of x_i (CS_{x_i}). In case there is no applicable rule that contradicts x_i, the algorithm terminates by returning a positive answer for x_i. Otherwise, it proceeds with the last step. Below, we denote as CR_{x_i} the set of the applicable rules that contradict (support the negation of) x_i.

$CR_{x_i} \leftarrow \{\}$
for all $r_i^{lm} \in R_c(x_i)$ **do**
  $SS_{r_i} \leftarrow \{\}$
  for all $b_t \in body(r_i^{lm})$ **do**
    if $b_t \in Hist_{x_i}$ **then**
      stop and check the next rule
    else
      $Hist_{b_t} \leftarrow Hist_{x_i} \cup b_t$
      call $P2P\_DR(b_t, P_i, P_t, SS_{b_t}, CS_{b_t}, Hist_{b_t}, Ans_{b_t}, T_t)$
      if $Ans_{b_t} = No$ **then**
        stop and check the next rule
      else if $Ans_{b_t} = Yes$ *and* $b_t \notin V_i$ **then**
        $SS_{r_i} \leftarrow SS_{r_i} \cup b_t$
      else
        $SS_{r_i} \leftarrow SS_{r_i} \cup SS_{b_t}$
      end if
    end if
  end for
  if $CR_{x_i} = \{\}$ *or* $Stronger(SS_{r_i}, CS_{x_i}, T_i) = SS_{r_i}$ **then**
    $CS_{x_i} \leftarrow SS_{r_i}$
  end if
  $CR_{x_i} \leftarrow CR_{x_i} \cup r_i^{lm}$
end for
if $CR_{x_i} = \{\}$ **then**
  return $Ans_{x_i} = Yes$ and SS_{x_i} and terminate
end if

In the last step, the algorithm compares SS_{x_i} and CS_{x_i} using again the $Stronger$ function. If SS_{x_i} is *stronger*, the algorithm returns a positive answer for x_i. In any other case (including the case that there is not enough trust information available to give priority to one of the competing rules), it returns a negative answer.

if $Stronger(SS_{x_i}, CS_{x_i}, T_i) = SS_{x_i}$ **then**
  return $Ans_{x_i} = Yes$ and SS_{x_i}
else
  return $Ans_{x_i} = No$
end if

The local reasoning algorithm $local\_alg$ is called by $P2P\_DR$ to determine whether a literal is a consequence of the local rules of the theory. The algorithm parameters are:

x_i: the queried literal
$localHist_{x_i}$: the list of pending queries in P_i
$localAns_{x_i}$: the local answer for x_i (initially No)

local_alg$(x_i, localHist_{x_i}, localAns_{x_i})$
for all $r_i^l \in R_s(x_i)$ **do**
  if $body(r_i^l) = \{\}$ **then**
    return $localAns_{x_i} = Yes$ and terminate
  else
    for all $b_i \in body(r_i^l)$ **do**
      if $b_i \in localHist_{x_i}$ **then**
        stop and check the next rule
      else
        $localHist_{b_i} \leftarrow localHist_{x_i} \cup b_i$
        call $local\_alg(b_i, localHist_{b_i}, localAns_{b_i})$
      end if
    end for
    if for every b_i: $localAns_{b_i} = Yes$ **then**
      return $localAns_{x_i} = Yes$ and terminate
    end if
  end if
end for

The $Stronger(S, C, T)$ function is used by $P2P\_DR$ to check which of S and C sets is *stronger*, based on T (the preference relation defined by the peer that the algorithm is called by). According to T, a literal a_k is considered to be *stronger* than a_l if P_k precedes P_l in T. The strength of a set is determined by the the weakest literal in this set.

Stronger(S, C, T)
$a^w \leftarrow a_k \in S$ s.t. *for all* $a_i \in S : P_k$ does not precede P_i in T
$b^w \leftarrow a_l \in C$ s.t. *for all* $b_j \in C : P_l$ does not precede P_j in T
if P_k precedes P_l in T **then**
  $Stronger = S$
else if P_l precedes P_k in T **then**
  $Stronger = C$
else
  $Stronger = None$
end if

Below we demonstrate how the algorithm works through an example. In the system depicted in Figure 1, there are six peers, each one with its local context theory, and a query about x_1 is issued to P_1.

P_1
$r_{11}^l : a_1 \rightarrow x_1$
$r_{12}^m : a_2 \Rightarrow a_1$
$r_{13}^m : a_3, a_4 \Rightarrow \neg a_1$

P_2
$r_{21}^l : c_2 \rightarrow a_2$
$r_{22}^l : b_2 \rightarrow a_2$
$r_{22}^m : b_5 \Rightarrow b_2$
$r_{24}^m : b_6 \Rightarrow b_2$

P_3
$r_{31}^l :\rightarrow a_3$

P_4
$r_{41}^l :\rightarrow a_4$

P_5
$r_{51}^l :\rightarrow b_5$

P_6
$r_{61}^l :\rightarrow b_6$

Figure 1: *A System of Six Context Theories*

Neither x_1 nor $\neg x_1$ derive from P_1's local theory, so the algorithm proceeds to the second step. It successively calls rules r_{11}^l and r_{12}^m, and issues a query about a_2 to P_2. In P_2, a_2 does not derive from the local theory, and the algorithm successively calls the two rules that support a_2; r_{21}^l and r_{22}^l. c_2, which is the only premise of r_{21}^l, is not supported by any rule, so r_{21}^l is not applicable. To check if rule r_{22}^l can be applied, the algorithm must check literal b_2. b_2 is supported by two mapping rules; r_{23}^m and r_{24}^m. To determine if these rules can be applied, the algorithms queries P_5 about b_5, and P_6 about b_6. P_5 and P_6 return positive answers for b_5 and b_6 respectively, as these literals are consequences of their local theories. As there is no rule in P_2 that contradicts b_2 or a_2, P_2 returns a positive answer for b_2 to P_1, and P_1 constructs the Supportive Set of a_1, which contains literal a_2 ($SS_{a_1} = \{a_2\}$). The next step is to check the only rule that contradicts a_1, rule r_{13}^m. Using a similar process, the algorithm ends up with a conflicting set that contains literals a_3 and a_4 ($CS_{a_1} = \{a_3, a_4\}$). To compare SS_{a_1} and CS_{a_2}, the algorithm uses the trust level order defined by P_1, T_1. Assuming that $T_1 = [P_4, P_2, P_6, P_3, P_5]$, a_2 and a_3 are respectively the weakest elements of SS_{a_1} and CS_{a_1}, and a_3 is weaker than a_2. Consequently, P_1 computes a positive answer for a_1, and, as there is no rule that contradicts x_1, it eventually returns a positive answer for x_1.

Properties of the Algorithm

In this section we describe some formal properties of $P2P\_DR$ with respect to its termination (Proposition 1), complexity (Propositions 2-3), and the possibility to create an equivalent unified defeasible theory from the distributed context theories (Theorem 1). The proofs for Propositions 1-3 and Theorem 1 are available in (Bikakis 2008). Proposition 1 holds as cycles are detected within the algorithm.

Proposition 1 *The algorithm is guaranteed to terminate returning either a positive or a negative answer for the queried literal.*

Prop. 2 is a consequence of two states that we retain for each peer, which keep track of the incoming and outgoing queries of the peer.

Proposition 2 *The total number of messages that are exchanged between the system peers for the computation of a*

single query is $O(n^2)$ (in the worst case that all peers have defined mappings with all the other system peers), where n stands for the total number of system peers.

Proposition 3 *The computational complexity of the algorithm on a single peer is in the worst case $O(n^2 \times n_l^2 \times n_r)$, where n stands for the total number of system peers, n_l stands for the number of literals a peer may define, and n_r stands for the total number of (local and mapping) rules that a peer theory may contain.*

Equivalent Unified Defeasible Theory

The goal of the procedure that we describe below is to build a global defeasible theory $T_v(P)$, which produces the same results with $P2P\_DR$. The existence of such theory will enable us to resort to centralized reasoning in cases that we want to avoid decentralized control. The procedure consists of the following steps:

1. The local rules of each peer theory are added as strict rules in $T_v(P)$.
2. The mapping rules of each peer theory are added as defeasible rules in $T_v(P)$.
3. For each pair of conflicting rules in $T_v(P)$, we add a priority relation using the $Priorities_{dl}$ process that we describe below.

The role of $Priorities_{dl}$ is to augment $T_v(P)$ with the required rule priorities considering the trust level orders of the system peers. The process takes as input a literal of the theory, say x_i, the strict and defeasible rules of $T_v(P)$ that support or contradict x_i ($R[x_i]$, $R[\neg x_i]$), and the trust ordering of P_i, T_i, and returns the Supportive Set of x_i (S_{x_i}), and augments $T_v(P)$ with the required priority relations. The algorithm follows three main steps: In the first step, it builds the Supportive Sets for the rules that support or contradict x_i. These sets are built in a similar way that $P2P\_DR$ computes the Supportive Sets of the respective rules in the distributed theories, with only one difference: If there is a literal in the body of a rule that contains w in its Supportive Set, then the algorithm assigns $\{w\}$ as the Supportive Set of the rule, meaning that this rule is inapplicable.

In the second step, $Priorities_{dl}$ collects all the pairs of applicable conflicting rules and adds suitable priority relations by applying the $Stronger$ function on their Supportive Sets (using the trust level order of the peer that defined x_i, T_i).

In the final step, the algorithm computes the Supportive Set of x_i using the following rules: (a) If there is no applicable supportive rule, it returns $\{w\}$; (b) If there is an applicable contradicting rule that is *stronger* than all applicable supportive rules, it returns $\{w\}$; and (c) In any other case it returns the Supportive Set of the strongest applicable rule, using the $Stronger$ function and T_i.

Priorities$_{dl}$(x_i,$R[x_i]$,$R[\neg x_i]$, T_i, S_{x_i})
  $S_{x_i} \rightarrow \{\}$
  for all $r_i \in R[x_i] \cup R[\neg x_i]$ **do**
    $S_{r_i} \leftarrow \{\}$
    for all $a_i \in body(r_i) \cap V_i$ **do**
      call $Priorities_{dl}$(a_i,$R[a_i]$,$R[\neg a_i]$, T_i, S_{a_i})
      $S_{r_i} \leftarrow S_{r_i} \cup S_{a_i}$
    end for
    for all $a_j \in body(r_i) \setminus V_i$ **do**
      call $Priorities_{dl}$(a_j,$R[a_j]$,$R[\neg a_j]$, T_j, S_{a_j})
      if $w \in S_{a_j}$ **then**
        $S_{r_i} \leftarrow \{w\}$
        stop and check next r_i
      else
        $S_{r_i} \leftarrow S_{r_i} \cup a_j$
      end if
    end for
  end for
  for all pairs ($r_i \in R[x_i], s_i \in R[\neg x_i] | \; w \notin S_{r_i}, w \notin S_{s_i}$) **do**
    if $Stronger(S_{r_i}, S_{s_i}, T_i) = S_{r_i}$ **then**
      add $r_i > s_i$ in $T_v(P)$
    else if $Stronger(S_{r_i}, S_{s_i}, T_i) = S_{s_i}$ **then**
      add $s_i > r_i$ in $T_v(P)$
    end if
  end for
  if $\exists r_i \in R[x_i] | S_{r_i} = \{\}$ *or*
  $\forall s_i \in R[\neg x_i]$: $w \in S_{r_i}$ *or* $r_i > s_i \in T_v(P)$ *and*
  $\forall t_i \in R[x_i]$: $Stronger(S_{r_i}, S_{t_i}, T_i) \neq S_{r_i}$ **then**
    return $S_{x_i} = S_{r_i}$
  else
    return $S_{x_i} = \{w\}$
  end if

To prove Theorem 1, which follows, we use the following definition:

Definition 1 *A MCS P is **acyclic** iff there is no rule $r \in P$ such that the conclusion of r may be used to prove a literal in the body of r.*

Theorem 1 *The global defeasible theory $T_v(P)$, augmented with the priority relations derived from the application of $Priorities_{dl}$ on all literals of the theory, produces, under the proof theory of (Antoniou et al. 2001), the same results as the application of $P2P\_DR$ on the distributed context theories of an acyclic MCS P.*

The latter property, which shows the equivalence with a defeasible theory, enables resorting to centralized reasoning by collecting the distributed context theories in a central entity and creating an equivalent defeasible theory. The complexity of $Priorities_{dl}$ used for the derivation of the equivalent global theory is comparable with the complexity of $P2P\_DR$.Via Theorem 1, $P2P\_DR$ has a precise semantic characterization. Defeasible Logic has a proof-theoretic (Antoniou et al. 2001), an argumentation-based (Governatori et al. 2004) and a model-theoretic semantics (Maher 2002).

Alternative Approaches for Conflict Resolution

In the algorithm that we described, each queried peer is required to return a single positive/negative answer for the literal it is queried about. When a conflict arises, a peer uses its trust information, to evaluate the quality of the answers it receives. Each answer is indirectly assigned with the trust value of the peer that returns it. This is one (and rather

the simplest) of the many different approaches that one can think of for conflict resolution. Below, we describe three different approaches, which differ in the type of information that a peer may use to evaluate the quality of the answers that it receives, and to resolve the potential conflicting answers that two or more different peers may return. To clarify the differences, we use the system of peers that we described in Figure 1.

Strict-Weak Answers

In this second approach, we attempt to associate the quality of the answer not only with the trust level of the queried peer, but also with the confidence of the queried peer on the answer it returns. Specifically, we define two levels of quality for each positive answer; (a) the *strict* answers, which derive from the local context theories; and (b) the *weak* answers, which derive from the combination of the local theory with the mappings of the queried peer. In this approach the *strength* of an element in a Supportive Set is determined primarily by the type of answer returned for this element (*strict* answers are considered stronger than *weak* ones), and secondly by the rank of the peer in the trust level order of the querying peer.

In the system of peers depicted in Figure 1, P_3 and P_4 will return a *strict* positive answer for a_3 and a_4 respectively, as the two literals derive directly from their local theories, while for a_2, P_2 will return a *weak* answer, as it cannot be locally proved. So, in this version, $SS_{a_1} = \{a_2\}$ is weaker than $CS_{a_1} = \{a_3, a_4\}$, despite the fact that P_2 precedes P_4 in T_1, and the algorithm will compute a negative answer for a_1, and eventually for x_1 as well.

Propagating Mapping Sets

Another approach is to evaluate the quality of an answer based on the ranks of the peers that are involved in the computation of this answer in the trust order defined by the queried peer. To support this feature, a peer does not return a single positive/negative answer for a literal it is queried about, but augments this answer with the Supportive Set of the queried literal. The peer, which receives the answer, uses this set to evaluate the quality of the answer, but also to build the Supportive Sets of its local literals.

In the system of peers in the example (Figure 1), P_3 and P_4 return a positive answer along with an empty Supportive Set for a_3 and a_4 respectively, as the two literals derive directly from their local theories. In the same way, P_5 and P_6 return positive answers along with empty Supportive Sets for literals b_5 and b_6 respectively to P_2. To build the Supportive Set of b_2, P_2 compares the Supportive Sets of the two rules that support b_2, (r_{23}^m and r_{24}^m). Considering that $T_2 = [P_1, P_5, P_6]$, P_2 computes and returns to P_1, $SS_{b_2} = \{b_5\}$. The algorithm will eventually compute $SS_{a_1} = \{a_2, b_5\}$ and $CS_{a_1} = \{a_3, a_4\}$. Considering that $T_1 = [P_4, P_2, P_6, P_3, P_5]$, b_5 is the weakest element of SS_{a_1} and a_3 is the weakest element of CS_{a_1}, and b_5 is weaker than a_3, so the algorithm will eventually return a negative answer for a_1 and x_1.

Complex Mapping Sets

The main feature of the previous approach is that along with the truth value of the queried literal, a peer also returns its Supportive Set. This set describes the *most trusted* course of reasoning that leads to the computed answer. However, trust is subjective. The *most trusted* between two or more different courses will be different considering the different trust level orderings of two different peers. This last approach has the distinct feature that the *most trusted* course is not determined by the queried peer but by the peer that issues the query. To support this feature, when a peer is queried about one of its local literals, it returns its truth value along with its Supportive Set, which in this case describes all the different ways that can be applied to support this literal. In this version, the Supportive Set of a literal is actually a set of the Supportive Sets of all the rules that can be applied to support this literal.

In the system of peers of the example (Figure 1), in order to build SS_{b_2}, the algorithm will use the Supportive Sets of both rules (r_{23}^m and r_{24}^m) that can be applied to support b_2. So, in this version, P_2 returns $SS_{b_2} = \{\{b_5\}, \{b_6\}\}$, and P_1 computes $SS_{a_1} = \{\{a_2, b_5\}, \{a_2, b_6\}\}$ and $CS_{a_1} = \{\{a_3, a_4\}\}$. From the two different courses that lead to a positive truth value for a_1, as these are described in SS_{a_1}, P_1 computes that the one described in the second set ($\{a_2, b_6\}$) is stronger, as P_6 precedes P_5 in T_1. Comparing this set with $CS_{a_1} = \{\{a_3, a_4\}\}$, the algorithm will determine that CS_{a_1} is weaker, as its weakest element, a_3 is weaker than b_6 according to T_1. Consequently, P_1 will compute and return positive answers for a_1 and eventually for x_1.

Conclusion

In this study, we proposed a model for Multi-Context Systems that represents contexts as local rule theories in a P2P system and mappings as defeasible rules, and uses context and preference information to resolve the potential conflicts that arise from the interaction of contexts through the mappings. We also described a distributed algorithm for query evaluation in MCS and analyzed its formal properties. Finally, we informally presented three alternative strategies for conflict resolution, which differ in the extent of context information that each local theory exploits to resolve potential conflicts. Part of our ongoing and future work includes:

- Implementing the algorithm in Logic Programming, using the equivalence with Defeasible Logic, and the well-studied translation of defeasible knowledge into logic programs under Well-Founded Semantics (Antoniou et al. 2006).
- Studying in more detail the three alternative strategies for conflict resolution that we described in the previous section. Specifically, we plan to study each different version of the algorithm with regard to its properties (termination, complexity, equivalent defeasible theory).
- Adding non-monotonic features in the local context theories to support uncertainty and ambiguity in the local context knowledge.

- Extending the model to support overlapping vocabularies, which will enable different contexts to use elements of common vocabularies (e.g. URIs).
- Studying applications in the Ambient Intelligence and Semantic Web domains, where the theories may represent ontological context knowledge (e.g. in DLP), policies and regulations.

References

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2001. Representation results for defeasible logic. *ACM Transactions on Computational Logic* 2(2):255–287.

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2006. Embedding defeasible logic into logic programming. *Theory Pract. Log. Program.* 6(6):703–735.

Benerecetti, M.; Bouquet, P.; and Ghidini, C. 2000. Contextual reasoning distilled. *JETAI* 12(3):279–305.

Bernstein, P. A.; Giunchiglia, F.; Kementsietsidis, A.; Mylopoulos, J.; Serafini, L.; and Zaihrayeu, I. 2002. Data Management for Peer-to-Peer Computing : A Vision. In *WebDB*, 89–94.

Bikakis, A. 2008. Distributed Reasoning with Conflicts in a Peer-to-Peer Setting. http://www.csd.uoc.gr/∼bikakis/P2PDR.pdf.

Brewka, G.; Roelofsen, F.; and Serafini, L. 2007. Contextual Default Reasoning. In *IJCAI*, 268–273.

Buvac, S., and Mason, I. A. 1993. Propositional Logic of Context. In *AAAI*, 412–419.

Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2004. Logical Foundations of Peer-To-Peer Data Integration. 241–251. ACM.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2005. Inconsistency Tolerance in P2P Data Integration: an Epistemic Logic Approach. In *DBPL-05*, volume 3774 of *LNCS*, 90–105. SV.

Chatalic, P.; Nguyen, G. H.; and Rousset, M.-C. 2006. Reasoning with Inconsistencies in Propositional Peer-to-Peer Inference Systems. In *ECAI*, 352–356.

Franconi, E.; Kuper, G. M.; Lopatenko, A.; and Serafini, L. 2003. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *DBISP2P*, 64–76.

Ghidini, C., and Giunchiglia, F. 2001. Local Models Semantics, or contextual reasoning=locality+compatibility. *Artificial Intelligence* 127(2):221–259.

Giunchiglia, F., and Serafini, L. 1994. Multilanguage hierarchical logics, or: how we can do without modal logics. *Artificial Intelligence* 65(1).

Governatori, G.; Maher, M. J.; Billington, D.; and Antoniou, G. 2004. Argumentation Semantics for Defeasible Logics. *Journal of Logic and Computation* 14(5):675–702.

Halevy, A. Y.; Ives, Z. G.; Suciu, D.; and Tatarinov, I. 2003. Schema Mediation in Peer Data Management Systems. In *ICDE*, 505.

Maher, M. J. 2002. A Model-Theoretic Semantics for Defeasible Logic. In *Paraconsistent Computational Logic*, 67–80.

McCarthy, J., and Buvač, S. 1998. Formalizing Context (Expanded Notes). In Aliseda, A.; van Glabbeek, R.; and Westerståhl, D., eds., *Computing Natural Language*. Stanford, California: CSLI Publications. 13–50.

McCarthy, J. 1987. Generality in Artificial Intelligence. *Communications of the ACM* 30(12):1030–1035.

Roelofsen, F., and Serafini, L. 2005. Minimal and Absent Information in Contexts. In *IJCAI*, 558–563.

Learning preference relations over combinatorial domains

Jérôme Lang and Jérôme Mengin
Institut de Recherche en Informatique de Toulouse
31062 Toulouse Cedex, France*

Abstract

We address the problem of learning preference relations over multi-attribute (or combinatorial) domains. We do so by making hypotheses about the dependence structure between attributes that the preference relation enjoys. The first hypothesis we consider is the simplest one, namely, separability (no dependences between attributes: the preference over the values of each attribute is independent of the values of other attributes); then we consider the more general case where the dependence structure takes the form of an acyclic graph. In all cases, what we want to learn is a set of local preference relations (or equivalently, a CP-net) rather than a fully specified preference relation. We consider three forms of consistency between a CP-net and a set of examples, and for two of them we give an exact characterization in the case of separability, as well as complexity results.

Introduction

In many applications, especially electronic commerce, it is important to learn the preferences of the user on a set of alternatives. Often, the set of alternatives has a combinatorial (or multiattribute) structure, that is, each alternative is a tuple of values for each of a given number of variables (or attributes). For instance, suppose we want to build a recommender system for movies. The available data are the preferences the user has expressed on the movies she has seen, and we would like to predict her preferences on movies she hasn't seen yet (Perny and Zucker 2001; Miller et al. 2003). The most obvious way of doing this is to describe movies using various attributes, such as genre, year, film maker etc. As another example (Viappiani, Faltings, and Pu 2006a; 2006b), one may want to build a system helping a user finding a flat from a large database. A flat is described by attributes such as price, location, size etc. and from the past interactions with the user, the system has to find out what her preferences are so that it can help us finding her ideal flat while minimizing the number of interactions.

There is an important difference between these two examples. The latter is an instance of what is usually called *preference elicitation*: the system interacts with the user by asking her specific requests, until she has found her target object or left the system (Chen and Pu 2004). The former is an instance of *passive learning*: the system has in input the whole set of preferences over films (which has been obtained independently) and has to suggest one (or several) new object(s), without any further interaction.

In both cases, however, the system has to learn preferences of a user (or, sometimes, a class of users) over a set of alternatives that possesses a combinatorial structure. Preferences over combinatorial domains have been investigated in detail by researchers in multiattribute decision theory (starting with Keeney and Raiffa 1976) and in artificial intelligence. Multiattribute decision theory has focused on *modelling* preferences, that is, giving axiomatic characterizations of classes of preference relations or utility functions, while artificial intelligence has focused on designing languages for *representing* preferences that are computationally efficient (they have to express these preferences as succinctly as possible, and to come with algorithms for finding optimal alternatives that are as fast as possible).

Classes of models and languages can be partitioned first according to the mathematical nature of the preferences they consider. Roughly speaking, one distinguishes between *ordinal* preferences (consisting in ranking the alternatives), and *numerical* preferences (consisting of utility functions mapping each alternative to some number). Here we focus on ordinal preferences. They have the advantage of often being easier to obtain from users (as it is well-known that users are ill at ease giving numerical values, except when these values are prices).

A key point, when dealing with ordinal preferences on combinatorial domains, is the dependence structure between attributes. CP-nets (Boutilier et al. 2004) are a graphical language for representing preferences that is based on conditional preferential independence (Keeney and Raiffa 1976). A CP-net is composed of a directed graph representing the preferential dependences between variables, and a set of conditional tables (one for each variable), expressing, for each variable, the local preference on the values of its domain given the possible combination of values of its parents. The transitive closure of these local preferences is a partial order over the set of alternatives, which can be extended into several total orders. This is one of the most popular preference representation languages, and many facets of CP-nets

*Acknowledgements: many thanks to Richard Booth, Kevin Garcia, Peter Haddawy, Mathieu Serrurier and Chattrakul Sombatheera for lots of helpful discussions on this topic. Anonymous referees also gave helpful comments.

have been studied, such as consistency, dominance checking, and optimization (constrained and unconstrained). One missing brick is the *learning* of CP-nets from a user.

Whereas learning or eliciting numerical preferences over multiattribute domains has been considered in some places (e.g. Ha and Haddawy 1997, Guo, Müller, and Weinhardt 2003), there have been few attempts at learning ordinal preferences. Athienitou and Dimopoulos (2007) propose algorithms to learn CP-nets; however, their approach suffers from an important drawback: it searches for a CP-net, whose associated partial order contains all the examples – we say that it *entails* the examples, – while we argue that what we intuitively look for is a CP-net whose associated partial order can be extended into at least one total order that contains the examples – we then say that the CP-net is *consistent* with them. To see this, consider the following example.

Example 1 *Suppose we have two binary attributes X_1 and X_2 (with domains $\{x_1, \overline{x_1}\}$ and $\{x_2, \overline{x_2}\}$ respectively), and the set of examples $\mathcal{E} = \{x_1x_2 \succ x_1\overline{x_2},\ x_1\overline{x_2} \succ \overline{x_1}x_2,\ \overline{x_1}x_2 \succ \overline{x_1}\overline{x_2}\}$. The transitive closure of $\mathcal{E}$ is the complete preference relation $x_1x_2 \succ x_1\overline{x_2} \succ \overline{x_1}x_2 \succ \overline{x_1}\overline{x_2}$. This preference relation is* separable*, which means that the agent's preferences over the values of one attribute do not depend on the value of the other attributes: here, the agent unconditionally prefers x_1 to $\overline{x_1}$ and x_2 to $\overline{x_2}$. The fact that $x_1\overline{x_2}$ is preferred to $\overline{x_1}x_2$ simply means that when asked to choose between X_1 and X_2, the agent prefers to give up X_2 (think of X_1 meaning "getting rich" and X_2 meaning "beautiful weather tomorrow").*

What do we expect to learn from the above set of examples $\mathcal{E}$? Intuitively, since $\mathcal{E}$ is a separable preference relation, we expect to output a CP-net $\mathcal{N}$ with an empty graph and the two unconditional preference tables $x_1 \succ \overline{x_1}$ and $x_2 \succ \overline{x_2}$. However, no CP-net implies $\mathcal{E}$, whatever its dependence graph. The CP-net $\mathcal{N}$ induces a *partial* preference relation in which $x_1\overline{x_2}$ and $\overline{x_1}x_2$ are incomparable; and more generally, no CP-net can "explain" that $x_1 \succ \overline{x_1}$ is "directly preferred" to $x_2 \succ \overline{x_2}$ (i.e., with no intermediate alternative). Therefore, if we look for a CP-net *implying* each of the examples, we will simply output 'failure". On the other hand, if we look for a CP-net that is simply *consistent* with the examples we will output the above CP-net.

The explanation is that when an agent expresses a CP-net, the preference relation induced by this CP-net *is not meant to be the whole agent's preference relation, but a subset (or a lower approximation) of it*. In other terms, when an agent expresses the CP-net $\mathcal{N}$, she simply expresses that she prefers x_1 to $\overline{x_1}$ *ceteris paribus* (i.e., for a fixed value of X_2) and similarly for the preference $x_2 \succ \overline{x_1}$; the fact that $x_1\overline{x_2}$ and $\overline{x_1}x_2$ are incomparable in $\mathcal{N}$ surely does not mean that the user really sees them incomparable, but, more technically, that CP-nets are not expressive enough for representing the missing preference $x_1\overline{x_2} \succ \overline{x_1}x_2$[1].

Another way of explaining this difference is that there are two ways of seeing a CP-net: either we identify it with its induced partial preference relation, or we identify it with the set of all complete preference relations that extend this partial preference relation. With the second view, the goal of the learning algorithm is to learn a collection of tables such that the examples are consistent with some preference relation in this set.

Another work on learning ordinal preferences is that of Sachdev (2007): he proposes algorithms to learn preference theories in the sense of Doyle, Shoham, and Wellman (1991). The preference theory he obtains is indeed consistent with the set of examples. In this work however, Sachdev considers that one is given (or has access to) the complete set of examples corresponding to an existing preference theory that one is trying to induce. We do not make this strong assumption here.

In the next section, we give some background on preferences on combinatorial domains, and then we introduce three kinds of compatibility between a CP-net and a set of examples, namely, *weak compatibility*, *strong compatibility* and *implicative compatibility*. We then focus on the simplest case, namely separable preference relations (which extend CP-nets with no preferential dependences): we show how weak compatibility can be reduced to a satisfiability problem and *vice versa*, which allows us to show that deciding weak compatibility is NP-complete ; we also give a way of deciding implicative consistency. We finally go further and show how to extend these results to the situation where the graphical component of the CP-net is fixed but can contain dependencies. In the last section, we conclude with some hints about the general case where we have to learn both the graph *and* the tables.

The learning problem

Multiattribute domains

We assume that we have a finite set $\mathcal{V} = \{X_1, \ldots, X_n\}$ of *attributes* with associated finite *domains* $D_1, \ldots, D_n$. $D = D_1 \times \ldots \times D_n$ is the set of all complete assignments, called *outcomes*.

For any nonempty subset $\mathcal{X}$ of $\mathcal{V}$, we let $D_{\mathcal{X}} = \times_{X_i \in \mathcal{X}} D_i$. Elements of $D_{\mathcal{X}}$ are called $\mathcal{X}$-assignments (i.e. value assignments for all attributes in X); they are denoted using vectorial notation, *e.g.*, $\vec{x}$. For any disjoint subsets $\mathcal{X}$ and $\mathcal{Y}$ of $\mathcal{V}$, $\vec{x} \in D_{\mathcal{X}}$ and $\vec{y} \in D_{\mathcal{Y}}$ then the concatenation of $\vec{x}$ and $\vec{y}$, $\vec{x}\vec{y}$, is formally defined as $\vec{x} \cup \vec{y}$ – i.e., it is the $X \cup Y$-assignment which assigns to attributes in X (resp. Y) the value assigned by $\vec{x}$ (resp. $\vec{y}$).

An attribute X_i is *binary* if D_i has two elements, which by convention we note x_i and $\overline{x_i}$.

A *preference relation* on a multiattribute domain D is a weak order on D, that is, a reflexive and transitive binary relation $\succeq$. If furthermore $\succeq$ is connected, that is, if for every $\vec{x}, \vec{y} \in D$ we have either $\vec{x} \succeq \vec{y}$ or $\vec{y} \succeq \vec{x}$ then $\succeq$ is a *complete* preference relation. A *strict preference relation* $\succ$ is an order on D, that is, an irreflexive and transitive (thus asymmetric) binary relation. If moreover $\succ$ is connected then $\succ$ is a *linear preference relation*. From a preference relation $\succeq$ we define a strict preference relation in the usual way: $\vec{x} \succ \vec{y}$ iff $\vec{x} \succeq \vec{y}$ and not ($\vec{y} \succeq \vec{x}$).

[1] If we want to do this, we have to resort to a more expressive language such as TCP-nets (Brafman and Domshlak 2002) or conditional preference theories (Wilson 2004).

Suppose now that we have a set of *examples* $\mathcal{E}$, where each example is a pair of distinct outcomes $(\vec{x}, \vec{y})$ (also denoted, equivalently, by $\vec{x} \succ \vec{y}$) such that $\vec{x}$ is preferred to $\vec{y}$. We sometimes denote such an example by $\vec{x} \succ \vec{y}$ rather than $(\vec{x}, \vec{y})$. In the following, $\mathcal{E}$ denotes a finite set of examples. In a recommender system for examples, these example may have been recorded in the course of a user interaction with the system.

Ideally, we would like to induce from these examples the complete ordering of all outcomes for the user. That is, we would like to learn how to order every unordered pair of distinct outcomes $\{\vec{x}, \vec{y}\}$. Therefore, our target is a linear preference, or else, a complete preference relation (if we allow for indifferences)

However, if we call $\mathcal{O}$ the set of total strict orders on D, and if the n attributes have m possible values each, there are m^n outcomes, thus $m^n!$ total strict orders in $\mathcal{O}$. There are too many elements in $\mathcal{O}$ to represent them efficiently, so we will have to restrict ourselves to a smaller hypothesis space. In the next section, we briefly present CP-nets, which will form our hypothesis space.

Ceteris paribus preferences and CP-nets

Preferences between outcomes that differ in the value of one attribute only, all other attributes being equal (or *ceteris paribus*) are often easy to assert, and to understand. CP-nets (Boutilier et al. 2004) are a graphical language for representing such preferences, that is based on conditional preferential independence (Keeney and Raiffa 1976). A CP-net is composed of a directed graph representing the preferential dependences between attributes, and a set of conditional preference tables (one for each attribute), expressing, for each attribute, the local preference on the values of its domain given the possible combination of values of its parents. This is one of the most popular preference representation languages on multiattribute domains, and many facets of CP-nets have been studied, such as consistency, dominance checking, and optimization (constrained and unconstrained).

Let us call a *swap* any pair of outcomes $\{\vec{x}, \vec{y}\}$ that differ in the value of one attribute only, and let us then call *swapped attribute* the attribute that has different values in $\vec{x}$ and $\vec{y}$. A CP-net specifies, for every swap $\{\vec{x}, \vec{y}\}$, which of $\vec{x} \succ \vec{y}$ or $\vec{y} \succ \vec{x}$ is true. This can be achieved in a compact manner when there are many *independencies* among attributes.

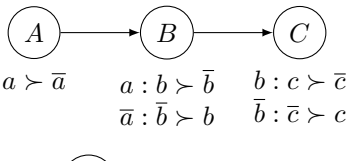
Example 2 *Consider three attributes A, B and C, with respective domains $\{a, \overline{a}\}$, $\{b, \overline{b}\}$ and $\{c, \overline{c}\}$, and suppose that the four swaps on B are ordered as follows: $abc \succ a\overline{b}c$, $\overline{a}\overline{b}c \succ \overline{a}bc$, $ab\overline{c} \succ a\overline{b}\overline{c}$, $\overline{a}\overline{b}\overline{c} \succ \overline{a}b\overline{c}$. We can see that, irrespective of the value of C, if a is the case, then b is preferred to $\overline{b}$, whereas if $\overline{a}$ is the case, then $\overline{b}$ is preferred to b. We can represent this ordering on the B-swaps with two conditional preferences: $a : b \succ \overline{b}$ and $\overline{a} : \overline{b} \succ b$, and say that, given A, B is (conditionally) preferentially independent of C.*

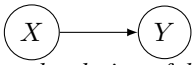
Definition 1 *Let $\{\mathcal{X}, \mathcal{Y}, \mathcal{Z}\}$ be a partition of the set $\mathcal{V}$ and $\succ$ a linear preference relation over D. $\mathcal{X}$ is* (conditionally) preferentially independent *of $\mathcal{Y}$ given $\mathcal{Z}$ (w.r.t. $\succ$) if and only if for all $\vec{x}_1, \vec{x}_2 \in D_{\mathcal{X}}$, $\vec{y}_1, \vec{y}_2 \in D_{\mathcal{Y}}$, $\vec{z} \in D_{\mathcal{Z}}$,*

$$\vec{x}_1\vec{y}_1\vec{z} \succ \vec{x}_2\vec{y}_1\vec{z} \text{ iff } \vec{x}_1\vec{y}_2\vec{z} \succ \vec{x}_2\vec{y}_2\vec{z}$$

Definition 2 *A* CP-net *over attributes $V = \{X_1, \ldots, X_n\}$ with domains $D_1, \ldots, D_n$ is a pair $\mathcal{N} = \langle G, P \rangle$ where G is a directed graph over $x_1, \ldots, x_n$ and P is a set of conditional preference tables $CPT(X_i)$ for each $X_i \in V$. For attribute X_i, we denote by $\mathsf{Par}(X_i)$ (resp. $\mathsf{NonPar}(X_i)$) the set of parents of X_i in G (resp. $\mathcal{V} - (\{X_i\} \cup \mathsf{Par}(X_i))$). Each conditional preference table is a list of rows of the form $\vec{u} : x_i^j \succ x_i^k$: it associates a total order on D_i with each instantiation $\vec{u}$ of $\mathsf{Par}(X_i)$, and indicates, for every possible instantiation $\vec{z}$ of $\mathsf{NonPar}(X_i)$, that $\vec{u}x_i^j\vec{z} \succ \vec{u}x_i^k\vec{z}$. When all attributes of $\mathcal{V}$ are binary, a CP-net over $\mathcal{V}$ is said to be* propositional.

Example 3 *A CP-net over attributes A, B and C, with respective domains $\{a, \overline{a}\}$, $\{b, \overline{b}\}$ and $\{c, \overline{c}\}$ is:*



where  *means "X is a parent of Y". The associated ordering of the swaps is:*

where $\vec{x} \longrightarrow \vec{y}$ means "$\vec{x}$ is preferred to $\vec{y}$".

Although a CP-net only specifies an ordering of all swaps, we are naturally interested in the transitive closure of this ordering; for a CP-net $\mathcal{N}$, we note $\succ_{\mathcal{N}}$ this transitive closure. Note that this relation $\succ_{\mathcal{N}}$ may not be total, and it may not be a strict order since it may contain cycles, and thus not be irreflexive. We know from (Boutilier et al. 2004) that if G is acyclic, then $\succ_{\mathcal{N}}$ is a strict order (i.e. contains no cycles). In this case we say that $\mathcal{N}$ is consistent; $\succ_{\mathcal{N}}$ may still not be total, it can then be completed in a number of total strict orders of $\mathcal{O}$. If $\succ$ is one of them, that is, if $\succ \in \mathcal{O}$ and if $\succ_{\mathcal{N}} \subseteq \succ$, we say that $\succ$ is a *completion* of $\mathcal{N}$. When $\succ_{\mathcal{N}}$ is not irreflexive, we say that $\mathcal{N}$ is *inconsistent*.

We recall the following property (Boutilier et al. 2004), which will be useful later:

Proposition 1 *(Boutilier et al. 2004, Th. 7 and 8) Let $\mathcal{N}$ be an acyclic CP-net and $\vec{x}$, $\vec{y}$ two outcomes. Then $\vec{x} \succ \vec{y}$ is implied by $\mathcal{N}$ if and only if there is a sequence of swaps $\{\vec{x}^0, \vec{x}^1\}, \{\vec{x}^1, \vec{x}^2\}, \ldots, \{\vec{x}^{k-1}, \vec{x}^k\}$ such that $\vec{x}^0 = \vec{x}$, $\vec{x}^k = \vec{y}$, and for every $0 \leq i < k$, $\vec{x}^i \succ \vec{x}^{i+1}$, that is, if X_{j_i} is the attribute swapped between $\vec{x}^i$ and $\vec{x}^{i+1}$, and if $\vec{u}$ is the vector of values commonly assigned by $\vec{x}$ and $\vec{y}$ to the parents of X_i, then $\mathcal{N}$ contains $\vec{u} : x_{j_i}^i \succ x_{j_i}^{i+1}$;*

Different forms of compatibility between sets of examples and CP-nets

We said earlier that the target of our learning process should be a strict total order[2] over all outcomes, but we have just seen that a CP-net $\mathcal{N}$ does not in general correspond to such an order, since the preference relation $\succ_{\mathcal{N}}$ induced from $\mathcal{N}$ is generally not complete. Actually, $\succ_{\mathcal{N}}$ can be seen as the set of all its completions, that is, a CP-net expresses a *set of linear preference relations*.

If an example $(\vec{x}, \vec{y})$ is a swap, then either $\vec{x} \succ_{\mathcal{N}} \vec{y}$ or $\vec{y} \succ_{\mathcal{N}} \vec{x}$, and clearly we would like $\mathcal{N}$ to be in agreement with the example, that is, for instance, such that $\vec{x} \succ_{\mathcal{N}} \vec{y}$. But if $(\vec{x}, \vec{y})$ is not a swap, there may be completions $\succ$ and $\succ'$ of $\succ_{\mathcal{N}}$ such that $\vec{x} \succ \vec{y}$ and $\vec{y} \succ' \vec{x}$.

So we should start by discussing the possible ways of measuring to which extent a given CP-net generalizes from a given set of examples.

Definition 3 *Let $\mathcal{N}$ be a CP-net over $\mathcal{V}$. An example $(\vec{x}, \vec{y})$ is*

- *implied by $\mathcal{N}$ if $\vec{x} \succ \vec{y}$ for every completion $\succ$ of $\succ_{\mathcal{N}}$;*
- *consistent with $\mathcal{N}$ if there is a completion $\succ$ of $\succ_{\mathcal{N}}$ such that $\vec{x} \succ \vec{y}$.*

Furthermore, we will say that a set of examples $\mathcal{E}$ is:

- *implied by $\mathcal{N}$ if for every completion $\succ$ of $\succ_{\mathcal{N}}$, for every $(\vec{x}, \vec{y}) \in \mathcal{E}$, $\vec{x} \succ \vec{y}$*
  (that is, if every example is implied by $\mathcal{N}$);
- *globally (or strongly) consistent with $\mathcal{N}$ if there is a completion $\succ$ of $\succ_{\mathcal{N}}$ such that, for every $(\vec{x}, \vec{y}) \in \mathcal{E}$, $\vec{x} \succ \vec{y}$.*
- *weakly consistent with $\mathcal{N}$ if for every $(\vec{x}, \vec{y}) \in \mathcal{E}$, there is a completion $\succ$ of $\succ_{\mathcal{N}}$ such that, $\vec{x} \succ \vec{y}$*
  (that is, if every example $(\vec{x}, \vec{y}) \in \mathcal{E}$ is individually consistent with $\mathcal{N}$).

Clearly, strong consistency implies weak consistency, and, if $\mathcal{N}$ is consistent and if $\mathcal{E}$ is implied by $\mathcal{N}$, then it is strongly consistent with $\mathcal{N}$.

Notice that an example $(\vec{x}, \vec{y}) \in \mathcal{E}$ is implied by $\mathcal{N}$ if and only if $\vec{x} \succ_{\mathcal{N}} \vec{y}$. Also, it can be shown easily that $\mathcal{E}$ is strongly consistent with $\mathcal{N}$ if and only if the transitive closure of $\succ_{\mathcal{N}} \cup \mathcal{E}$ contains no cycle.

(Athienitou and Dimopoulos 2007) try to learn a CP-net that implies a given set of examples. However, there are cases where learning such a CP-net may not be appropriate.

We have (cf. Ex. 1) that the transitive closure of a set of examples may be a total order over the set of alternatives, that the preferences specified by this set of examples may be separable, although they cannot be implied by any CP-net. Consider now a second example.

Example 4 $\mathcal{E} = \{x_1x_2 \succ x_1\overline{x_2},\ x_1\overline{x_2} \succ \overline{x_1}x_2\}$. *This set of examples is obtained from that of Example 1 by removing the third example. Intuitively, there is no reason to think that the agent's preference relation is not separable. However, there exists no CP-net with $G = \langle \mathcal{V}, \emptyset \rangle$ that implies $\mathcal{E}$. When allowing for the more complicated structure $G = \langle \mathcal{V}, \{X_1 \to X_2\} \rangle$ we obtain the conditional preference tables $x_1 \succ \overline{x_1}$, $x_1 : x_2 \succ \overline{x_2}$, $\overline{x_1} : \overline{x_2} \succ x_2$, whose induced preference relation is $x_1x_2 \succ x_1\overline{x_2} \succ \overline{x_1x_2} \succ \overline{x_1}x_2$. Therefore, asking for a CP-net that* implies *$\mathcal{E}$ leads to a more complicated structure than necessary (a preferential dependence between X_1 and X_2 whereas nothing tells us there should be one).*

In both Examples 1 and 4, $\mathcal{E}$ is strongly (and *a fortiori* weakly) consistent with the separable CP-net $\{x_1 \succ \overline{x_1},\ x_2 \succ \overline{x_2}\}$. Therefore weak and strong compatibility look like more reasonable notions when it comes to learn CP-nets.

The difference between weak and strong compatibility (that we could also call local and global compatibility) is more subtle. While strong compatibility requires that the examples are all compatible with a *single* preference relation extending $\mathcal{N}$, weak compatibility only requires each example to be compatible with *some* preference relation extending $\mathcal{N}$. As a consequence, if $\mathcal{E}$ is inconsistent (for instance because it contains two opposite examples $\vec{x} \succ \vec{y}$ and $\vec{y} \succ \vec{x}$, or more generally a series of examples $\vec{x}_1 \succ \vec{x}_2$, $\vec{x}_2 \succ \vec{x}_3$, $\ldots$, $\vec{x}_p \succ \vec{x}_1$), then there cannot be an $\mathcal{N}$ such that $\mathcal{E}$ is strongly consistent with $\mathcal{N}$, whereas it might still be the case that there is an $\mathcal{N}$ such that $\mathcal{E}$ is weakly consistent with $\mathcal{N}$, as it can be seen on the following example.

Example 5 $\mathcal{E} = \{\overline{x_1}x_2 \succ x_1\overline{x_2},\ x_1\overline{x_2} \succ \overline{x_1}x_2\}$. *$\mathcal{E}$ is clearly inconsistent, and yet $\mathcal{E}$ is weakly consistent with the separable CP-net $\mathcal{N}$ whose tables are $\{x_1 \succ \overline{x_1}, x_2 \succ \overline{x_2}\}$. Because $\mathcal{E}$ is inconsistent, it is not strongly consistent with $\mathcal{N}$ (nor with any other CP-net).*

Note that if $\mathcal{N}$ is itself inconsistent (i.e., possesses cycles), then no set of examples can be strongly consistent with $\mathcal{N}$ whereas there are sets of examples which are implied by $\mathcal{N}$ (and, a fortiori, are weakly consistent with $\mathcal{N}$).

If the examples all come from a single user and are reliable, then weak consistency is much too weak. However, if they come from multiple users (given that we want to learn the generic preferences of a group of users), or a single user in different contexts, then it becomes reasonable: for instance, we may learn that all users in the group unconditionally prefer x_1 to $\overline{x_1}$ and x_2 to $\overline{x_2}$, whereas their preferences between $x_1\overline{x_2}$ and $\overline{x_1}x_2$ may differ (think as x_1 and x_2 as, respectively, "being invited to a fine dinner" and "receiving a \$50 award"). Moreover, weak consistency is relevant even for a single user if we allow for errors or for changes of mind.

In the sequel, we will focus on weak consistency, because it is easier to characterize. The computation of strong consistency is left for future research.

Definition 4 *Let $G = \langle \mathcal{V}, E \rangle$ be a graph over $\mathcal{V}$ and $\mathcal{E}$ a set of examples. $\mathcal{E}$ is*

- weakly G-compatible *if there there exists a CP-net $\mathcal{N}$ with graph G such that $\mathcal{E}$ is weakly consistent with $\mathcal{N}$.*

[2] Our methodology and results would easily carry on to the problem of learning nonstrict preference relations (where indifference is allowed). We stick here to strict preference relation because the presentation is simpler.

- strongly G-compatible *if there there exists a CP-net $\mathcal{N}$ with graph G such that $\mathcal{E}$ is strongly consistent with $\mathcal{N}$.*
- implicatively G-compatible *if there there exists a CP-net $\mathcal{N}$ with graph G such that $\mathcal{E}$ is implied by $\mathcal{N}$.*

Observation 1 *If G is acyclic then implicative G-compatibility implies strong G-compatibility, and strong G-compatibility implies weak G-compatibility.*

This is just because if G is acyclic then any CP-net $\mathcal{N}$ whose associated graph is G is consistent. We also have the obvious fact:

Observation 2 *If $G \subseteq G'$ and $\mathcal{E}$ is weakly (resp. strongly, implicatively) G'-compatible, then $\mathcal{E}$ is weakly (resp. strongly, implicatively) G-compatible.*

As usual in machine learning, we have a preference for learning *simple* structures: here, simplicity is measured by the size of the tables, which is directly related to the number of edges in the graph. In particular, the simplest CP-nets are the separable ones (i.e., those without edges); next Section is dedicated to this specific class of CP-nets.

Learning separable preference relations

Computing a CP-net weakly consistent with a set of examples

We start by the simplest case of separable preference relations over binary domains, that is:

- $G = \emptyset$;
- $D = \{x_1, \overline{x_1}\} \times \ldots\ldots \{x_n, \overline{x_n}\}$.

We first define the following translation from sets of examples $\mathcal{E}$ to sets of clauses. Let $\vec{x} \succ \vec{y}$ be an example. Define $Diff(\vec{x}, \vec{y}) = \{x_i \mid (\vec{x})_i = x_i \text{ and } (\vec{y})_i = \overline{x_i}\} \cup \{\overline{x_i} \mid (\vec{x})_i = \overline{x_i} \text{ and } (\vec{y})_i = x_i\}$.

Now, with each example $\vec{x} \succ \vec{y}$ we associate the following clause $C_{\vec{x}\succ\vec{y}}$ that contains x_i iff $x_i \in Diff(\vec{x}, \vec{y})$ and $\neg x_i$ iff $\overline{x_i} \in Diff(\vec{x}, \vec{y})$.

For instance, if $\vec{x} = x_1\overline{x_2}x_3x_4$ and $\vec{y} = \overline{x_1}x_2x_3\overline{x_4}$ then $Diff(\vec{x}, \vec{y}) = \{x_1, \overline{x_2}, x_4\}$ and $C_{\vec{x}\succ\vec{y}} = x_1 \vee \neg x_2 \vee x_4$.

If $\mathcal{E}$ is a set of examples then $\Phi_{\mathcal{E}}$ is the set of clauses defined by $\Phi_{\mathcal{E}} = \{C_e \mid e \in \mathcal{E}\}$.

Lastly, we define the following one-to-one correspondence between truth assignments over $\{x_1, \ldots, x_n\}$ and separable CP-nets over $\mathcal{V}$. If M is such a truth assignment, then the $\mathcal{N}_M$ contains the preference table $x_i \succ \overline{x_i}$ for every i such that $M \models x_i$ and the preference table $\overline{x_i} \succ x_i$ for every i such that $M \models \neg x_i$. For instance, if $M(x_1) = \top$, $M(x_2) = \bot$, $M(x_3) = \bot$ and $M(x_4) = \top$ then $\mathcal{N}_M$ contains the preference tables $\{x_1 \succ \overline{x_1},\ \overline{x_2} \succ x_2,\ \overline{x_3} \succ x_3,\ x_4 \succ \overline{x_4}\}$.

Proposition 2
$M \models \Phi_{\mathcal{E}}$ if and only if $\mathcal{E}$ is weakly consistent with $\mathcal{N}_M$.

Before proving Proposition 2 we establish the following simple Lemma, which is a consequence of Proposition 1 (since it is very simple, we give its proof anyway).

Lemma 1 *Let $\mathcal{N}$ be a CP-net with G containing no edges, and $\vec{y} \neq \vec{x}$. Then $\mathcal{N} \models \vec{x} \succ \vec{y}$ if and only if $\mathcal{N}$ contains $x_i \succ \overline{x_i}$ for every $x_i \in Diff(\vec{x}, \vec{y})$ and $\overline{x_i} \succ x_i$ for every $\overline{x_i} \in Diff(\vec{x}, \vec{y})$.*

Proof: Without loss of generality, let $\vec{x} = x_1 \ldots x_n$ and $\vec{y} = \overline{x_1} \ldots \overline{x_i} x_{i+1} \ldots x_n$. If $\mathcal{N}$ contains $x_1 \succ \overline{x_1}, \ldots, x_i \succ \overline{x_i}$ then $\mathcal{N} \models \vec{x} \succ \vec{y}$. Conversely, without loss of generality assume $\mathcal{N}$ does not contain $x_1 \succ \overline{x_1}$, which implies that it contains $\overline{x_1} \succ x_1$. Consider a lexicographic preference relation $\succ$ on D in which X_1 is the most important attribute. We have $\vec{y} \succ \vec{x}$, and yet $\succ$ extends $\succ_{\mathcal{N}}$, therefore we cannot have $\mathcal{N} \models \vec{x} \succ \vec{y}$. ■

Now we establish Proposition 2:

Proof:

- *(⇐)* Let M be an interpretation and $\mathcal{N}_M$ the CP-net associated with $\mathcal{N}$. Assume $M \not\models \Phi_{\mathcal{E}}$, *i.e.*, there exists an example $\vec{x} \succ \vec{y}$ in $\mathcal{E}$ such that $M_{\mathcal{N}} \models \neg C_{\vec{x}\succ\vec{y}}$. Without loss of generality, let $\vec{x} = x_1 \ldots x_n$ and $\vec{y} = \overline{x_1} \ldots \overline{x_i} x_{i+1} \ldots x_n$. Then we have $M \models \neg x_1 \wedge \ldots \wedge \neg x_i$, therefore $\mathcal{N}_M$ contains $x_1 \succ \overline{x_1}, \ldots, x_i \succ \overline{x_i}$, which by Lemma 1 implies that $\mathcal{N} \models \vec{y} \succ \vec{x}$, therefore $\mathcal{N}_M$ is not consistent with $\vec{x} \succ \vec{y}$, and *a fortiori*, $\mathcal{N}_M$ is not weakly consistent with $\mathcal{E}$.
- *(⇒)* Let M be an interpretation over $x_1, \ldots, x_n$ and $\mathcal{N}_M$ the CP-net associated with M. Assume that $\mathcal{N}_M$ is not weakly consistent with $\mathcal{E}$, which means that there exists an example $\vec{x} \succ \vec{y}$ in $\mathcal{E}$ such that $\mathcal{N}_M \models \vec{y} \succ \vec{x}$. Without loss of generality, let $\vec{x} = x_1 \ldots x_n$ and $\vec{y} = \overline{x_1} \ldots \overline{x_i} x_{i+1} \ldots x_n$. By Lemma 1 this implies that $\mathcal{N}_M$ contains $\overline{x_1} \succ x_1, \ldots, \overline{x_i} \succ x_i\}$. This implies that $M \models \neg x_1 \wedge \ldots \wedge \neg x_i$, therefore $M \not\models C_{\vec{x}\succ\vec{y}}$, and *a fortiori*, $M \not\models \neg\Phi_{\mathcal{E}}$.

■

Corollary 1 *$\mathcal{E}$ is weakly $\langle \mathcal{V}, \emptyset \rangle$-compatible if and only if $\Phi_{\mathcal{E}}$ is satisfiable.*

This correspondence between unconditional CP-nets and interpretations over $\mathcal{V}$ enables us to draw the following result:

Proposition 3 *Deciding whether a set of examples is weakly $\langle \mathcal{V}, \emptyset \rangle$- compatible is* NP*-complete .*

Proof: Membership is easy: given a set of examples $\mathcal{E}$, guess an unconditional CP- net $\mathcal{N}$ and check that $\mathcal{E}$ is weakly consistent with $\mathcal{N}$, which can be done in time $\mathcal{O}(|\mathcal{E}|.n)$ using Lemma 1. For hardness we use the following reduction from 3SAT. Let $\Phi = \{C_1, \ldots, C_p\}$ be a set of 3-clauses. For every $C = l_1 \vee l_2 \vee l_3$ in Φ create an example $e_C = (\vec{x} \succ \vec{y})$ with

- $\vec{x} = \varepsilon_1.x_1\varepsilon_2.x_2 \ldots \varepsilon_n.x_n$,
- $\vec{y} = \varepsilon'_1.x_1\varepsilon'_2.x_2 \ldots \varepsilon'_n.x_n$,
- for every i, $\varepsilon_i.x_i = \begin{cases} x_i & \text{if } l_j = x_i \text{ for some } j \\ \neg x_i & \text{if } l_j = \neg x_i \text{ for some } j \\ x_i & \text{otherwise} \end{cases}$

- for every i, $\varepsilon'_i.x_i = \begin{cases} \neg x_i & \text{if } l_j = x_i \text{ for some } j \\ x_i & \text{if } l_j = \neg x_i \text{ for some } j \\ x_i & \text{otherwise} \end{cases}$

Now, let $\mathcal{E}_\Phi = \{e_C \mid C \in \Phi\}$. For example, if $\Phi = \{x_1 \vee \neg x_2 \vee x_3, \neg x_1 \vee x_2 \vee x_4, x_2 \vee x_3 \vee x_4\}$ then $\mathcal{E}_\Phi = \{x_1\overline{x_2}x_3x_4 \succ \overline{x_1}x_2\overline{x_3}x_4, \overline{x_1}x_2x_3x_4 \succ x_1\overline{x_2}x_3\overline{x_4}, x_1x_2x_3x_4 \succ x_1\overline{x_2x_3x_4}\}$. We easily check that $\Phi_{\mathcal{E}_\Phi} = \mathcal{E}$, therefore, using Corollary 1 we get that Φ is satisfiable if and only if $\mathcal{E}$ is weakly $\langle \mathcal{V}, \emptyset \rangle$-compatible. ■

The generalization to non-binary domains is not difficult. Instead of having one propositional symbol per attribute, we have one propositional symbol for each pair of values of a attribute. For instance, if we have a attribute X whose domain is $\{d_1, d_2, d_3\}$ then we have the three propositional symbols $d_1 \succ d_2$, $d_1 \succ d_3$ and $d_2 \succ d_3$. The main difference with the binary case is the transitivity requirement. Let $Trans = \bigwedge_{X_i \in \mathcal{V}} Trans_{X_i}$ be the propositional formula expressing transitivity – for instance, for $D_1 = \{d_1, d_2, d_3\}$ we have $Trans_{X_1} = (d_1 \succ d_2 \wedge d_2 \succ d_3 \rightarrow d_1 \succ d_3) \wedge (d_1 \succ d_3 \wedge \neg(d_2 \succ d_3) \rightarrow \neg(d_1 \succ d_2) \wedge ....$ Note that $Trans$ is polynomially long.

The one-to-one correspondence between interpretations and CP-nets now works only for interpretations satisfying $Trans$, and Proposition 2 is generalized into:

Proposition 4 *$M \models \Phi_\mathcal{E} \wedge Trans$ if and only if $\mathcal{N}_M$ is weakly consistent with $\mathcal{N}_M$.*

And Corollary 1 becomes:

Corollary 2 *$\mathcal{E}$ is weakly $\langle \mathcal{V}, \emptyset \rangle$-compatible if and only if $\Phi_\mathcal{E} \wedge Trans$ is satisfiable.*

Computing a CP-net implied by a set of examples

It is easy to characterize whether there exists a CP-net that implies $\mathcal{E}$. With each example $\vec{x} \succ \vec{y}$ we associate the following cube (conjunction of literals) $\Gamma_{\vec{x} \succ \vec{y}}$: it contains x_i iff $x_i \in Diff(\vec{x}, \vec{y})$ and $\neg x_i$ iff $\overline{x_i} \in Diff(\vec{x}, \vec{y})$. Let $\Gamma_\mathcal{E} = \bigwedge\{\Gamma_e \mid e \in \mathcal{E}\}$. Using Lemma 1, we get the following result:

Proposition 5 *$M \models \Gamma_\mathcal{E}$ if and only if $\mathcal{N}_M$ implies $\mathcal{E}$.*

Corollary 3 *There exists a CP-net implying $\mathcal{E}$ if and only if $\Gamma_\mathcal{E}$ is satisfiable.*

Corollary 4 *Deciding whether there exists a CP-net implying $\mathcal{E}$ is in* P.

Learning non-separable preference relations over a fixed acyclic structure

We no longer assume that the preference relation is separable, but we assume that it can be represented by a CP-net over a fixed acyclic graph $(\mathcal{V}, E)$: we are given a set of examples of pairwise comparisons $\mathcal{E}$, and we want to generate a set $\mathcal{P}$ of preference tables for the graph $(\mathcal{V}, E)$ such that the CP-net $\mathcal{N} = ((\mathcal{V}, E), \mathcal{P})$, is weakly compatible with $\mathcal{E}$. Ideally, we would be able to generalize in a simple way the technique given from Section : that is, we translate examples into propositional clauses, the models of which would correspond to CP-nets that are weakly consistent with the examples. Unfortunately, this is not simple.

The following result, by (Boutilier et al. 2004), provides a condition that ensures weak compatibility of a CP-net with a given example $\vec{o} \succ \vec{o}' \in \mathcal{E}$:

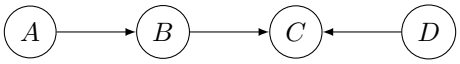
Proposition 6 *(Boutilier et al. 2004, Corollary 4.1) If $\mathcal{N}$ is an acyclic CP-net, $\vec{o}$, $\vec{o}'$ are two outcomes, and if there exists an attribute $X \in \mathcal{V}$ such that $\vec{o}$ and $\vec{o}'$ assign the same values to all ancestors of X in $\mathcal{N}$, and such that, given the values assigned by $\vec{o}$ and $\vec{o}'$ to the parents of X, $\vec{o}$ assigns a more preferred value to X than that assigned by $\vec{o}'$ according to the preference table of $\mathcal{N}$ for X, then $\vec{o}' \not\succ_\mathcal{N} \vec{o}$.*

Note the condition is not necessary. However, given a set $\mathcal{E}$ of examples and a given structure $(\mathcal{V}, E)$, we can generate propositional clauses that correspond to these conditions: if we find a set of tables that satisfies these clauses, then we are certain that the CP-net is weakly consistent with the set of examples.

Let us first define the propositional literals that will be used in the clauses. Given an acyclic graph $(\mathcal{V}, E)$ and an attribute $X \in \mathcal{V}$, let U be the set of its parents in the graph: the table for X in a CP-net over $(\mathcal{V}, E)$ contains, for every assignment u for the attributes in U, and any pair of distinct values x, x' in the domain of X, either $u : x \succ x'$ or $u : x' \succ x$. So we can define a propositional variable whose truth value says which is the case of the two possibilities. So our propositional language contains a propositional variable for every pair of possible values for every attribute $X \in \mathcal{V}$ and every assignment for the parents of X. Without explicitly giving a name to this variable, we will simply represent the literals for variable X and assignment u by $u : x \succ x'$ and $u : x' \succ x$.

Now, given a pairwise comparison of two outcomes $\vec{o} \succ \vec{o}'$, and a graph $\mathcal{G} = (\mathcal{V}, E)$, we define a clause $\Psi_{\mathcal{G}, \vec{o} \succ \vec{o}'}$ as the set of the literals $u : x \succ x'$ for every variable X such that $\vec{o}$ and $\vec{o}'$ assign the same values to all ancestors of X in $\mathcal{G}$ and the value assigned to X by $\vec{o}$ is x, that assigned by $\vec{o}'$ is x', and $x \neq x'$, and where u is the value assigned by $\vec{o}$ and $\vec{o}'$ to the parents of X. Note that if $\mathcal{G} = (\mathcal{V}, \emptyset)$, then $\Psi_{\mathcal{G}, \vec{o} \succ \vec{o}'} = \Phi_{\vec{o} \succ \vec{o}'}$. We will denote by $\Psi_{\mathcal{G}, \mathcal{E}}$ the conjunction of the clauses corresponding to all examples of $\mathcal{E}$.

For example, suppose we have four binary variables A, B, C and D, and the following graph $\mathcal{G}$:



Then $\Psi_{\mathcal{G}, abcd \succ a\overline{b}\overline{c}d} = a : b \succ \overline{b}$, whereas $\Psi_{\mathcal{G}, abcd \succ \overline{a}\overline{b}\overline{c}\overline{d}} = a \succ \overline{a} \vee d \succ \overline{d}$.[3]

[3]Note that the clause $\Psi_{\vec{o} \succ \vec{o}'}$ is never empty: assume it is, and take any order on $\mathcal{V}$ (w.l.o.g., $X_1 \succ \ldots \succ X_n$) being compatible with $\mathcal{G}$ (which is possible because $\mathcal{G}$ is acyclic); we prove by induction on k that $\vec{o}$ and $\vec{o}'$ assign the same values to X_k. This is true for $k = 1$, because $\Psi_{\vec{o} \succ \vec{o}'}$ does not contain any literal referring to X_1, and X_1 has no parents in G. Assume it is true for all $j \leq k$. $\Psi_{\vec{o} \succ \vec{o}'}$ does not contain any literal referring to X_{k+1}, and the values of the parents of X_{k+1} (which are contained in $\{\mathcal{X}_1, \ldots, X_k\}$)

Corollary 5 *If $\mathcal{G}$ is an acyclic graph and $\Psi_{\mathcal{G},\mathcal{E}}$ is satisfiable, then $\mathcal{E}$ is weakly $\mathcal{G}$-compatible.*

In order to obtain strong compatibility, we can use a stronger formula: we let, for every $(\vec{o},\vec{o}') \in \mathcal{E}$, $\Lambda_{\mathcal{G},\vec{o}\succ\vec{o}'} = \Psi_{\mathcal{G},\vec{o}\succ\vec{o}'} \wedge \neg\Psi_{\mathcal{G},\vec{o}'\succ\vec{o}}$. Then $\Lambda_{\mathcal{G},\mathcal{E}}$ is the conjunction of the formulas corresponding to all examples of $\mathcal{E}$.

Proposition 7 *If $\mathcal{G}$ is an acyclic graph and $\Lambda_{\mathcal{G},\mathcal{E}}$ is satisfiable, then $\mathcal{E}$ is strongly $\mathcal{G}$-compatible.*

Proof: We first define two relations between outcomes: we let $\vec{o}\gtrdot_{\mathcal{N}}\vec{o}'$ when the conditions of proposition 6 are met: there exists a variable $X \in \mathcal{V}$ such that $\vec{o}$ and $\vec{o}'$ assign the same values to all ancestors of X in $\mathcal{N}$, and such that, given the values assigned by $\vec{o}$ and $\vec{o}'$ to the parents of X, $\vec{o}$ assigns a more preferred value to X than that assigned by $\vec{o}'$ according to the preference table of $\mathcal{N}$ for X. Let then $\vec{o} \gg_{\mathcal{N}} \vec{o}'$ if $\vec{o}\gtrdot_{\mathcal{N}}\vec{o}'$ but $\vec{o}' \not\gtrdot_{\mathcal{N}} \vec{o}$. (Boutilier et al. 2004) prove that the transitive closure of $\gg_{\mathcal{N}}$ is irreflexive and contains $\succ_{\mathcal{N}}$. Now suppose that a CP-net $\mathcal{N}$ satisfies $\Lambda_{\mathcal{G},\mathcal{E}}$, then $\gg_{\mathcal{N}}$ is such that $\vec{o} \gg_{\mathcal{N}} \vec{o}'$ for every $(\vec{o},\vec{o}') \in \mathcal{E}$. Consider now a completion $\succ$ of $\gg_{\mathcal{N}}$: it satisfies $\mathcal{E}$, and it is a completion of $\succ_{\mathcal{N}}$. ■

Note that the formula $\Lambda_{\mathcal{G},\mathcal{E}}$ is very strong: in the case where the graph has no vertex, it ensures that the resulting CP-net implies the examples.

It is possible to define another formula, the unsatisfiability of which ensures that $\mathcal{E}$ is not weakly $\mathcal{G}$-compatible, but at a high cost: Proposition 1 indicates that an example $(\vec{x},\vec{y})$ is implied by a CP-net if and only if there is a swapping sequence $\vec{x} = \vec{x}_0, \vec{x}_1, \ldots, \vec{x}_n = \vec{y}$ from $\vec{x}$ to $\vec{y}$ such that $x_i \succ x_{i+1}$ for every i; so the example is weakly consistent if an only if for every sequence of swaps $\vec{y} = \vec{x}_0, \vec{x}_1, \ldots, \vec{x}_n = \vec{x}$ from from $\vec{y}$ to $\vec{x}$, $x_i \succ x_{i+1}$ holds for at least one i. However, the translation of a set of examples into a set of clauses using this characterization does not seem to be of practical use, since there can be too many decreasing sequences of flips from one outcome to another. Even if we restrict our attention to what we may call *direct* sequences, ones in which no variable is flipped more that once, there will be be $|Diff(\vec{x},\vec{y})|!$ such sequences; and by doing so we would lose the practical usefulness of the approach.

We end up this section by briefly addressing complexity issues. We know from (Domshlak and Brafman 2002) that dominance checking in acyclic CP-nets is NP-complete. Therefore, checking whether an acyclic CP-net implies (resp. is weakly consistent with) a set of examples is NP-complete (resp. coNP-complete). However, checking whether a given set of examples is weakly or implicatively $\mathcal{G}$-compatible requires first finding the suitable CP-net; such a CP-net is exponentially large in the maximum number of parents in $\mathcal{G}$, therefore weak and implicative $\mathcal{G}$-compatibility may well be above NP and coNP, except in the specific case where the number of parents is bounded by a constant (in the latter case, implicative $\mathcal{G}$-compatibility is NP-complete and weak $\mathcal{G}$-compatibility is in Σ_2^p).

are the same in $\vec{o}$ and $\vec{o}'$, therefore o and o' must assign the same values to X_{k+1}. Therefore we have $\vec{o} = \vec{o}'$, which is impossible because examples involve distinct outcomes.

Conclusion

Learning a good representation of an ordering of multiattribute outcomes is a difficult task because of the exponential number of these orderings. CP-nets provide a compact, but incomplete, representation for such an ordering. Because not all orderings can be exactly captured by a CP-net, it seems reasonable to learn a CP-net consistent with all examples, rather than to look for a CP-net that would imply all of them. In the specific case of separable preference relations, checking if a set of examples can be implied by an acyclic CP-net can be achieved in polynomial time, but on the other hand, we may often fail to find such a CP-net implying the examples. Moreover, although checking if there exists a CP-net that is consistent with each example taken individually is NP-complete (still in the case of separable preference relations), our translation of the problem into clauses means that a good SAT solver may be able to find such a CP-net in reasonable time. As for ensuring global consistency, this seems much more difficult: we do not even have a proof of membership to NP.

We have assumed in the paper that a structure $\mathcal{G}$ is given, and that we want to learn a CP-net over that graph. In general, we may not know the structure. In this case, the goal of the learning problem is to learn the structure of the CP-net as well as conditional preference tables for this graph. It seems natural to try to learn a CP-net over a structure as simple as possible, as suggested in (Athienitou and Dimopoulos 2007). In order to achieve that, we can start with a graph with no edge, and try to learn tables for this graph. If this is not successful, we try to learn tables for CP-nets with one edge. Again, if this is not successful, we can try to find tables for CP-nets with two edges and so on... Of course, enumerating all possible structures would not be feasible, and would not be desirable either: it may be the case that only a CP-net with a very complex structure is weakly consistent with all examples; this may indicate that we are over-fitting the data, because data is noisy or because the total ordering that we should learn cannot be represented by a CP-net. In order to keep the CP-net simple, rather than aiming for a CP-net that is 100% weakly compatible with the examples, we can compute, for each CP-net, its rate of weak compatibility with the examples, and learn a CP-net that represents a good tradeoff between this rate of compatibility and simplicity.

References

Athienitou, F., and Dimopoulos, Y. 2007. Learning CP-networks: a preliminary investigation. In *Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling (PREF'07)*.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. CP-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Brafman, R. I., and Domshlak, C. 2002. Introducing variable importance tradeoffs into CP-nets. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighteenth Annual Conference*, 69–76.

Chen, L., and Pu, P. 2004. Survey of preference elicitation methods. Technical Report 200467, Ecole Polytechnique Fédérale de Lausane.

Domshlak, C., and Brafman, R. 2002. CP-nets—reasoning and consistency testing. In *Proceedings of KR0-2*, 121–132.

Doyle, J.; Shoham, Y.; and Wellman, M. P. 1991. A logic of relative desire (preliminary report). In *ISMIS '91: Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems*, 16–31. London, UK: Springer-Verlag.

Guo, Y.; Müller, J.; and Weinhardt, C. 2003. Learning user preferences for multi-attribute negotiation: An evolutionary approach. In *CEEMAS*, 303–313.

Ha, V. A., and Haddawy, P. 1997. Problem-focused incremental elicitation of multi-attribute utility models. In *UAI*, 215–222.

Keeney, R. L., and Raiffa, H. 1976. *Decision with Multiple Objectives: Preferences and Value Trade-offs*. Wiley.

Miller, B. N.; Albert, I.; Lam, S. K.; Konstan, J. A.; and Riedl, J. 2003. Movielens unplugged: Experiences with an occasionally connected recommender system. In *Proceedings of ACM 2003 Conference on Intelligent User Interfaces (IUI'03)*. Chapel Hill, North Carolina: ACM.

Perny, P., and Zucker, J.-D. 2001. Preference-based search and machine learning for collaborative filtering: the film-conseil movie recommender system. *I3* 1(1):1–40.

Sachdev, M. 2007. On learning of ceteris paribus preference theories. Master's thesis, Graduate Faculty of North Carolina State University.

Viappiani, P.; Faltings, B.; and Pu, P. 2006a. Evaluating preference-based search tools: a tale fo two approaches. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, 205–210.

Viappiani, P.; Faltings, B.; and Pu, P. 2006b. Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research* 27:465–503.

Wilson, N. 2004. Consistency and constrained optimisation for conditional preferences. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, 888–892.

Defeasible Logic to Model n-person Argumentation Game

Duy Hoang Pham, Subhasis Thakur, Guido Governatori
School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia
{pham,subhasis,guido}@itee.uq.edu.au}

Abstract

In multi-agent systems, an individual agent can pursue its own goals, which may conflict with those hold by other agents. To settle on a common goal for the group of agents, the argumentation/dialogue game provides a robust and flexible tool where an agent can send its explanation for its goal in order to convince other agents. In the setting that the number of agents is greater than two and they are equally trustful, it is not clear how to extend existing argumentation/dialogue frameworks to tackle conflicts from many agents. We propose to use the defeasible logic to model the n-person argumentation game and to use the majority rule as an additional preference mechanism to tackle conflicts between arguments from individual agents.

Introduction

In a group of agents, there are several situations requiring agents to settle on a common goal despite that each agent can pursue its goals which may conflict with other agents. A simple but efficient method to tackle the problem is to give weights over the goals. However, this method is not robust and limits the autonomy of an individual agent. Also, the conflicts among agents are likely to arise from a partial view and incomplete information on working environment of individual agents. To settle conflicts among agents an agent can argue to convince others about its pursued goal and provides evidences to defend its claim. This interaction between agents can be modelled as an argumentation game (Prakken & Sartor 1996; Jennings *et al.* 1998; Parsons & McBurney 2003; Amgoud, Dimopoulos, & Moraitis 2007). In an argumentation game, an agent can propose an explanation for its pursued goal (i.e., an argument), which can be rejected by counter-evidences from other agents. This interaction can be iterated until an agent (the winner) successfully argues its proposal against other agents. The argumentation game approach offers a robust and flexible tool for agents to resolve conflicts by evaluating the status of arguments. Dung's argumentation semantics (Dung 1995) is widely recognised to establish relationships among arguments. The key notion for a set of arguments is whether a set of arguments is self-consistent and provides the base to derive a conclusion. A conclusion is justified, and thus provable, if there is a set of supporting arguments and all counter-arguments are deficient when we consider the arguments in the set of supporting arguments.

An argumentation game is more complicated when the number of participants is greater than two. It is not clear how to extend existing approaches to cover the argumentation in groups of more than two agents, especially when agents are equally trustful. That is arguments from individual agents have the same weight. In this case, the problem amounts to how to decide which argument has precedence over competitive arguments. In other words, the problem is to determine the global collective preference of a group of agents.

The main idea behind our approach is that if individual preferences of agents are not sufficient to solve a conflict (for example, we have several arguments without any relative preference over them), the group of agents uses the majority rule (Lin 1996) over initial proposals to determine the "most common" claim known as the "topic" of a dialogue. That is the topic preferred by the majority of the group. An agent either supports the topic or defends its own claim against the topic. Our majority mechanism simplifies the complexity of the n-person argumentation and provides a strategy for an agent to select an argument for defending its proposal. That is an argument causing more "supporters" to reconsider "their attitude" will be preferred by defending agents.

Each of our agents has three types of knowledge: its private knowledge, background knowledge, and knowledge obtained from other agents. The background knowledge presents the expected behaviour of a member of the group that is commonly shared by the group. The knowledge about other agents growing during the interactions enables an agent to efficiently convince others about its own goal. Essentially, the background knowledge is preferred over other sources because it represents common expectations and constraints of the group. Any argument violating the background knowledge is not supported by the group.

Defeasible logic is chosen as our underlying logic for the argumentation game due to its efficiency, simplicity in representing incomplete and conflicting information and relationships with logic programming (Antoniou *et al.* 2006). Furthermore, the logic has a powerful and flexible reasoning mechanism (Antoniou *et al.* 2000; Maher *et al.* 2001) which enables our agents to capture Dung's argumentation semantics by using two features of defeasible reasoning, namely the ambiguity propagating (the preference over conflicts is

unknown) and ambiguity blocking (the preference is given).

Our paper is structured as follows. In the second section, we briefly introduce essential notions of defeasible logics, the construction of arguments using defeasible reasoning with respect to (w.r.t) ambiguous information, and the majority rule. In the third section, we introduce n-person argumentation framework using defeasible logic. We present firstly the external model of agents' interaction, which describes a basic procedure for an interaction between agents. Secondly, we define the internal model, which shows how an agent can deal with different individual knowledge sources in order to propose and to defend its goal against the other agents. The fourth section provides an overview of research works related to our approach. The final section concludes the paper.

Background

Defeasible Logic

Following the presentation in (Billington 1993), the basic components of defeasible logic (DL) are: *facts*, *strict rules*, *defeasible rules*, *defeaters*, and a *superiority relation*.

Facts are undeniable statements, which are always true. *Strict rules*, similar to rules in classical logics, are rules whose conclusions are unquestionable. *Defeasible rules* are different from strict rules in the way that their conclusions can be overridden by contrary evidences. *Defeaters* are rules that cannot be used to draw any conclusion but to prevent some conclusions from some defeasible rules by producing evidence to the contrary. The *superiority relation* defines priorities among rules. That is, one rule may override the conclusion of another rule when we have to solve a conflict between rules with opposite conclusions.

A defeasible theory D is a triple $(F,R,>)$ where F is a finite set of facts, R a finite set of rules, and $>$ a superiority relation on R.

The language of DL consists of a finite set of literals. Given a literal l, we use $\sim l$ to denote the propositional literal complementary to l, that is if $l = p$, then $\sim l = \neg p$, and if $l = \neg p$, than $\sim l = p$.

A rule r in R is composed of an antecedent or body $A(r)$ and a consequent or head $C(r)$. $A(r)$ consists of a finite set of literals while $C(r)$ contains a single literal. $A(r)$ can be omitted from the rule if it is empty. The set of rules R can include all three types of rules, namely R_s (strict rules), R_d (defeasible rules), and R_{dft} (defeaters). We will use R_{sd} for the set of strict and defeasible rules, and $R[q]$ for the set of rules whose head is q.

A conclusion derived from the theory D is a tagged literal and is categorised according to how the conclusion can be proved:

- $+\Delta q$: q is definitely provable in D
- $-\Delta q$: q is definitely unprovable in D.
- $+\partial q$: q is defeasibly provable in D.
- $-\partial q$: q is defeasibly unprovable in D.

Provability is based on the concept of a derivation (or proof) in $D = (F,R,>)$. Informally, definite conclusions can derive from strict rules by forward chaining, while defeasible conclusions can obtain from defeasible rules iff all possible "attacks" are rebutted due to the superiority relation or defeater rules.

A derivation is a finite sequence $P = (P(1),\ldots,P(n))$ of tagged literals satisfying proof conditions (which correspond to inference rules for each of the four kinds of conclusions). $P(1..i)$ denotes the initial part of the sequence P of length i. In the follows, we present the proof for definitely and defeasibly provable conclusions[1]:

$+\Delta$: If $P(i+1) = +\Delta q$ then
(1) $q \in F$ or
(2) $\exists r \in R_s[q]\ \forall a \in A(r) : +\Delta a \in P(1..i)$

$+\partial$: If $P(i+1) = +\partial q$ then either
(1) $+\Delta q \in P(1..i)$ or
(2.1) $\exists r \in R_{sd}[q]\ \forall a \in A(r) : +\partial a \in P(1..i)$ and
(2.2) $-\Delta{\sim}q \in P(1..i)$ and
(2.3) $\forall s \in R_{sd}[{\sim}q]$ either
(2.3.1) $\exists a \in A(s) : -\partial a \in P(1..i)$ or
(2.3.2) $\exists t \in R_{sd}[q]$ such that $t > s$ and
$\forall a \in A(t) : +\partial a \in P(1..i)$

The set of conclusions of a defeasible theory is finite[2], and it can be computed in linear time (Maher 2001). In addition, several efficient implementations have been proposed (see (Maher *et al.* 2001)).

Example 1 Consider the defeasible theory D has a set of defeasible rules:

$$R_d = \{r_1 :\Rightarrow a; r_2 :\Rightarrow {\sim}a; r_3 :\Rightarrow b; r_4 : a \Rightarrow {\sim}b\}$$

and a superiority relation

$$> = \{r_2 > r_1\}$$

r_1 and r_2 have empty therefore they are applicable to derive $+\partial a$ and $+\partial{\sim}a$ respectively. These conclusions are clearly ambiguous. Thanks to the superiority relation, the conclusion of a is overridden. That means $-\partial a$ is in the conclusions from theory D. As a result, $+\partial b$ is added to the conclusion set without any ambiguity. r_4 is no longer applicable due to $-\partial a$.

Defeasible logic can be extended by an ambiguity propagating variant (See (Governatori *et al.* 2004; Antoniou *et al.* 2000)). The superiority relation is not considered in the inference process. The inference with the ambiguity propagation introduces a new tag Σ. A literal p ($+\Sigma p$) means p is supported by the defeasible theory and there is a monotonic chain of reasoning that would lead us to conclude p in the absence of conflicts. A literal that is defeasibly provable ($+\partial$) is supported, but a literal may be supported even though it is not defeasibly provable. Thus support is a weaker notion than defeasible provability.

[1] Refer to (Antoniou *et al.* 2001) for proof conditions for all tagged conclusions.

[2] It is the Herbrand base that can be built from the literal occurring in the rules and the facts of the theory

$+\Sigma$:	$-\Sigma$:
If $P(i+1) = +\Sigma q$ then	If $P(i+1) = -\Sigma q$ then
$\exists r \in R_{sd}[q]$:	$\forall r \in R_{sd}[q]$:
$\forall a \in A(r) : +\Sigma a \in P(1..i)$	$\exists a \in A(r) : -\Sigma a \in P(1..i)$

We can achieve ambiguity propagation behaviour by making a minor change to the inference condition for $+\partial_{AP}$[3]

$+\partial_{AP}$: If $P(i+1) = +\partial q$ then either
(1) $+\Delta q \in P(1..i)$ or
  (2.1) $\exists r \in R_{sd}[q]\ \forall a \in A(r) : +\partial_{AP} a \in P(1..i)$ and
  (2.2) $-\Delta{\sim}q \in P(1..i)$ and
  (2.3) $\forall s \in R_{sd}[{\sim}q]$
    $\exists a \in A(s) : -\partial_{AP} a \in P(1..i)$ or

Example 2 We modify the defeasible theory D in example 1 by removing the superiority relation:

$$R_d = \{r_1 :\Rightarrow a; r_2 :\Rightarrow {\sim}a; r_3 :\Rightarrow b; r_4 : a \Rightarrow {\sim}b\}$$

Without the superiority relationship, there is no means to decide between a and $\sim a$ due to both of r_1 and r_2 are applicable. In a setting where the ambiguity is blocked, b is not ambiguous because r_3 for b is applicable whilst r_4 is not since its antecedent is not provable. If the ambiguity is propagated, we have evidence supporting all of four literals since all of the rules is applicable. $+\Sigma a, +\Sigma{\sim}a, +\Sigma b$ and $+\Sigma{\sim}b$ are included in the conclusion set. Moreover we can derive $-\partial a$, $-\partial{\sim}a$, $-\partial b$ and $-\partial{\sim}b$ showing that the resulting logic exhibits an ambiguity propagating behaviour. In the second setting b is ambiguous, and its ambiguity depends on that of a.

Argumentation by Defeasible Logic

In what follows, we briefly introduce the basic notions of an argumentation system using defeasible logic as underlying logical language. Moreover, we present the acceptance of an argument w.r.t Dung's semantics.

Definition 1 *An argument A for a literal p based on a set of rules R is a (possibly infinite) tree with nodes labelled by literals such that the root is labelled by p and for every node with label h:*

1. *If $b_1, \ldots, b_n$ label the children of h then there is a rule in R with body $b_1, \ldots, b_n$ and head h.*
2. *If this rule is a defeater then h is the root of the argument.*
3. *The arcs in a proof tree are labelled by the rules used to obtain them.*

In general, arguments are defined to be proof trees (or monotonic derivations). Defeasible logic requires a more general notion of proof tree that admits infinite trees, therefore the distinction is kept between an unrefuted, but infinite, chain of reasoning and a refuted chain. Depending on the rules used, there are different types of arguments.

- A supportive argument is a finite argument in which no defeater is used.
- A strict argument is an argument in which only strict rules are used.
- An argument that is not strict, is called defeasible

Relationships between two arguments, A and B, are determined by those of literals which are constituted in these arguments. An argument A *attacks* a defeasible argument B if a conclusion of A is the complement of a conclusion of B, and that conclusion of B is not part of a strict sub-argument of B. A set of arguments $\mathscr{S}$ attacks a defeasible argument B if there is an argument A in $\mathscr{S}$ that attacks B.

A defeasible argument A is *undercut* by a set of arguments $\mathscr{S}$ if $\mathscr{S}$ supports an argument B attacking a proper non-strict sub-argument of A. An argument A is undercut by $\mathscr{S}$ means we can show that some premises of A cannot be proved if we accept the arguments in $\mathscr{S}$.

It is noticed that the concepts of the attack and undercut concern only defeasible arguments and sub-arguments. For strict arguments we stipulate that they cannot be undercut or attacked.

A defeasible argument is assessed as valid if we can show that the premises of all arguments attacking it cannot be proved from the valid arguments in $\mathscr{S}$. The concepts of provability depend on the methods used by the reasoning mechanism to tackle ambiguous information. According to the features of the defeasible reasoning, we have two definitions of acceptable arguments (definition 2 and 3).

Definition 2 *In case of the reasoning with the ambiguity propagation, an argument A for p is acceptable w.r.t a set of arguments $\mathscr{S}$ if A is finite, and*

1. *A is strict, or*
2. *every argument attacking A is attacked by $\mathscr{S}$.*

Definition 3 *If the reasoning with the ambiguity blocking is used, an argument A for p is acceptable w.r.t a set of arguments $\mathscr{S}$ if A is finite, and*

1. *A is strict, or*
2. *every argument attacking A is undercut by $\mathscr{S}$.*

Due to the concept of acceptance, we can determine the status of an argument. If an argument can resist a reasonable refutation, this argument is justified (definition 4). If an argument can not overcome attacks from other arguments, this argument is rejected (definition 5).

Definition 4 *Let D be a defeasible theory. We define J_i^D as follows.*

- $J_0^D = \emptyset$
- $J_{i+1}^D = \{a \in Args_D |\ a \text{ is acceptable w.r.t } J_i^D\}$

The set of justified arguments in a defeasible theory D is $JArgs^D = \bigcup_{i=1}^{\infty} J_i^D$.

Definition 5 *Let D be a defeasible theory and $\mathscr{T}$ be a set of arguments. We define $R_i^D(\mathscr{T})$ as follows.*

- $R_0^D(\mathscr{T}) = \emptyset$
- $R_{i+1}^D(\mathscr{T}) = \{a \in Args_D |\ a \text{ is rejected by } R_i^D(\mathscr{T}) \text{ and } \mathscr{T}\}$.

The set of rejected arguments in a defeasible theory D w.r.t. $\mathscr{T}$ is $RArgs^D(\mathscr{T}) = \bigcup_{i=1}^{\infty} R_i^D(\mathscr{T})$.

[3]The proof for $-\partial_{AP}$ is derived from that of $+\partial_{AP}$ using the strong negation principle (Maher *et al.* 2001).

Majority Rule

The *majority rule* from (Lin 1996) retrieves a maximal amount of consistent knowledge from a set of agents' knowledge. Conflicts between agents can be tackled by considering not only the number of agents supporting that information but also the importance (reliability) of the agents. The approach provides a useful and efficient method to discover information largely held by agents. The majority knowledge can be used either to reinforce the current knowledge of an agent or to introduce new information into the agent's knowledge.

Due to possible conflicting information within a source, the merging operator by majority cannot directly apply to our framework. Instead, the majority rule pools potential joint conclusions derived by the defeasible reasoning, which resolves possible conflicts.

Considering the knowledge sources $\{T_1,\ldots,T_n\}$, C_i denotes the set of tagged conclusions that can be derived by the defeasible reasoning from the corresponding theory T_i. The level that the theory T_i supports a literal l corresponds to its weight represented w_{T_i} as follows:

$$support(l,T_i)=\begin{cases} w_{T_i} & l\in C_i \\ 0 & \text{otherwise}\end{cases}$$

The majority knowledge from the others, T_{maj}, whose elements are inferred from $\{C_1,\ldots,C_n\}$ by the majority rule, is determined by the formula:

$$T_{maj}=\left\{c:\sum_{T_i} support(c,T_i)>\frac{\sum w_{T_i}}{2}\right\}$$

n-Person Argumentation Framework

In this section, we develop our framework by using the argument construction from the defeasible reasoning. In particular, we define an external model which describes interactions between agents in order to achieve a goal supported by the majority. Also, we present an internal model which illustrates the reasoning method on knowledge from other agents exposed during interactions.

Model Agents' Interaction

This section describes the basic scenario where an individual agent exchanges arguments to promote its own goal and to reach an agreement by the majority. Consider a set of agents $\mathscr{A}$ sharing a set of goals $\mathscr{G}$ and external constraints represented as a defeasible theory T_{bg}. These external constraints are also known as background knowledge which provides common expectations and restrictions among agents in $\mathscr{A}$. An individual agent in $\mathscr{A}$ can have its own view on the working environment, therefore can pursue its own goals. In this work, we model the interactions among these agents in order to establish a goal accepted by the majority of the group. Due to the partial view and incomplete information of an agent, we believe the argumentation game is a useful method to tackle this problem.

Determine common goal. In order to pursue a goal, an agent generates an argument for its goal. This goal is considered as its main claim. The process to determine the goal supported by the group involves multiple steps in a dialogue as follows.

1. Each agent broadcasts an argument for its goal. The system can be viewed as an argumentation game with n players corresponding to the number of agents.
2. The group of agents determines the dialogue topic by using the majority rule over the set of claims (i.e. the goals from players)[4]. The claim supported by more than a half of the group is selected. If the group can not settle a topic, the previous step is repeated. The dialogue terminates early if agents fail to achieve a majority goal and they do not have any new goal to propose.
3. An agent can rest if its claim is supported by the majority. Otherwise, the agent can provide a new argument to defend its claim against the common one. At this step, the group creates a set of majority arguments $Args_i^{maj}$ and a set of majority premises P_i^{maj} where i indicates the iteration. An agent utilises these sets to select its new arguments for subsequence steps. Also, new arguments are required to be justified by the background knowledge.
4. A dialogue terminates when all agents pass for an iteration (i.e., do not propose a new argument). Now, the group can settle on the common goal and the explanation accepted by the majority of the group.

Example 3 Suppose that there is three agents A_1, A_2, and A_3. A_1 and A_2 respectively propose $Args^{A_1}$ and $Args^{A_2}$

$$Args^{A_1}=\{\Rightarrow e\Rightarrow b\Rightarrow a\}$$
$$Args^{A_2}=\{\Rightarrow e\Rightarrow c\Rightarrow a\}$$

whilst A_3 claims

$$Args^{A_3}=\{\Rightarrow d\Rightarrow \sim a\}.$$

The topic of the dialogue accepted by the majority is a.

Identify majority arguments. Once the group successfully identifies the common claim, the group is divided into two sub-groups namely "pros-group" and "cons-group". Agents in the pros-group support the common claim whilst the cons-group does not. By using the majority rule the agents in the cons-group determine their defensive arguments by attacking the "most common" premise among the arguments from the pros-group. That will force the pros-group to reconsider their claim.

At iteration i, G_i^{maj} is the claim by the majority of active agents (the agents broadcast their arguments). $Args_i^{maj}$ represents the set of majority arguments which are played by the agents to support G_i^{maj}.

$$Args_i^{maj}=\bigcup_{j=0}^{|\mathscr{A}|} Args^{A_j}|Args^{A_j}\vdash G^{maj}$$

[4]Note that agents in $\mathscr{A}$ have the same weight, therefore the majority rule is applied knowledge sources such that each source has the weight of 1.

where $Args^{A_j}$ is the argument played by agent A_j. The set of majority premises at iteration i is

$$P_i^{maj} = \{p | p \in Args_i^{maj}\}$$

We define the preference over P_i^{maj} as given $p_1, p_2 \in P_i^{maj}$, $p_2 \succeq p_1$ if the frequency of p_1 in $Args_i^{maj}$ is less than that of p_2.

Let $i = 0$ be the first iteration that agents in the group reach a common claim. The topic of the dialogue is set to G_0^{maj}. Given two consecutive iterations: i and $i+1$, G_i^{maj} and G_{i+1}^{maj} are incompatible claims. That is the pros-group for G_i^{maj} at iteration i is attacked by the cons-group which gives G_{i+1}^{maj} in the next iteration as the counter-evidence. In the case that the cons-group does not have an argument which directly attacks G_i^{maj}, the cons-group uses the order of premises in P_i^{maj} as a preference mechanism to select a counter-argument. The idea is that P_i^{maj} eventually contains premises which are sub-claims of G_i^{maj}. The higher order a premises p in P_i^{maj} has, the more agents in the pros-group support p. Consequently, if p is rebutted, the pros-group should revise its attitude towards the claim.

Example 4 Reconsidering example 3, we have

$$\begin{aligned} G_0^{maj} &= a \\ Args_0^{maj} &= \{\Rightarrow e \Rightarrow b \Rightarrow a; \Rightarrow e \Rightarrow c \Rightarrow a\} \\ P_0^{maj} &= \{a^2, b^1, c^1, e^2\} \end{aligned}$$

The superscript of a premise in P_0^{maj} represents its frequency in $Args_0^{maj}$. Since the main claim of A_3 does not pass the majority selection, A_3 can defend its proposal by attacking either b or c or e in the next step. An argument against e is likely to be a better selection compared with those against b or c. Another alternative is that A_3 proposes a new argument for $\sim a$ stronger than any of the arguments played by A_1 and A_2

Model Agent's Internal Knowledge

The motivation, which drives an agent to participate in the dialogue, is to promote its own goal. However, its argument for the goal will be accepted if the argument is shared by the majority of the group. To gain the acceptance of the majority, the agent should consider common constraints and expectations of the group, governed by the background knowledge, as well as the attitude of other agents when proposing a claim. The majority rule over the knowledge obtained from other agents enables an agent to probe a common attitude among agents.

At the beginning of the dialogue, the majority rule determines the main claim. In the follow iteration, this rule identifies sub-claims to help an agent to effectively defend its original claim. The idea is that an agent should launch an argument which is likely to alter the majority opinion. The majority rule provides a preference that is among the supportive arguments for the main claim, identifying the most common premise if an agent refutes this premise.

Knowledge representation. An agent, A_{me}, has three types of knowledge including the background knowledge T_{bg}, its own knowledge about working environment T_{me}, and the knowledge about others:

$$\mathscr{T}_{other} = \{T_j : 1 \leq j \leq |\mathscr{A}| \& j \neq me\}$$

T_j is obtained from agent $Ag_j \in \mathscr{A}$ during iterations (proposing arguments for individual goals). All of these knowledge is represented in defeasible logic. $T_j \in \mathscr{T}_{other}$ is constructed from an argument Ar proposed by the agent Ag_j. At iteration i, the theory obtained from Ag_j is accumulated from previous steps $T_i^j = \bigcup_{k=0}^{i} T_k^j$.

In our framework, agents can have conflicting knowledge due to partial view and incomplete information sources. We assume that the defeasible theories contain only defeasible rules and defeasible facts (rules with empty body). Therefore, the knowledge of an agent can be rebutted by that from other agents.

Knowledge Integration. To generate an argument, an agent should ponder knowledge from multiple sources. In this section, we present two simple methods to integrate knowledge sources based on ambiguity blocking and ambiguity propagation: given two sources of knowledge, if the preference between these two sources is known we can perform the ambiguity blocking integration; Otherwise, we can select ambiguity propagation integration.

Ambiguity blocking integration. This integration extends the standard defeasible reasoning by creating a new superiority relation from that of the knowledge sources i.e. given two knowledge sources as T_{sp} – the superior theory, and T_{in} – the inferior theory we generate a new superiority relation $R_d^{sp} > R_d^{in}$ based on rules from two sources. The integration of the two sources denotes as $T_{INT} = T_{sp} \ni T_{in}$. Now the standard defeasible reasoning can be applied for T_{INT} to produce a set of arguments $Args_{AB}^{T_{sp} \ni T_{in}}$.

Example 5 Given two defeasible theories

$$\begin{aligned} T_{bg} = \{&R_d = \{r_1 : e \Rightarrow c; r_2 : g, f \Rightarrow \sim c, r_3 :\Rightarrow e\}; \\ &> = \{r_2 > r_1\}\} \end{aligned}$$

and

$$T_{me} = \{R_d = \{r_1 :\Rightarrow d; r_2 : d \Rightarrow \sim a; r_3 :\Rightarrow g\}\}$$

The integration produces $T_{bg} \ni T_{me} =$

$$\begin{aligned} \{&R_d = \{r_1^{T_{bg}} : e \Rightarrow c; r_2^{T_{bg}} : g, f \Rightarrow \sim c, r_3^{T_{bg}} :\Rightarrow e; \\ &r_1^{T_{me}} :\Rightarrow d; r_2^{T_{me}} : d \Rightarrow \sim a; r_3^{T_{me}} :\Rightarrow g\}; \\ &> = \{r_2^{T_{bg}} > r_1^{T_{bg}}\}\} \end{aligned}$$

Ambiguity propagation integration. Given two knowledge sources T_1 and T_2, the reasoning mechanism with ambiguity propagation can directly apply to the combination theory denoted as $T_{INT} = T_1 + T_2$. There is no preference between the two source of knowledge, therefore, there is no method to solve the conflicts between the two sources. That is the supportive and op-positive arguments for any premise are removed from the final set of arguments. The set of arguments obtained by this integration denotes as $Args_{AP}^{T_1+T_2}$.

Justification by background knowledge. Agent A_{me} generates the set of arguments for its goals by combining its private knowledge T_{me} and the background knowledge T_{bg}. The combination is denoted as $T_{me}^{'} = T_{bg} \ni T_{me}$ and the set of arguments is $Args^{T_{me}^{'}}$. Due to the non-monotonic nature of the underlying logics, the combination can produce arguments being beyond those from individual knowledge. That is the combination can produce arguments which are totally new to the two sources. From A_{me}'s view, this can bring more opportunities to fulfil its goals. However, A_{me}'s arguments must be justified by the background knowledge T_{bg}. In other words, T_{bg} governs essential behaviours (expectations) of the group. Any attack to T_{bg} is not supported by members of $\mathscr{A}$.

Agent A_{me} maintains the consistency with the background knowledge T_{bg} by following procedure:

1. Create $T_{me}^{'} = T_{bg} \ni T_{me}$. The new defeasible theory is obtained by replicating all rules from common constraints T_{bg} into the internal knowledge T_{me} while maintaining the superiority of rules in T_{bg} over that of T_{me}.
2. Use the ambiguity blocking feature to construct the set of arguments $Args_{AB}^{T_{bg}}$ from T_{bg} and the set of arguments $Args_{AB}^{T_{me}^{'}}$ from $T_{me}^{'}$.
3. Remove any argument in $Args^{T_{me}^{'}}$ attacked by those in $Args^{T_{bg}}$, obtaining the justified arguments by the background knowledge
   $JArgs^{T_{me}^{'}} = \{a \in Args^{T_{me}^{'}}$ and a is not attacked by $Args_{AB}^{T_{bg}}\}$

Example 6 Given two defeasible theories, T_{bg} and T_{me} ,in example 5. We have

$$\begin{aligned} Args^{T_{bg}} = \{ & \Rightarrow e; \\ & \Rightarrow e \Rightarrow c\} \\ Args^{T_{bg} \ni T_{me}} = \{ & \Rightarrow e; \\ & \Rightarrow e \Rightarrow c; \\ & \Rightarrow d; \\ & \Rightarrow g; \\ & \Rightarrow d \Rightarrow \sim a\} \end{aligned}$$

In this example, there is not any attack between arguments in $Args^{T_{bg}}$ and $Args^{T_{bg} \ni T_{me}}$. In other words, arguments from $Args^{T_{bg} \ni T_{me}}$ are acceptable by those from $Args^{T_{bg}}$. The set of justified argument w.r.t $Args^{T_{bg}}$

$$JArgs^{T_{bg} \ni T_{me}} = Args^{T_{bg} \ni T_{me}} e;$$

Pondering knowledge from the others. During the dialogue, an agent can exploit the knowledge that other agents exposed in order to defend its main claims. Due to possible conflicts in proposals from other agents, an agent can use the sceptical semantics of the ambiguity propagation reasoning in order to retrieve the consistent knowledge. That is given competing arguments, the agent does not have any preference over them and they will be rejected. The consistent knowledge from the others allows an agent to discover "collective wisdom" distributed among agents. From those arguments, agent A_{me} should justify arguments against the set of majority premises P_i^{maj} at iteration i of the dialogue. The judgement is done by using the arguments from the background knowledge $Args^{T_{bg}}$. The procedure runs as follows:

1. Create a new defeasible theory
   $T_{me}^{''} = T_{bg} \ni T_{me} + \mathscr{T}_{other}$.
2. Generate the set of arguments $Args_{AP}^{T_{me}^{''}}$ from $T_{me}^{''}$ using the feature of ambiguity propagation.
3. Justify the new set of arguments
   $JArgs^{T_{me}^{''}} = \{a | a \in Args_{AP}^{T_{me}^{''}}$ and a is accepted by $Args^{T_{bg}}$.

At iteration i of the dialogue, the group determines the set P_i^{maj} containing premises support by the majority. In order to refute the majority claim, A_{me} can select an argument from $JArgs_{AB}^{T_{me}^{'}} \bigcup JArgs_{AP}^{T_{me}^{''}}$ that attack a premise $p \in P_i^{maj}$. The preference of an argument against p is determined by the weight of p. Since the weight of p is proportional to the number of agents supporting p. If p is attacked, the majority can change in favour to A_{me}.

Example 7 Suppose that

$$T_{bg} = \{R_d = \{r_1 : e \Rightarrow c; r_2 : g, f \Rightarrow \sim c\}; >= \{r_2 > r_1\}\}$$

and the private knowledge of A_{me} has

$$T_{me} = \{R_d = \{r_1 :\Rightarrow d; r_2 : d \Rightarrow \sim a; r_3 :\Rightarrow g\}\}$$

Agent A_{me} currently plays $\Rightarrow d \Rightarrow \sim a$ and knows about other agents

$$\mathscr{T}_{other} = \{T_1, T_2\}$$

where

$$T_1 = \{\Rightarrow e \Rightarrow f \Rightarrow b \Rightarrow a\}$$

and

$$T_2 = \{\Rightarrow e \Rightarrow c \Rightarrow a\}$$

and at this step the majority premises

$$P_i^{maj} = \{a^2, e^2, f^1, b^1, c^1\}.$$

The superscript of an element of P_{mj}^{i} represents the frequency (weight) of this element.

The defeasible reasoning with ambiguity propagation for the combination $T_{bg} + T_{me} + \mathscr{T}_{other}$ generates a set of arguments $\Rightarrow g, \Rightarrow e, \Rightarrow e \Rightarrow f \Rightarrow b, \Rightarrow g, f \Rightarrow \sim c. \Rightarrow g, f \Rightarrow \sim c$ is due to the superiority relation in T_{bg}. Given the current knowledge of A_{me} this is only argument that A_{me} can play.

Related Work

Substantial work have been done on argumentation games in the artificial intelligence and Law-field. (Prakken & Sartor 1996) introduces a dialectical model of legal argument, in the sense that arguments can be attacked with appropriate counterarguments. In the model, the factual premises are not arguable, they are treated as strict rules. (Lial 1998) presents an early specification and implementation of an argumentation game based on the Toulmin argument-schema without a specified underlying logic. (Lodder 2000) presented The

Pleadings Game as a normative formalization and fully implemented computational model, using conditional entailment. The goal of the model was to identify issues in the argumentation rather than as in our case elaborating on the status of the main claim.

Using the defeasible logic to capture concepts of the argumentation game is supported by (Letia & Vartic 2006; Nilsson, Lundstrøm, & Hamfelt 2005) and recently (Thakur *et al.* 2007; Eriksson Lundström *et al.* 2008). (Letia & Vartic 2006) focuses on persuasive dialogues for cooperative interactions among agents. It includes in the process cognitive states of agents such as knowledge and beliefs, and presents some protocols for some types of dialogues (e.g. information seeking, explanation, persuasion). (Nilsson, Lundstrøm, & Hamfelt 2005) provides an extension of defeasible logic to include the step of the adversarial dialogue by defining a metaprogram for an alternative computational algorithm for ambiguity propagating defeasible logic while the logic presented here is ambiguity blocking.

We tackle the problem of evolving knowledge of an agent during iterations, where the argument construction is an extension of (Thakur *et al.* 2007; Eriksson Lundström *et al.* 2008). In our work, we define the notion of majority acceptance and a method to weight arguments. In (Thakur *et al.* 2007), the strength of unchallenged rules is upgraded over iterations. That is the conclusions supported by these rules are not rebutted by the current iteration, these conclusions are unarguable in follow iterations. The upgrade is applied to all participants during iterations of the argumentation game. (Eriksson Lundström *et al.* 2008) distinguishes participants of the argumentation game. That is one participant must provide a strong argument (i.e. a definite proof) in order to defeat arguments from other participants. Both of the works do not directly handle the challenge coming from multiple participants.

We extend the protocol of a argumentation game to settle on a common goal. The termination condition of our framework is either there is no more argument to rebut or an agent can pass its proposal at one iteration. Settling on a common goal among agents can be seen as a negotiation process where agents exchange information to resolve conflicts or to obtain missing information. The work in (Amgoud, Dimopoulos, & Moraitis 2007) provides a unified and general formal framework for the argumentation-based negotiation dialogue between two agents for a set of offers. The work provides a formal connection between the status of a argument including accepted, rejected, and undecided with possible actions of an agent (accept, reject, and negotiate respectively). One important feature of the framework is that this representation is independent with logical languages modelling knowledge of an agent. Moreover, an agent's knowledge is evolved by accumulating arguments during interactions.

We have advantages of using the defeasible logic since it provides us an elegant tool to naturally capture the above statuses of arguments. Accepted, rejected, undecided conditions can be simulated by the proof conditions of defeasible reasoning w.r.t ambiguity of premises. If the preference over knowledge sources is known, the accepted and rejected arguments is corresponding to $(+\partial, -\partial)$ using the feature of ambiguity blocking. Otherwise, three conditions of arguments are derived from $(+\partial, -\partial$ and $+\Sigma)$. These notions correspond to the existence of a positive proof, a negative proof, and a positive support of a premise. In addition, defeasible logic provides a compact representation to accommodate new information from other agents.

From the perspective of coordination among agents, (Parsons & McBurney 2003) presents an argumentation based communication, where agents can exchange arguments for their goals and plans to achieve the goals. The acceptance of an argument of an agent depends on the attitudes of this agent namely credulous, cautious, and sceptical. Also, (Rueda, Garcia, & Simari 2002) proposes a communication mechanism based on argumentation for collaborative BDI agents, in which agents exchange their proposals and counter-proposals in order to reach a mutual agreement. During the course of conversations, an agent can retrieve missing literals (regarded as sub-goals) or fulfil its goals by requesting collaboration from other agents. However, these works did not clearly show how an agent can tackle conflicts from multiple agents, especially when the preference over exchanged arguments is unknown.

The main difference in our framework is the external model where more than two agents can argue to settle on a common goal. Since there is no preference over the proposal of individual agents, the majority rule enables the group to identify the majority preference over individual claims. On one hand, we present the notion of the acceptance by the majority of agents. On the other hand, this notion relaxes the complexity of n-persons argumentation game by partitioning agents into two sub-groups: one supports the major claim; the other opposes it. Moreover, the majority rule allows an agent to probe the attitudes of the group in order to dynamically create a preference over its defensive arguments if its main claim is not accepted by the majority of agents. The strategy to defend against the topic of the dialogue is to attack the most common premise among the arguments supporting the topic.

In our framework, an individual agent efficiently tackle with conflicts from multiple sources of knowledge owing to the use of the defeasible logic as the underlying logic. The construction of arguments requires an individual agent to integrate the background knowledge commonly shared among agents, knowledge from other agents, and its private knowledge. The background knowledge has the priority over the other sources, therefore when integrating any conflict with this knowledge is blocked. Since all agents are equally trustful, the knowledge from other agents has the same weight. To achieve a consensus from knowledge of other agents and to discover "collective wisdom", the ambiguity propagation is applied over all knowledge sources of an individual agent.

Conclusions

This paper has presented an n-person argumentation framework based on the defeasible logic. In the framework, we propose an external model based on the argumentation/dialogue game which enables agents in a group settle on a common goal. An agent proposes its goal including

the explanation and argues with other agents about the goal. At the termination, the group identifies a common goal accepted by the majority of the group and the supportive argument for the goal.

We also propose an internal model of an agent where an individual agent can efficiently construct arguments from multiple sources of knowledge including the background knowledge presenting the common constraints and expectations of the group, knowledge from others which is evolved during iterations, and its private knowledge. The background knowledge is preferred over the other sources of knowledge. Due to the flexibility of defeasible logic in tackling the ambiguous information, these types of knowledges can be efficiently integrated with the private knowledge of an agent (with or without a preference over the knowledge sources) to generate and justify its arguments. The majority rule relaxes the complexity of n-persons argumentation dialogue game. This rule is used to identify the topic of the dialogue among the claims of agents. That is the majority acceptance of an argument. Also, an agent can use the majority rule as a method to select an argument which challenges the major number of agents in order to better defend its goal.

References

Amgoud, L.; Dimopoulos, Y.; and Moraitis, P. 2007. A unified and general framework for argumentation-based negotiation. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 1–8. New York, NY, USA: ACM.

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2000. A flexible framework for defeasible logics. In *Proc. American National Conference on Artificial Intelligence (AAAI-2000)*, 401–405.

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2001. Representation results for defeasible logic. *ACM Transactions on Computational Logic* 2(2):255–287.

Antoniou, G.; Billington, D.; Governatori, G.; and Maher, M. J. 2006. Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming* 6(6):703–735.

Billington, D. 1993. Defeasible logic is stable. *Journal of Logic and Computation* 3:370–400.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–358.

Eriksson Lundström, J.; Governatori, G.; Thakur, S.; and Padmanabhan, V. 2008. An asymmetric protocol for argumentation games in defeasible logic. In *10 Pacific Rim International Workshop on Multi-Agents*, volume 5044. Springer.

Governatori, G.; Maher, M. J.; Antoniou, G.; and Billington, D. 2004. Argumentation Semantics for Defeasible Logic. *J Logic Computation* 14(5):675–702.

Jennings, N. R.; Parsons, S.; Noriega, P.; and Sierra, C. 1998. On argumentation-based negotiation. In *Proceedings of the International Workshop on Multi-Agent Systems*.

Letia, I. A., and Vartic, R. 2006. Defeasible protocols in persuasion dialogues. In *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, 359–362. Washington, DC, USA: IEEE Computer Society.

Lial, T. B.-C. 1998. Specification and implementation of toulmin dialogue game. In J.C. Hage, T.J.M. Bench-Capon, A. K. C. d. V. M. C. G., ed., *Jurix 1998: Jurix: The Eleventh Conference*, 5–20. Nijmegen: Gerard Noodt Instituut.

Lin, J. 1996. Integration of weighted knowledge bases. *Artificial Intelligence* 83:363–378.

Lodder, A. R. 2000. Thomas f. gordon, the pleadings game - an artificial intelligence model of procedural justice. *Artif. Intell. Law* 8(2/3):255–264.

Maher, M. J.; Rock, A.; Antoniou, G.; Billignton, D.; and Miller, T. 2001. Efficient defeasible reasoning systems. *International Journal of Artificial Intelligence Tools* 10(4):483–501.

Maher, M. J. 2001. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming* 1(6):691–711.

Nilsson, J. F.; Lundstrøm, J. E.; and Hamfelt, A. 2005. A metalogic formalization of legal argumentation as game trees with defeasible reasoning. In *Proceedings of ICAIL'05, International Conference on AI and Law*, Proceedings of ICAIL. ACM.

Parsons, S., and McBurney, P. 2003. Argumentation-based dialogues for agent coordination. group decision and negotiation. *Group Decision and Negotiation* (12):415–439.

Prakken, H., and Sartor, G. 1996. A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law* 4:331–368.

Rueda, S. V.; Garcia, A. J.; and Simari, G. R. 2002. Argument-based negotiation among bdi agents. *Journal of Computer Science and Technology* 2(7).

Thakur, S.; Governatori, G.; Padmanabhan, V.; and Eriksson Lundström, J. 2007. Dialogue games in defeasible logic. In *20th Australian Joint Conference on Artificial Intelligence, AI 2007*, volume 4830, 497–506. Springer.

Incorporating a Qualitative Ranked Preference System into Planning

Chris L. Schmidt and James P. Delgrande
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
{cschmidt, jim} @cs.sfu.ca

Abstract

Planning is one of the fundamental problems of Artificial Intelligence. In brief, the problem is to form a plan that will lead from an initial state to a goal state. Typically, goals are absolute and must be satisfied. However, realistically, a user may have preferences along with goals. A plan that satisfies the goals is acceptable, but one that satisfies the goals and complies with expressed preferences is a superior alternative. In this paper we define a system for specifying preferences. The system is purely qualitative, specifying what is preferred without using numbers to denote by how much one possibility is preferred to another. The system is flexible, allowing many different kinds of preferences to be specified and could be adapted for any application where preferences are appropriate. It includes a novel method to express unbounded preferences that cannot be expressed in other systems. We then do a preliminary examination of two search heuristics for producing plans that optimally satisfy a preference specification.

Introduction

In a classical planning problem, a series of actions is generated that will lead from an initial state to a state that satisfies a set of goal conditions. An often more realistic approach to planning is to consider preferences along with goals. The goals are hard conditions that must be met for a plan to be considered successful. Preferences enable soft conditions to be added. These conditions don't have to be met for a plan to be successful, but set forth what plans are preferred when differing plans satisfy the goal conditions. If a planning problem is to drive from Vancouver to San Francisco, the goal is simply to reach San Francisco. One preference might be to visit Portland on the way; another may be to avoid construction when possible.

We propose a flexible system for specifying preferences that could be applied to virtually any field where preferences are useful. We focus on using the system to define preferences over histories and plans. The system is strictly qualitative, not requiring users to attach possibly numeric values to their preferences. This is advantageous since such values can be difficult to elicit from users. Numbers are used as indices when describing a preference specification, however, they are used in an ordinal capacity, not to determine by how much one option is preferred to another. The system includes a construct for defining unbounded preferences that would otherwise be unmanageable. The system allows the strategy for defining preferred histories to be altered to meet a user's needs. We define an approach, but depending on the circumstances a user may want to interpret the information in a preference specification somewhat differently. We also do some early work on examining how a planner may be used to produce preferred plans as defined by a preference specification. We experiment with search guided by two different heuristic functions. Our aim in this is to prove the usefulness of the preference specification system as well as the heuristics themselves.

In the next section we give some background on other work in the field. We then go through some preliminary definitions. Then we define the three levels of our preference system and how they relate. After that we specify a strategy for comparing histories with respect to a preference specification. Following this, we look at two search strategies that can be used to produce optimal histories. To finish, we discuss our work and how it relates to other work in the field.

Background

The research involving preferences is too extensive to properly summarize in this paper. We can only focus on work closely related to our own. A well known preference specification scheme, *CP-nets*, is seen in (Boutilier et al. 2004). Preferences are built around *ceteris paribus* (all else being equal) statements such as "All else being equal, if A is $true$, then B is preferred over $\neg B$". These preferences are then compiled into a graph that can be used to find the optimal values for the system. It is simple and effective for a range of applications, but it is restricted to relatively basic preferences.

Another proposal for preferences in general comes from (Brewka 2004) in which Brewka proposes a qualitative preference system that involves formulas organized into a *ranked knowledge base* in which formulas' relative importance is defined by a rank attached to each formula. This ranking concept is utilized by our system.

Moving into the application of preferences to planning, we first mention a use of the previous system to prioritize a set of goals in (Feldmann, Brewka, and Wenzel 2006). The authors apply the system to planning by employing an iterative strategy where they search for a solution and then

resume, searching for solutions that are strictly better than the best already found.

Delgrande et al. (Delgrande, Schaub, and Tompits 2006) propose a preference language which directly defines when one history is preferred to a second by way of formulas simultaneously involving elements of both histories. We've used their query language with some minor adaptations in our approach. The same authors, see also (Delgrande, Schaub, and Tompits 2007) have proposed a language that can define aggregate values of histories. This system can be used to define preferences similar to the unbounded preference orderings we employ in our system.

Finally, the work that our proposed preference specification system is most influenced by is presented in (Son and Pontelli 2006) and (Bienvenu, Fritz, and McIlraith 2006). Son and Pontelli present a language for specifying preferences, $\mathcal{PP}$, and then compile these preferences into an answer set program that computes preferred plans. In their paper, Bienvenu et al. propose the language $\mathcal{LPP}$ which extends $\mathcal{PP}$. They then use $\mathcal{LPP}$ in a progression planner that makes use of an admissible optimistic heuristic to find the most preferred plan with respect to a preference specification. Both languages use *basic desire formulas* (BDFs) which describe properties of a history. They then place BDFs into orderings they call *atomic preference formulas* (APFs) which specify which BDFs are more preferable than others. $\mathcal{LPP}$ attaches values to the BDFs in an APF that define how preferred they are relative to one another. $\mathcal{PP}$ does not. The next level is that of *general preference formulas* (GPFs) that provide methods of combining APFs. Finally, $\mathcal{LPP}$ includes another level of *aggregated preference formulas* (AgPFs) that can relate the relative importance of GPFs.

Preliminaries

We use the notation from (Delgrande, Schaub, and Tompits 2006) that is an adaptation of (Gelfond and Lifschitz 1998) to define action signatures and histories.

Definition 1 *An action signature Σ is a quadruple (D, F, V, A), where D is a set of value names, F is a set of fluent names, $V : F \rightarrow 2^D \setminus \emptyset$ assigns a domain to each fluent and A is a set of action names.*

Σ is propositional iff $D = \{0, 1\}$ and is finite iff D, F, and A are finite. A fluent $f \in F$ is propositional iff $V(f) = \{0, 1\}$.

Similar to non-fluents in (Bienvenu, Fritz, and McIlraith 2006) we will define a subset of F, F_C whose values will be constant. F_N will refer to the subset of non-constant fluents, $F_N \cup F_C = F$.

Definition 2 *Let $\Sigma = (D, F, V, A)$ be an action signature. A history H of length n, over Σ is a sequence*

$$(s_0, a_1, s_1, \ldots, s_{n-1}, a_n, s_n)$$

where $n \geq 0$, and

- *each state, $s_i, 0 \leq i \leq n$, is a mapping assigning each fluent $f \in F$ a value $v \in V(f)$, and*
- $a_1 \ldots a_n \in A$

The states of a history may be thought of as possible worlds, and the actions take one possible world into another. For a propositional action signature $\Sigma = (D, F, V, A)$, fluent $f \in F$ is said to be $true$ at state s iff $s(f) = 1$, otherwise f is $false$ at s.

The Preference System

The preference system is structured similarly to the languages $\mathcal{PP}$ and $\mathcal{LPP}$, though different terms are attached to the constructs used along with more significant changes. At the lowest level we have *queries* which express properties of a history. The next level of *preference orderings*, or simply orderings, express which queries are more desirable to satisfy than others. The top level is a single *ranked preference specification*, which places orderings into a ranked knowledge base which defines which orderings are more important than others.

Queries

Queries are the building blocks of our preference specification system. Queries are simply formulas that when applied to a history evaluate to $true$ or $false$. They do not express preferences in their own right. They simply define properties of a history. If planning a vacation, a user might prefer to travel to a city that has a professional baseball team. In this case a query needs to be written that formalizes this concept, evaluating to $true$ when applied to a history containing travel to an appropriate city and $false$ otherwise.

The preference system we propose is defined to allow different languages to be used at the query level. Thus, our system can be applied to areas other than planning. Depending on the application, different query specification languages may be appropriate. For planning, we use a language that describes properties of histories. Here we are going to use the language from (Delgrande, Schaub, and Tompits 2006), with some minor modifications.

Our language is a first order sorted language. We need to define temporal preferences and will have time-stamp variables that can take on the values of the timepoints in a history as well as object variables that range over the objects in a domain. We forgo the additional notation to identify the two sorts as context makes it sufficiently clear. We will assume that there is a finite set of timepoints and objects and thus any formula involving quantified variables can be reduced to a propositional logic formula.

Definition 3 *Let $\Sigma = (D, F, V, A)$ be an action signature and $n \geq 0$ a natural number.*

We define the query language as follows:

1. *The alphabet consists of*
   (a) *a set $\mathcal{V}$ of variables,*
   (b) *the set of integers,*
   (c) *the arithmetic function symbols '$+$' and '$\cdot$',*
   (d) *the arithmetic relation symbol '$<$',*
   (e) *the equality symbol '$=$',*
   (f) *the set $A \cup F \cup D$ of action and fluent names and values,*
   (g) *the sentential connectives '$\neg$' and '$\supset$',*

(h) the quantifier symbol '$\exists$', and

(i) the parentheses '(' and ')'.

2. *A* time term *is an arithmetic term recursively built from $V \cup \{0, \ldots, n\}$, employing $+$, $-$, $\cdot$, and parentheses in the usual manner.*
   A time atom *is an arithmetic expression of form $(t_1 < t_2)$ or $(t_1 = t_2)$, where t_1, t_2 are time terms.*
3. *A* fluent value term *is either a member of D, a member of F_C, an expression of the form $f(t)$, where $f \in F_N$ and t is a time term, or an arithmetic term recursively built from value terms employing $+$, $-$, $\cdot$, and parentheses in the usual manner.*
   A fluent value atom *is an expression of the form $(v_1 = v_2)$ or $(v_1 < v_2)$, where v_1 and v_2 are value terms.*
   An action atom *is an expression of form $a(t)$, where $a \in A$ and t is a time term.*
   The set of atoms *is made up of the set of time atoms, fluent value atoms, and action atoms. An atom containing no variables is* ground.
4. *A* literal *is an atom, or an atom preceded by the sign $\neg$.*
5. *A* formula *is a Boolean combination of atoms, along with quantifier expressions of form $\exists i$, for $i \in \mathcal{V}$, formed in the usual recursive fashion.*
   A formula containing no free variables is closed.
6. *A* query *is a closed formula.*

Beyond the primitive operators above, we define the operators $\vee$ and $\wedge$, as well as the universal quantifier $\forall$ in the usual manner. Next is the definition for the semantics of the language, defining when a query is true with respect to a given history.

Definition 4 *Let $H = (s_0, a_1, s_1, \ldots, a_n, s_n)$ be a history over Σ of length n,*
We interpret the language recursively as follows:

1. *If Q is a ground time atom or a ground fluent value atom, then $H \models_\Sigma Q$ iff Q is true according to the rules of integer arithmetic.*
2. *If t is a ground time term t takes its value according to the rules of integer arithmetic.*
3. *If v is a ground fluent value term:*
   (a) if $v = f \in F_N$, then $v = s_0(f)$
   (b) if $v = f(t)$ where $f \in F_C$ and t is a time term, then $v = s_t(f)$ if $0 \leq t \leq n$ and is undefined otherwise.
   (c) if v is an arithmetic term then v takes its value according to the rules of arithmetic.
4. *If $Q = a(t)$ is a ground action atom, then $H \models_\Sigma Q$ iff $a = a_t$, if $0 \leq t \leq n$ and is undefined otherwise.*
5. *If $Q = \neg\alpha$, then $H \models_\Sigma Q$ iff $H \not\models_\Sigma \alpha$.*
6. *If $Q = (\alpha \supset \beta)$, then $H \models_\Sigma Q$ iff $H \not\models_\Sigma \alpha$ or $H \models_\Sigma \beta$.*
7. *If $Q = \exists i\, \alpha$ for a time-stamp variable α, then $H \models_\Sigma Q$ iff, for some $0 \leq m \leq n$, $H \models_\Sigma \alpha[i/m]$.*

A fluent value term of the form $f(t)$ or action atom, $a(t)$, is considered undefined if the time term t is either negative or beyond the end of the history. If a query is written that depends on one of these cases, producing a value of $true$ or $false$ could cause unexpected behavior. We avoid this with the undefined value. Any arithmetic that involves an undefined value is itself undefined. Boolean operators on undefined operands can be understood by considering undefined to be the value 0.5 with $true$ and $false$ taking their usual values of 1 and 0. Then $\alpha \vee \beta = max(\alpha, \beta)$, $\alpha \wedge \beta = min(\alpha, \beta)$, and $\neg\alpha = 1 - \alpha$.

Some examples of queries in the vacation domain:

- Visit Chicago at some time.

$$\exists t\, atCity(Chicago, t) \tag{1}$$

- Visit Paris at some time and London at a later time.

$$\exists t_1\, \exists t_2\ (t_1 < t_2)\ \wedge \\ atCity\,(Paris, t_1) \wedge\ atCity\,(London, t_2) \tag{2}$$

- Never visit a city in the desert.

$$\forall x\, \forall t\, \neg(atCity(x, t) \wedge inDesert(x)) \tag{3}$$

- Visit a city with a baseball team.

$$\exists x\, \exists t\, atCity(x, t) \wedge hasBaseballTeam(x) \tag{4}$$

Preference Orderings

Now that queries can be specified, we need a structure to specify which queries are preferred to others. For this purpose we next define preference orderings. Preference orderings define an ordering of alternative queries, each alternative being more preferred than the next. There will be two kinds of orderings, basic and unbounded. We begin with the basic type.

Definition 5 *A* basic preference ordering *o is a formula of the form, $\alpha_0 \gg \alpha_1 \gg \ldots \gg \alpha_n$ where $\alpha_i, 0 < i < n$ are queries with $n \geq 1$.*

The lowest indexed, and therefore most preferred, query that a history satisfies will be used to compare it to other histories. If a history does not satisfy any query in a basic ordering, then it is not comparable to other histories with respect to that ordering.

Definition 6 *History H_1 is at least as preferred as H_2 w.r.t. the basic preference ordering o, $\alpha_0 \gg \alpha_1 \gg \ldots \gg \alpha_n$, written $H_1 \geq_o H_2$, iff*

$$H_1 \models \alpha_i,\ H_1 \not\models \alpha_t,\ 0 \leq t < i, \\ H_2 \models \alpha_j,\ H_2 \not\models \alpha_t,\ 0 \leq t < j, \\ i < j.$$

Here is an example that expresses a desire to visit New York most, followed by Chicago, followed by Detroit. (Queries are given names representative of their meanings for the sake of brevity)

$$visitNewYork \gg visitChicago \gg visitDetroit \gg \\ true$$

The $true$ query at the least preferred position means that the three cities mentioned are preferred to all other cities. The $true$ and $false$ literals can be used as the least preferred query to signify whether or not all plans can be compared

w.r.t. that ordering. If the formula $true$ is the least preferred query then it will be satisfied by all plans and all plans will have a value for comparison. If $false$ is used instead, plans that fail to satisfy all other queries in the ordering will be incomparable with all other plans w.r.t. that ordering. The $false$ query can be omitted without changing the meaning of a query. A common situation where this is useful occurs with conditional preferences. If we prefer β to $\neg\beta$ only when α is satisfied, the ordering $\alpha \wedge \beta \gg \alpha \wedge \neg\beta \gg false$ is sufficient. Adding a query of $true$ would add a preference of α over $\neg\alpha$.

A flexible aspect of the orderings is that it allows simple construction of systems where transitivity holds or doesn't hold. If a system is desired where $\alpha \gg \beta$ and $\beta \gg \gamma$, but $\alpha \not\gg \gamma$, it can be done by creating two separate orderings, $\alpha \gg \beta \gg false$ and $\beta \gg \gamma \gg false$. Otherwise, the single ordering $\alpha \gg \beta \gg \gamma \gg true$ can be used.

A limitation of basic preference orderings is seen if we consider a preference to maximize the number of states when a fluent is true. Queries can be written that are satisfied if the fluent is true 0 times, 1 time, 2 times, etc.. The problem is that a basic preference ordering such as this would be difficult to create and compute due to its size. Such a preference represents a theoretically unbounded preference ordering. To deal with such preferences we present a construct to formally represent them in a compact manner.

Definition 7 *An* unbounded preference ordering*, o, takes the form $max/min\,(V, \sigma, q, opt)$ where V is a set of variables, σ is a time or fluent value term possibly involving members of V, q is a query involving all variables in V and opt is an integer or fluent value. Define Θ as the set of all grounding substitutions for V. Given a history H, $o(H)$ will represent the sum:*

$$\sum_{\{\theta \in \Theta |\ H \models q(\theta)\}} \sigma(\theta) \tag{5}$$

The value $o(H)$ for an unbounded ordering is computed by finding all substitutions over all variables in V that cause q to be satisfied. This set of substitutions is applied to σ and the resulting values are summed. The value opt defines a cutoff beyond which all values are considered optimal. Note that variables in V are each of a specific sort, time-stamp or object in our query language, and Θ's substitutions necessarily follow the sorts (a time-stamp will never be substituted in place of an object variable and vice versa).

A preference between two histories is determined by comparing their sum values. If both are better than the optimal value cutoff then they are considered equally preferred w.r.t. the unbounded ordering.

Definition 8 *Given a maximizing unbounded ordering o, H_1 is at least as preferred as H_2, written $H_1 \geq_o H_2$, iff*

$$o(H_1) > o(H_2),\ o(H_2) < opt$$

Interpretation of a minimizing order is similarly defined by reversing the inequalities.

Again thinking of the vacation domain, let's say we want to maximize the number of states where we are in Chicago. We can do so with the following unbounded ordering.

$$max(\{t\}, 1, atCity(Chicago, t), 4) \tag{6}$$

Given a history only time-stamps for which $atCity(Chicago, t)$ is $true$ will be applied to the variable t. Since $\sigma = 1$ the preference will simply count the number of times this occurs. The optimal value is 4, so if a history has four or more appropriate states, then it is optimal w.r.t. the ordering.

The inclusion of this structure will save space in defining preferences as well as minimizing necessary computation. To illustrate this, consider a user who wants to maximize the number of states where the fluent f is true. With knowledge of the longest allowed plan this could be done by creating a query for each possible number of occurrences from 0 to n. The formula for f being true in exactly two states is as follows.

$$\begin{aligned}\exists t_1\, \exists t_2\, \forall t_3\, (t_1 \neq t_2 \wedge t_2 \neq t_3 \wedge t_1 \neq t_3) \supset \\ (f(t_1) \wedge f(t_2) \wedge \neg f(t_3))\end{aligned} \tag{7}$$

Similar formulas can be written for any value up to n, but quickly become unwieldy. Although we are considering a very simply preference using a basic preference ordering to represent it is quite difficult. However, when expressed with the unbounded ordering structure you get a preference such as (6) which is not only simpler to define, it is simpler computationally. If the ordering were defined by writing out each query in a basic ordering, then it becomes possible to have to check each and every formula to evaluate a history w.r.t. the ordering. In many examples, including this one, calculating the value of the unbounded preference will require drastically less computation than for a similar basic ordering. To evaluate this unbounded preference, one variable ranges over every time-stamp. In formulas in an equivalent basic ordering, multiple variables may have to range over all time-stamps.

Here is a more complex unbounded preference ordering that aims to maximize the maximum value of a non-propositional fluent f over a history. We have to be careful here so that only one substitution for the variable gives the maximum value. Therefore the equation specifies the earliest occurrence of this maximum value.

$$\begin{aligned}max(\{t\}, f(t), \quad & \forall t_1\, (t_1 \neq t \wedge f(t_1) < f(t)) \vee \\ & (f(t_1) = f(t) \wedge t < t_1) \\ & , 1000)\end{aligned} \tag{8}$$

Now, a final example that involves more than one variable. If we are in a domain that involves loading trucks for delivery, a preference to minimize the unused space over all of the trucks and states may be desired. If the fluent $unusedSpace(x, t)$ defines the amount of unused space for truck x in state t we can use the following.

$$\begin{aligned}min(\{x, t\}, \quad & unusedSpace(x, t), \\ & unusedSpace(x, t) > 0, 0)\end{aligned} \tag{9}$$

One preference that is commonly assumed in planning with and without preferences is that a shorter plan is preferred. A preference for shorter plans can be defined with an unbounded ordering. Thus we have a natural way of defining this preference instead of assuming it. We will also be able to define the importance of this preference w.r.t. other preferences by using the ranking system described below instead of always using it as a tie-breaker in cases where the other preferences are satisfied equally.

Unbounded orderings have been defined to maintain the flexibility of the preference system. They will be applicable to query languages that support first order notation, but if the query language doesn't support first order notation, then unbounded orderings won't be appropriate.

Preference orderings only define qualitative preferences between histories. We say that one history is preferred to another if it satisfies a more preferred query, or has a better value with an unbounded ordering, but we don't look to the size of the difference to provide any meaning.

Ranked Preference Specification

We've now established a query language and a structure to define preference orderings among queries. We now want to be able to specify relative importance among orderings. One ordering may represent a preference over which city to visit while another represents a preference over method of travel. It is natural for a user to think that one preference's satisfaction is more important than another. Maybe they prefer travel by train to airplane, but it isn't as important as making it to a preferred city.

Given a group of preference orderings, relative importance between these orderings will be defined by attaching ranks in the manner that ranks are attached to formulas in ranked knowledge bases (RKBs) as seen in (Brewka 2004). Orderings will have a rank assigned to them represented by a non-negative integer, thus organizing them into a *ranked preference specification* (RPS). We can consider a RPS as a group of sets where each set consists of all orderings assigned a particular rank.

Definition 9 *A* ranked preference specification *consists of a group of sets of orderings* $(O_0, O_1, \ldots O_n)$. *Ordering* o *is at least as important as* o' *iff* $o \in O_i$, $o' \in O_j$, *and* $i \leq j$.

If in our example there were three preference orderings that represented preference over which city to visit, C, how to travel, T, and what attractions to visit, A, and the city to visit and attractions visited are most important while method of travel is less so, an appropriate RPS would be:

$$O_0 = \{C, A\}, O_1 = \{T\}$$

Note that increasing ranks represent decreasing importance of the orderings at those ranks. As with the ordering indices, these ranks represent purely qualitative difference in importance. An ordering in a lower rank is simply considered more important, but there is no meaning in the numeric values of the ranks beyond that.

Our system does not specifically define when one history is preferred to another with respect to a RPS. The strategy we use is described in the next section, but varying strategies can be created given the information in a RPS. Depending on the strategy used, the meaning of the ranks could change, however, the intention in our definition is that if one history is preferred to another w.r.t. the orderings at a given rank, then it would be preferred irrespective of how the histories satisfy orderings in less important ranks.

Although the system is qualitative it is possible to create a RPS that embodies a meaning that is associated with using quantitative values. Let's assume a user has a choice between visiting Chicago, Montreal, and Detroit and a choice between traveling by bus or train. If a user strongly prefers a trip to Montreal over Chicago or Detroit they might give Montreal a value of 8, Chicago, 4, and Detroit, 3. They might then give travel by train a 6 and bus a 4. Traveling to Montreal is clearly the most important and it might initially seem difficult to use our qualitative system to express it.

However, if instead of using a single preference ordering to represent the preference among cities, two orderings are used, it becomes possible. The following RPS is sufficient.

$$O_0 = \{visitMontreal \gg true\}$$
$$O_1 = \{visitChicago \gg visitDetroit \gg true, \\ rideTrain \gg rideBus \gg true\}$$

The preference for Montreal is clearly made stronger by moving it to a more important rank. If there are queries in an ordering that a user considers significantly more or less preferred than others, this can be represented by breaking an ordering into multiple pieces as was done with this example.

Defining Preferred Histories

We now have a system by which a user can describe their preferences. With the ranks of preference orderings and levels of satisfaction within the orderings there are numerous options on how to do this. Depending on the application or setup of the preferences, one method of comparison may be more appropriate than another. It may be more important to emphasize satisfaction of the entire set of preferences at a given rank or focus more on individual preferences. It might be more important to satisfy the most preferred queries in an ordering or to strongly avoid the least preferred queries. Some general strategies for doing this are discussed in (Brewka 2004). They would need modification before being applied to RPSs, but the strategies are relevant.

We will use a straightforward cardinality based approach to compare histories. Given two histories, we will count how many times each history is preferred to another w.r.t. the orderings at a given rank. Whichever has more 'victories' at a given rank is preferred. If they tie, then the orderings at the next rank are examined. If the histories are tied in the end, then they are considered equally preferred.

Definition 10 *Given histories* H_1 *and* H_2 *and RPS* P, H_1 *is at least as preferred as* H_2, $H_1 \succeq_P H_2$, *if for some* i

$$\left|\{o \,|\, o \in O_i, H_1 \geq_o H_2\}\right| > \left|\{o \,|\, o \in O_i, H_2 \geq_o H_1\}\right|$$

and for all j, $0 \leq j < i$,

$$\left|\{o \,|\, o \in O_j, H_1 \geq_o H_2\}\right| = \left|\{o \,|\, o \in O_j, H_2 \geq_o H_1\}\right|$$

Planning: Producing Preferred Histories

Now we turn to the process of producing preferred histories. We begin by defining the problem. We will use STRIPS style actions as originally set forth by Fikes and Nilsson (Fikes and Nilsson 1971). Our formalization is similar to that of (Gelfond and Lifschitz 1998).

Definition 11 *A STRIPS action A is a pair (F, X) where F is a query describing the conditions under which A is executable and X is a set of possibly negated fluent value terms describing the consequences of A.*

A STRIPS action is executable if its precondition query holds and the resulting state is reached by adding/removing the positive/negative literals in X.

Definition 12 *Given a STRIPS action A, (F, X), and states s and s' the execution of A causes a transition from s to s' iff*

$$\begin{aligned} s &\models F \quad , \\ s' &\models v \quad \textit{iff} \quad v \in X, \textit{or} \\ & \quad\; s \models v \textit{ and } \neg v \notin X \end{aligned}$$

A planning problem is defined as follows.

Definition 13 *A planning problem consists of an action signature Σ, a set of STRIPS actions, an initial state s_0, a goal query G, and a RPS P.*

Definition 14 *Given a planning problem, a plan p is a series of STRIPS actions, $a_0, a_1, \ldots, a_n$, that when iteratively applied to state s_0, every action is executable and conclude with final state s_n which satisfies the goal query.*

We will use H_p to represent the history produced by a plan p. Plan p_1 is at least as preferred as plan p_2 iff $H_{p_1} \succeq_P H_{p_2}$. We will define *optimal* and *k-optimal* plans as was done in (Bienvenu, Fritz, and McIlraith 2006).

Definition 15 *Given a planning problem, a plan p is optimal if for any other plan p', $H_p \succeq_P H_{p'}$. It is k-optimal if p's length is no greater than k and $H_p \succeq_P H_{p'}$ for all plans of length no greater than k.*

We have created a forward-chaining planner that can use a RPS to guide its search. The aim is to quickly produce an optimal plan. To do this the planner applies two heuristics to partial plans to guide a best-first search strategy. The first is an admissible optimistic heuristic that mirrors the heuristic used by the PPLAN planner in (Bienvenu, Fritz, and McIlraith 2006). The second heuristic is a non-admissible distance based heuristic.

For both of our heuristics, we need to define the notion of a partial plan's distance to satisfaction of a query. This will put a value on how far a partial plan is from satisfying a given query. To do this we compute the minimal number of fluent or action terms in a query that are currently unsatisfied that need to be satisfied for the query to be satisfied. If the query is already satisfied, the distance is 0, and if the query is unsatisfiable, the distance is infinity, ∞. Since we are evaluating this value for incomplete plans we must keep in mind that there will be at least one more state to follow the current final state.

Definition 16 *Given a query q, a partial plan's history of length n, H, and a maximum plan length k the value $dist(H, q)$ is recursively defined as follows:*

- *if $q = true$ $dist(H, q) = 0$,*
- *if $q = false$, $dist(H, q) = \infty$,*
- *if q is a time or fluent value atom, $dist(H, q) = 0$ if $H \models q$, ∞ otherwise,*
- *if $q = f or q = \neg f, f \in F_C$, $dist(H, q) = 0$ if $H \models q$, ∞ otherwise,*
- *if $q = A(t), F(t), \neg A(t)$, or $\neg F(t)$ and t is a time term,*
  - *if $0 \leq t \leq n$: $dist(H, q) = 0$ if $H \models q$, ∞ otherwise,*
  - *if $n < t \leq k$: $dist(H, q) = 1$,*
  - *if $t < 0$: $dist(H, q) = \infty$*
- *if $q = A(n), F(n), \neg A(n)$, or $\neg F(n)$, $dist(H, q) = 1$,*
- *if $q = \phi \vee \psi$, $dist(H, q) = min(dist(H, \phi), dist(H, \psi))$,*
- *if $q = \phi \wedge \psi$, $dist(H, q) = dist(H, \phi) + dist(H, \psi))$,*
- *if $q = \exists t\, \psi$, $dist(H, q) = \min_{0 \leq i \leq k}(dist(H, q^{i/t}))$,*
- *if $q = \forall t\, \psi$,*

$$dist(H, q) = \sum_{0 \leq i \leq min(n+1, k)} dist(H, \psi^{i/t})$$

Quantified object variables can be viewed as conjunctions or disjunctions over all substitutions. For quantified time variables we range over timepoints, even beyond the current endpoint. This embodies the notion that queries with universally quantified time variables will normally never be completely satisfied and queries with existentially quantified time variables will always have a chance to be satisfied. Constant fluents and fluents and actions with a time-stamp within the current history are either already satisfied, or will never be, and thus produce distances of 0 or ∞. Literals that have a time-stamp of n (representing the last state of a history) are considered to never be satisfied since the current final state will not be the final state of a complete plan's history. Negation can efficiently be moved to the literal level by use of DeMorgan's laws and therefore we only consider negation at that level.

Optimistic Heuristic

The optimistic heuristic follows the assumption that a node will lead to a plan that satisfies the most preferred query in an ordering that hasn't become unsatisfiable. In this way we have a heuristic that will never cause a node to be less preferred than one of its predecessors. A pessimistic value is also defined that assumes no query currently unsatisfied will become satisfied. This will be used to compare histories that are tied w.r.t. the optimistic values..

Definition 17 *Given a preference ordering o and an history H associated with a partial plan, $o_{opt/pess}(H)$is defined as follows:*

- *if o is a basic ordering,*
  - $o_{opt}(H) = i$ *if* $dist(H, \alpha_i) < \infty$, $dist(H, \alpha_j) = \infty, 0 \leq j < i$
  - $o_{pess}(H) = i$ *if* $dist(H, \alpha_i) = 0$, $dist(H, \alpha_j) > 0, 0 \leq j < i$

- *if o is an unbounded ordering,*
  - $o_{opt}(H) = opt$,
  - $o_{pess}(H) = o(H)$

Definition 18 *Given a preference ordering o and an history H associated with a plan, $o_{opt}(H) = o_{pess}(H)$ and are equal to $o_{pess}H$ as defined in Definition 17.*

The optimistic and pessimistic values for a complete plan are equal because the plan is complete, we know exactly how well it will satisfy the preference orderings. When comparing two plans, partial or complete, we will first compare their optimistic satisfaction of preferences. If necessary we then compare their pessimistic satisfaction. We use the strategy from Definition 10 when making the comparison.

Greedy Distance Heuristic

The second heuristic function is based directly on the distance function we've defined. It simply chooses which partial plan is preferred by how close it is to satisfying desirable queries. This is clearly a non-admissible heuristic as the values for a partial plan may increase or decrease as it is extended.

Definition 19 *Given a basic preference ordering o and a non-negative integer n and two histories, H_1 and H_2, H_1 is preferred w.r.t. the most preferred n queries of the ordering, $H_1 \geq_{o_{gdy(n)}} H_2$, if*

$$dist(H_1, \alpha_i) < dist(H_2, \alpha_i),\ i \leq n$$
$$dist(H_1, \alpha_j) = dist(H_2, \alpha_j) \neq 0,\ 0 \leq j < i$$

If $H_1 \geq_{o_{gdy(n)}} H_2$ then there is some query in the most preferred n queries in the ordering that H_1 is closer to satisfying and the two histories are equally close to satisfying, without satisfying, all more preferred queries. Therefore, $H_1 \geq_{o_{gdy(0)}} H_2$ if H_1 is closer to satisfying the most preferred query, $H_1 \geq_{o_{gdy(1)}} H_2$ if $H_1 \geq_{o_{gdy(0)}} H_2$ or H_1 is closer to satisfying the second query and they tie on the first.

The concept of distance to satisfaction doesn't apply to the unbounded preferences. For these, whichever produces the preferred number will win. Unbounded preferences that count occurrences, compute max and min will tend to only increase or only decrease over time. When this is the case this strategy should be effective.

Again using Definition 10 we will first compare two histories w.r.t. $\geq_{o_{gdy(0)}}$. If no preference is found we iteratively increase the depth until either one history is preferred, or every ordering is considered in its entirety.

One fine point comes up when dealing with basic preference orderings that allow the possibility of a history not satisfying any of the queries. Strategies could be devised that try to determine when to make an effort to satisfy them and when to ignore them. We choose to treat these preferences as all others and the search attempts to satisfy the most preferred queries in these orderings.

Implementation and Experiments

To test our algorithms we implemented a forward chaining planner in C++ that uses our heuristics to guide a best-first search. It maintains a frontier of nodes, each of which contains a partial plan. The frontier initially only contains the initial state. The planner then iteratively expands nodes by creating a new node for every action that can be executed from the node being expanded. All new nodes are then added to the frontier sorted using the selected heuristic. If two histories are equivalent with the currently used heuristic, then the shorter plan will favored. The basic algorithm is in figure 1.

Search Algorithm

```
add rootNode to frontier
while frontier is non-empty
    currentNode = best node from frontier
    [GRE]if currentNode can't beat best solution
    [GRE]     discard currentNode
    if currentNode is a solutionNode
        return currentNode
    if currentNode is not at maximum length
    for each action executable from currentNode
        create newNode
        if newNode is a solution
            [OPT] create solutionNode and add to frontier
            [GRE] if ideal: return solution
            [GRE] else: store solution
        add newNode to frontier
return NULL
```

Figure 1: Pseudocode for search algorithm. "[GRE]" and "[OPT]" mark lines only applicable when using the Greedy or Optimistic heuristics. Returning a node implies returning the plan associated with that node.

We tested our planner using a planning domain presented in (Bienvenu, Fritz, and McIlraith 2006) that involves planning a meal. We make use of test cases they used to test their planner, PPLAN in that paper. Although the test cases were presented using $\mathcal{LPP}$, translation to our RPS system is straightforward due to the similarities in the approaches. The results for PPLAN are taken from running on preferences specified $\mathcal{LPP}$.

As was done in (Bienvenu, Fritz, and McIlraith 2006) and (Kvarnström and Doherty 2000) we used domain-dependent control knowledge to improve the search. For example, a plan that includes driving to one location and immediately driving elsewhere without doing anything there is always undesirable. We accomplished this by simply adding two preference orderings at their own most preferred rank that are satisfied unless an obviously poor selection of actions has taken place. Since they are in the most preferred rank, any partial plan not satisfying these preferences is placed at the bottom of the frontier and will not be considered unless there is no goal-satisfying plan that doesn't violate them.

When the greedy heuristic is used we are able to ignore some nodes once a solution has been found. If the current node cannot optimistically be preferred over the best solution currently stored, there is no reason to expand it. It is therefore discarded.

We discovered an interesting issue when running experiments on the two search methods. The file that defined the planning domain had an unintended effect on the efficiency

#	Gre	Opt	PP	#	Gre	Opt	PP	#	Gre	Opt	PP
1	15	14	22	**21**	7	7	8	**41**	5	11	7
2	3	3	10	**22**	3	3	10	**42**	7	23	23
3	118	122	43	**23**	4	4	12	**43**	3	10	15
4	5	4	7	**24**	6	5	15	**44**	44	43	31
5	3	2	3	**25**	4	4	7	**45**	16	317*	36
6	27	28	19	**26**	18	18	22	**46**	3	12	7
7	6	6	8	**27**	58	19272	19137	**47**	15	7659	8157
8	5	4	9	**28**	4/169	169	169	**48**	21	11	15
9	2	2	3	**29**	3	3	10	**49**	7	36	85
10	7	7	7	**30**	2	2	3	**50**	5/340	340	340
11	4	4	12	**31**	8	11	11	**51**	4	3	2
12	3	3	10	**32**	12/166	163	163	**52**	7	11	11
13	4	4	15	**33**	4	6	15	**53**	2	2	3
14	7	6	8	**34**	6	14	22	**54**	3	3	10
15	6	6	16	**35**	55/169	169	169	**55**	3	20	10
16	20	20	29	**36**	5	6	23	**56**	17	15	119
17	5	5	10	**37**	22	206*	37	**57**	5	5	22
18	8	17	19	**38**	7	7	15	**58**	15	5	21
19	3	2	3	**39**	17	47	54	**59**	3	21	10
20	12	22	19	**40**	47/151	151	151	**60**	10	5	21

Figure 2: Table of test results. Gre: Greedy Heuristic, Opt: Optimistic Heuristic, PP: PPLAN, * marks results with a best/worst disparity of an order of magnitude or more

of the search. The order that the actions were defined in the file determined the order that actions were executed when expanding a node in the search-space. When the new nodes that are produced tie with respect to the heuristic function being used, the order they are produced effects the order that they end up in the frontier and thus the order they are expanded. We found that reordering the actions in the file could produce large differences in the number of nodes expanded when the search was run. The difference between the best and worst case can be as large as an order of magnitude. This problem appears in our own planner as well as with PPLAN. Due to this issue we have randomized the order that actions are used to expand a node. We produced our results by running the search 50 times and averaging the results. This unexpected variance from an unforeseen and preferably irrelevant factor is something that will need to be monitored in any similar planning experiments.

Although this issue exists we still give our results in comparison to those of PPLAN. This may not be ideal, but the variations due to action ordering are usually not serious. The comparison still gives a general idea of success. We use the number of nodes expanded as the basis for the comparison. In the case of the greedy heuristic there are sometimes two numbers. The first is the number of nodes expanded before finding the optimal solution. The second is the total number of nodes expanded before the algorithm halted. The numbers are different when the optimal plan doesn't optimally satisfy each and every ordering. Therefore, the planner needs to continue until it can guarantee the optimal solution has been found.

The results for our tests are displayed in figure 2. It is apparent that both of our heuristics and PPLAN often perform equally well. The most successful heuristic is our non-admissible greedy function which was rarely defeated by a significant margin and occasionally was one or more orders of magnitude better than the optimistic heuristics. Our optimistic heuristic applied to our system performs well in comparison to PPLAN either expanding fewer nodes or a similar number of nodes in the majority of cases. In two cases the best test run was an order of magnitude better than the worst test run. In these test cases, the best run was similar to that of PPLAN. These are marked with an asterisk.

Discussion and Related Work

We have proposed a preference specification system that can express temporal preferences with respect to histories. It is flexible, allowing users to alter some aspects of the system so it can be applied to any field where preferences are desirable. We will now compare it with some of the preference systems/languages mentioned earlier.

As previously demonstrated, the conditional preferences of CP-nets (Boutilier et al. 2004) can easily be encoded in our language. However, CP-nets are unable to represent the complex first-order temporal preferences of our language. The work with ranked knowledge bases from (Brewka 2004) and (Feldmann, Brewka, and Wenzel 2006) However, it applies rankings to formulas as opposed to the orderings proposed here. As such it would be extremely difficult to represent the preferences that can be defined with the orderings proposed here.

Recall that Son and Pontelli proposed their language,$\mathcal{PP}$ (Son and Pontelli 2006) and Bienvenu et al. (Bienvenu, Fritz, and McIlraith 2006) made some additions and changes to create their language $\mathcal{LPP}$. We will compare their languages to ours level by level. At the query or BDF level, $\mathcal{PP}$ and $\mathcal{LPP}$ use *linear temporal logic* (LTL) with only $\mathcal{LPP}$ making use of first order logic. Such languages can be used in place of our query language, but they lack timestamp variables and would thus make our unbounded ordering structure less functional. APFs are almost identical to our basic preference orderings, though we don't attach values to queries as in $\mathcal{LPP}$. Also, $\mathcal{PP}$ and $\mathcal{LPP}$ don't allow histories to ever be incomparable with respect to an APF. Neither language can specify preferences similar to our unbounded orderings. GPFs include disjunction, conjunction, and conditional means to combine APFs. These can be represented in our system, though conjunctions may lead to exponential blow-up in translation. AgPFs can be implemented in our system with a combination of the ranks attached to orderings and adjusting the strategy for calculating preferences among histories. We believe our system of ranks is more intuitive.

The other separation between our work and (Bienvenu, Fritz, and McIlraith 2006) is in the strategy used for defining and producing preferred plans. Their system is generally qualitative, but it becomes partially quantitative when they attach numeric values to the formulas in an APF. In their planner they use a strategy that uses these values as weights in the comparison of plans. Our aim is to see how successful a system can be without any quantitative aspects creeping in. Though this may lead to a larger number of equivalently preferred histories, in those cases the specification simply

hasn't provided the information necessary to distinguish between them.

Delgrande et al. (Delgrande, Schaub, and Tompits 2007) (Delgrande, Schaub, and Tompits 2006) present a system from which our query language is based. They build a preference language on top of the query language. The primary advantage of our system is in the orderings which allow preferences among several alternatives to easily by specified. Their language also lacks a mechanism to rank the importance of preferences.

The most novel element of our RPS system is the unbounded ordering construct. The only proposal we are aware of that can represent similar preferences is presented by Delgrande et al. (Delgrande, Schaub, and Tompits 2007). Their methodology involves defining aggregate values by means of a recursive macro system. If this approach was added to the preference system proposed in this paper, it would fall into the query level. It would then still be necessary to have a construct for creating an unbounded ordering. Since the unbounded ordering construct allows these values to be specified on its own, such a strategy is not needed in our system. Also worth mentioning is PDDL3, a language used in planning competitions, that can express a portion of those unbounded orderings that only involve object variables in V (Gerevini and Long 2006).

To summarize, the RPS system is flexible. It can be applied to many applications with the use of other query languages. Preference orderings can be defined that enforce or violate transitivity and that give comparisons between all or only a subset of all alternatives (histories in this paper). Varying strategies for using the information in an RPS to compare alternatives can be used to suit a user's needs. Unbounded orderings allow preferences that are undefinable or unmanageable in most other systems. Finally, the system is purely qualitative, not relying on precise values to define preferences or compare alternatives.

Though more experimental results need to be explored, our results are promising. Firstly, they give evidence that the move away from the values attached to formulas in APFs doesn't hurt our system in terms of efficiently finding optimal plans. Secondly, we showed that a non-admissible heuristic may be the best option. The greedy heuristic tended to find optimal solutions more quickly than the optimistic heuristics. If the solution is ideal it is able to immediately halt, however, even when the optimal plan is not ideal it tended to expand no more nodes than the admissible optimistic heuristics. The reason for this is that we use the best solution found to prune nodes that can't optimistically beat this solution. This breaks down to the same strategy the optimistic heuristics use for pruning non-optimal branches in the search tree. Baier et al. (Baier, Bacchus, and McIlraith 2007) and (Baier and McIlraith 2007) have done more extensive work using heuristic strategies that often calculate distance metrics to preferences and goals and experiment with them in different combinations demonstrating strong results.

Future work needs to go into testing these and other heuristics in a wider range of planning problems and domains as well as experiments that place focus on unbounded orderings. We were also only concerned with the efficiency of our heuristics and did not put time into implementing our planner in an efficient manner. Making use of strategies being used elsewhere in planning with our system should be explored in the future.

References

Baier, J. A., and McIlraith, S. 2007. On domain-independent heuristics for planning with qualitative preferences. In *Proceedings of the 7th IJCAI International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC-07)*.

Baier, J.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1808–1815.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, 134–144. Lake District, UK: AAAI Press.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res. (JAIR)* 21:135–191.

Brewka, G. 2004. A rank based description language for qualitative preferences. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI04)*, 303–307. IOS Press.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2006. An extended query language for action languages (and its application to aggregates and preferences). In *Eleventh International Workshop on Non-Monotonic Reasoning NMR2006*.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2007. A general framework for expressing preferences in causal reasoning and planning. *J. Log. Comput.* 17(5):871–907.

Feldmann, R.; Brewka, G.; and Wenzel, S. 2006. Planning with prioritized goals. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, 503–514. Lake District, UK: AAAI Press.

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.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electron. Trans. Artif. Intell.* 2:193–210.

Gerevini, A., and Long, D. 2006. Plan constraints and preferences in `PDDL3`. In *Proceedings of the ICAPS 2006 Workshop on Preferences and Soft Constraints in Planning*.

Kvarnström, J., and Doherty, P. 2000. Talplanner: A temporal logic based forward chaining planner. *Ann. Math. Artif. Intell.* 30(1-4):119–169.

Son, T. C., and Pontelli, E. 2006. Planning with preferences using logic programming. *TPLP* 6(5):559–607.

Preferred answer sets supported by arguments

Ján Šefránek
Comeniu University, Bratislava, Slovakia
e-mail: sefranek@ii.fmph.uniba.sk

Abstract

We are aiming at a semantics of prioritized logic programs which always selects a preferred answer set, if there is a non-empty set of (standard) answer sets of the given program.

It is shown in a seminal paper by Brewka and Eiter that the goal mentioned above is incompatible with their second principle and it is not satisfied in their semantics of prioritized logic programs. Similarly, also according to other established semantics, based on a prescriptive approach, there are prioritized logic programs with standard answer sets, but without preferred answer sets.

Our solution is as follows. According to the standard prescriptive approach no rule can be fired before a more preferred rule, unless the more preferred rule is blocked. This is a rather imperative approach, in its spirit. In our approach, rules can be blocked by more preferred rules, but the rules which are not blocked are handled in a more declarative style, their execution does not depend on the given preference relation on the rules.

An argumentation framework is proposed in this paper. Argumentation structures are derived from the rules of a given program. An attack relation on argumentation structures is defined, which is derived from attacks of more preferred rules against the less preferred rules. Preferred answer sets correspond to complete argumentation structures, which are not attacked by another complete argumentation structures.

Keywords: extended logic program, answer set, preference, prioritized logic program, preferred answer set, argumentation structure

Introduction

Background Meaning of a nonmonotonic theory is often characterized by a set of *(alternative) belief sets*. It is natural and appropriate to select sometimes some of the belief sets as more *preferred*.

We are focused in this paper on *extended logic programs* with a preference relation on rules (and with a kind of answer set semantics), see f.ex. (Brewka and Eiter 1999; Delgrande, Schaub, and Tompits 2003; Schaub and Wang 2001; Wang, Zhou, and Lin 2000). Such kind of programs is denoted by the term *prioritized* logic programs in this paper.

An investigation of basic *principles* which should be satisfied by any system which is based on prioritized defeasible rules is of fundamental importance. This type of research has been initialized in the seminal paper (Brewka and Eiter 1999). Two basic principles are accepted in the paper.

Problem It is natural to require that some preferred answer sets can be *selected* from a non-empty set of standard answer sets of a (prioritized) logic program.

Unfortunately, there are prioritized logic programs with standard answer sets, but without preferred answer sets according to the semantics of (Brewka and Eiter 1999) (and also of (Delgrande, Schaub, and Tompits 2003) or (Wang, Zhou, and Lin 2000)). This feature is a consequence of the *prescriptive* (Delgrande et al. 2004) approach to preference handling. According to that approach, the preference relation defined on the rules of the given prioritized logic program specifies the order in which rules are to be applied.

Moreover, the second of the principles accepted by (Brewka and Eiter 1999) is violated, if a function is assumed, which selects a non-empty subset of preferred answer sets from a non-empty set of all standard answer sets of a prioritized logic program. See Proposition 6.1 of (Brewka and Eiter 1999).

Goal and proposed solution We believe that the possibility to select always a preferred answer set from a non-empty set of standard answer sets is of critical importance. This goal requires to accept a *descriptive* approach to preference handling. The approach is characterized by (Delgrande and Schaub 2000; Delgrande et al. 2004) as follows: The order in which rules are applied, reflects their "desirability'. We attempt to precise in conclusions the sense in which the term is used in this paper.

Our goal is:

1. to modify the understanding of Principles proposed by (Brewka and Eiter 1999) in such a way that they do not contradict a selection of a non-empty set of preferred answer sets from the underlying non-empty set of standard answer sets,
2. to introduce such a notion of preferred answer sets which enables a selection of a preferred answer set from a non-empty set of standard answer sets.

The proposed solution is sketched as follows. A notion of argument and argumentation structure is introduced. The

notions are inspired by (García and Simari 2004), but they are rather different. The basic argumentation structures correspond to the rules. Some derivation rules are defined for argumentation structures. The set of argumentation structures is closed w.r.t. the rules. A transfer from a preference on rules to a preference on arguments is suggested. Attacks of more preferred rules against the less preferred rules are transferred via another set of derivation rules to the attacks of more preferred arguments against the less preferred arguments. Arguments attacked by more preferred arguments are called blocked. Preferred answer sets are defined in terms of complete and non-blocked arguments.

According to the prescriptive attitude towards prioritized logic programs no rule can be fired before a more preferred rule, unless the more preferred rule is blocked. This is a rather imperative approach, in its spirit. In our approach, rules can be *blocked* by more preferred rules, but the rules which are not blocked are handled in a more declarative style. The execution of non-blocked rules does not depend on their order. Dependencies of more preferred rules on less preferred rules does not prevent the execution of non-blocked rules in our approach. Of course, this approach is computationally more demanding than the prescriptive approach. In (Delgrande, Schaub, and Tompits 2003) a compilation of prioritized programs to extended programs is proposed. The standard answer sets of the "output" program are equivalent - modulo new symbols - to the preferred answer sets of the original prioritized program. We are not aware of a similar result for an approach based on the descriptive attitude.

Finally a remark – this paper is a result of a (rather unsuccessful) attempt to modify approaches of (Brewka and Eiter 1999), (Delgrande, Schaub, and Tompits 2003) and (Wang, Zhou, and Lin 2000) in a way, which enables a selection of a preferred answer set from a non-empty set of standard answer sets.

Main contributions Contributions of this paper are summarized as follows. A modified set of principles for preferred answer sets specification is proposed. A new argumentation framework is constructed, which enables a selection of preferred answer sets. Rules for derivation of argumentation structures and rules for derivation of attacks of some argumentation structures against other argumentation structures are defined. Preferred answer sets are defined in terms of complete and non-blocked argumentation structures. It is proven that our notion of preferred answer sets satisfies specified principles, see Theorems 37,38,39. Finally, we emphasize that each program with a non-empty set of answer sets has a preferred answer set.

Preliminaries

The language of extended logic programs is used in this paper.

Let At be a set of atoms. The set of *objective literals* is defined as $Obj = At \cup \{\neg\ A : A \in At\}$. If L is an objective literal then the expression of the form $not\ L$ is called *default* literal. Notation: $Def = \{not\ L \mid L \in Obj\}$. Sets of default literals are called *assumptions* in this paper. The set of literals Lit is defined as $Obj \cup Def$. A convention: $not\ not\ L = L$, $\neg\neg A = A$, where $L \in Obj$ and $A \in At$. If X is a set of objective literals, then $not\ X = \{not\ L \mid L \in X\}$.

A *rule* is each expression of the form $L \leftarrow L_1, \ldots, L_k$, where $k \geq 0$, $L \in Obj$ and $L_i \in Lit$. If r is a rule of the form as above, then L is denoted by $head(r)$ and $\{L_1, \ldots, L_k\}$ by $body(r)$. A finite set of rules is called *extended logic program* (program hereafter).

The set of *conflicting literals* is defined as $CON = \{(L_1, L_2) \mid L_1 = not\ L_2 \vee L_1 = \neg L_2\}$. A set of literals S is *consistent* if $(S \times S) \cap CON = \emptyset$. An *interpretation* is a consistent set of literals. A *total* interpretation is an interpretation I such that for each objective literal L either $L \in I$ or $not\ L \in I$. A literal L is *satisfied* in an interpretation I iff $L \in I$. A set of literals S is satisfied in I iff $S \subseteq I$. A rule r is satisfied in I iff $head(r)$ is satisfied in I whenever $body(r)$ is satisfied in I.

If S is a set of literals, then we denote $S \cap Obj$ by S^+ and $S \cap Def$ by S^-. Symbols $body^+(r)$ and $body^-(r)$ are used here in that sense (notice that the usual meaning of $body^-(r)$ is different). If $X \subseteq Def$ then $pos(X) = \{L \in Obj \mid not\ L \in X\}$. Hence, $not\ pos(body^-(r)) = body^-(r)$. If r is a rule, then the rule $head(r) \leftarrow body^+(r)$, is denoted by r^+.

Answer set of a program can be defined as follows (only consistent answer sets are defined).

Definition 1 A total interpretation S is an *answer set* of a program P iff S^+ is the least model[1] of the program $P^+ = \{r^+ \mid S \models body^-(r)\}$. □

Note that an answer set S is usually represented by S^+ (this convention is sometimes used also in this paper).

The set of all answer sets of a program P is denoted by $SM(P)$. A program is called *coherent* iff it has an answer set.

Strict partial order is a binary relation, which is irreflexive, transitive and, consequently, assymetric.

Definition 2 (Prioritized logic program) A *prioritized logic program* $(P, \prec, \mathcal{N})$ is a program P together with a strict partial order $\prec$ on rules of P and with a function $\mathcal{N}$ assigning names to rules of P.

If $r_1 \prec r_2$ it is said that r_2 is more preferred than r_1. □

A remark - if a symbol r is used in this paper in order to denote a rule, then r is considered as the name of that rule (no different name $\mathcal{N}(r)$ is introduced).

Definition 3 Let a program P and an answer set S be given. Let be $R = \{r \in P \mid body(r) \subseteq S\}$. It is said that R is the set of all *generating rules* of S^+. □

[1] P^+ is treated as definite logic program, i.e. each objective literal of the form $\neg A$, where $A \in At$, is considered as a new atom.

Principles

Principles suggested by (Brewka and Eiter 1999) are discussed in this section. The principles relate an order on rules with a corresponding order on answer sets. In other words: they (partially) specify what means that an order on answer sets corresponds to the given order on rules. The first two principles are from (Brewka and Eiter 1999). Postulate III reproduces an idea of Proposition 6.1 from (Brewka and Eiter 1999).

The Principles of (Brewka and Eiter 1999) are expressed in an abstract way for the general case of nonmonotonic knowledge bases (prioritized defeasible rules). We restrict the discussion (and the wording) of the Principles to the case of logic programs and answer sets.

Principle I Let a prioritized logic program $(P, \prec, \mathcal{N})$ be given. Let A_1 and A_2 be two answer sets of the program P. Let $R \subset P$ be a set of rules and $d_1, d_2 \notin R$ are rules. Let A_1^+, A_2^+ be generated by the rules $R \cup \{d_1\}$ and $R \cup \{d_2\}$, respectively. If d_1 is preferred over d_2, then A_2 is not a preferred answer set of $(P, \prec, \mathcal{N})$. □

Principle II Let A be a preferred answer set of a prioritized logic program $(P, \prec, \mathcal{N})$. and r be a rule such that $body^+(r) \not\subseteq A^+$. Then A is a preferred answer set of $(P \cup \{r\}), \prec', \mathcal{N}')$, whenever $\prec'$ agrees with $\prec$ on rules in P and $\mathcal{N}'$ extends $\mathcal{N}$ with the name r. □

Principle III Let a prioritized logic program $(P, \prec, \mathcal{N})$ be given and $\mathcal{B} \neq \emptyset$ be the set of all answer sets of P. Then there is a selection function Σ s.t. $\Sigma(\mathcal{B})$ is the set of all preferred answer sets of $(P, \prec, \mathcal{N})$, where $\emptyset \neq \Sigma(\mathcal{B}) \subseteq \mathcal{B}$. □

It is shown in (Brewka and Eiter 1999), Proposition 6.1, that Principle II is incompatible with Principle III, if the notion of preferred answer set from (Brewka and Eiter 1999) is accepted:

Example 4 ((Brewka and Eiter 1999)) Consider program P, whose single standard answer set is $S = \{b\}$ and the rule (1) is preferred over the rule (2).

$$c \leftarrow not\ b \quad (1)$$
$$b \leftarrow not\ a \quad (2)$$

S is not a preferred answer set in the framework of (Brewka and Eiter 1999)[2]; there is no BE-preferred answer set of this $(P, \prec, \mathcal{N})$ and there are also many other cases of prioritized programs with standard answer sets, but without BE-preferred, D-preferred or W-preferred answer sets.

Assume that S, the only standard answer set of P, is selected – according to the Principle III – as the preferred answer set of $(P, \prec, \mathcal{N})$. Let P' be $P \cup \{a \leftarrow c\}$ and $a \leftarrow c$ be preferred over the both rules 1 and 2. P' has two standard answer sets, S and $T = \{a, c\}$. Note that $\{c\} \not\subseteq S^+$. Hence, S should be the preferred answer set of P' according to the Principle II. However, in the framework of (Brewka and Eiter 1999) the only preferred answer set of $(P', \prec', \mathcal{N}')$ is T. This selection of preferred answer set satisfies clear intuitions – T is generated by the two most preferred rules. □

Principle III is of crucial value according to our view. If we accept a program as a representation of a domain, then we consider its answer sets as (alternative) condensed representations of the domain. If a preference relation is introduced, some most preferred condensed representations should be considered. The preference relation on rules and its impact on a preference relation on answer sets should not be totally destructive – at least one of the original condensed representations should be preferred. Therefore, we are aiming at a less restrictive correspondence between order on rules and order on answer sets.

In any case, there are good reasons to modify (or reject) Principle II. Let a program P with a set of answer sets $\mathcal{B}$ be given. If P is extended to P', we can sometimes get an extended set of answer sets $\mathcal{B}' \supset \mathcal{B}$ (in contrast to standard logical theories and their models). Hence, we select preferred answer sets of P' from a broader variety of possibilities. Consequently, no condition satisfied by some $B \in \mathcal{B}$ should constraint the selection of preferred answer set from $\mathcal{B}'$ (even from $\mathcal{B}' \setminus \mathcal{B}$). Principle II is not accepted in this paper. Principle II is not accepted also in (Sakama and Inoue 2000). According to (Delgrande et al. 2004) descriptive approaches do not satisfy this principle in general.

We propose a new principle below. The principle express our position: the use of a preference relation should be focused on the blocking of less preferred rules and not on their application. Moreover, the principle is useful in preventing a problem with Rintanen's approach, see Example 3.2 from (Brewka and Eiter 1999). We translate the default theory from the example into a logic program.

Example 5 Let P be

$$r_1 \quad b \leftarrow a, not\ \neg b$$
$$r_2 \quad \neg a \leftarrow not\ a$$
$$r_3 \quad a \leftarrow not\ \neg a$$

If $i < j$, then r_i is more preferred than r_j (in all examples of this paper). There are two standard answer sets of P: $S_1 = \{\neg a\}, S_2 = \{a, b\}$. S_2 corresponds to the preferred extension of the corresponding default theory according to Rintanen. This is rejected in (Brewka and Eiter 1999) because of Principle II.

We see another possibility how to prevent the selection of S_2. See Principle IV below (one of the generating rules of S_2 is blocked by a more preferred rule). □

Let P be a program and $r_1, r_2 \in P$. It is said, that r_1 *blocks* r_2 iff $not\ head(r_1) \in body^-(r_2)$.

Principle IV.

[2] We will use notation from (Delgrande, Schaub, and Tompits 2003). BE-preferred means preferred according to (Brewka and Eiter 1999), D-preferred according to (Delgrande, Schaub, and Tompits 2003) and W-preferred according to (Wang, Zhou, and Lin 2000). Definitions of BE-, D- and W- preferred answer sets are missing in this paper because of the limited space. We hope that the main line of thoughts of this paper does not suffer from that. A precise formal comparison of approaches mentioned above can be found in (Schaub and Wang 2001).

Let A_1, A_2 be answer sets of a prioritized program $(P, \prec, \mathcal{N})$, $r_2 \prec r_1$ and r_1 blocks r_2.

Let a set of generating rules of A_1 contains r_1 and let r_2 be a member of each set of generating rules of A_2.

Then A_2 is not a preferred answer set. □

As regards a choice of principles, we accept the position of (Brewka and Eiter 1999): even if somebody does not accept a set of principles for preferential reasoning, those (and similar) principles are still of interest as they may be used for classifying different patterns of reasoning.

In order to conclude this section: we accept Principles I, III and IV.

From programs to arguments

We intend to apply the preference relation on rules only in order to block the less preferred rules whenever assumptions (the sets of default literals) in their bodies are contradicted by consequences of more preferred rules. Therefore, our attention is focused on assumptions, consequences of assumptions, negative programs and negative equivalents of programs.

We propose to consider assumptions as arguments and to do a transition from the preference on rules to a preference on arguments. In this section are recapped some notions useful for that goal (from (Dimopoulos and Torres 1996) and (Sefranek 2006)).

Definition 6 ($\ll_P$) An objective literal L *depends* on an assumption (on a set of default literals) W *with respect to a program* P ($L \ll_P W$) iff there is a sequence of rules $\langle r_1, \ldots, r_k \rangle$, $k \geq 1$, $r_i \in P$ such that

- $head(r_k) = L$,
- $W \models body(r_1)$,
- for each i, $1 \leq i < k$, $W \cup \{head(r_1), \ldots, head(r_i)\} \models body(r_{i+1})$.

The set $\{L \in Lit \mid L \ll_P Xs\} \cup Xs$ is denoted by $Cn_{\ll_P}(Xs)$.

The assumption Xs is *self-consistent* w.r.t. a program P iff $Cn_{\ll_P}(Xs)$ is consistent. □

If $Z \subseteq Obj$, we will use sometimes the notation $Cn_{\ll_{P \cup Z}}(Xs)$, assuming that the program P is extended by the set of facts Z.

Dependencies on assumptions are expressed in (Dimopoulos and Torres 1996) in terms of support. Some important results on negative programs, used in our argumentation framework, are based on the notion of support.

Definition 7 An assumption Xs is a *support* for an objective literal L (for a set S of objective literals) in a program P iff $L \in Cn_{\ll_P}(Xs)$ ($S \subset Cn_{\ll_P}(Xs)$).

Let Xs be a support for L (for S) in P. Then Xs is a *minimal support* for L (for S) in P iff for no $Y \subseteq Xs$ holds $L \in Cn_{\ll_P}(Y)$ ($S \subset Cn_{\ll_P}(Y)$). □

Definition 8 Let P be a program and Xs be a self-consistent assumption.

An interpretation I is a *supported interpretation* of Xs iff $I = Cn_{\ll_P}(Xs)$.

An interpretation is *supported* iff it is the supported interpretation of some self-consistent assumption Xs. □

Answer sets can be equivalently characterized in terms of supported interpretations and in terms of dependencies.

Proposition 9 *An interpretation S is an answer set of a program P iff S is total and $S = Cn_{\ll_P}(S^-)$.*

A supported interpretation I is an answer set of P iff it is total. □

Definition 10 Two logic programs, P_1 and P_2, are *support-equivalent* iff for every assumption Xs and for every objective literal L holds $L \in Cn_{\ll_{P_1}}(Xs)$ iff $L \in Cn_{\ll_{P_2}}(Xs)$. □

Definition 11 If for each rule $r \in P$ holds that $body(r) = body^-(r)$, then P is a *negative* logic program. A negative program is *reduced* iff $r_1 = r_2$ whenever $body(r_1) \subseteq body(r_2)$. □

Our notion of preferred answer set is based on argumentation structures which correspond to negative equivalents of programs. The following two propositions provide a theoretical background for that use of argumentation structures.

Definition 12 The *negative equivalent* of a given logic program P is the negative logic program R containing exactly every rule r, where $body(r)$ is a minimal support (in P) of some objective literal L and $L = head(r)$. □

Proposition 13 *Let P be a program. Then its negative equivalent R is reduced and support-equivalent to P and, consequently, $SM(P) = SM(R)$.*

It is intended to apply the preference relation on rules for blocking the less preferred rules with assumptions contradicted by the consequence of a more preferred rule. Therefore, a way how to transfer the preference relation to the negative equivalent of the given program could be interesting. However, a reasonable and straightforward transfer is impossible – the rules of the negative equivalent do not correspond to the original rules in a unique way. An attempt to construct the transfer by means of argumentation structures is presented in the following sections.

Argumentation structures

A descriptive approach to preferred answer sets specification is presented in this paper. Remind that according to (Delgrande and Schaub 2000) a descriptive approach can be characterized by such order of applied rules which reflects the "desirability" of rules application.

We attempt to specify "desirability" in terms of arguments. Briefly, a preferred answer set is supported by a complete argument, which is not blocked by a more preferred argument. An order of rules can be reconstructed from the argument.

Our aim is to transfer a preference relation defined on rules to a notion of preferred answer sets via a notion of argument (the method is inspired by (García and Simari 2004)). While there defeasible rules are treated as (defeasible) arguments, here (defeasible) assumptions, sets of default negations, are considered as arguments. Reasoning about a logic program is here understood as a kind of argumentation, rules of the program are considered as unquestionable truths and sets of default literals can be viewed as defeasible arguments, which may be contradicted by consequences of some applicable rules.

The preference relation on rules is used in order to ignore the attacks of less preferred arguments against more preferred arguments. The core problem is to transfer the preference relation defined on rules to arguments.

Let us begin by an example illustrating how could a notion of argument be used in the context of logic programs.

Example 14 ((Brewka and Eiter 1999))

$$r_1 \qquad b \leftarrow a, not\ \neg b$$
$$r_2 \qquad \neg b \leftarrow not\ b$$

Consider the rule r_2. The literal $\neg b$ is supported by the argument $not\ b$. If the argument is true (can be consistently evaluated as true with respect to a program containing r_2) then also $\neg b$ can be evaluated as true. As regards the rule r_1, default negation $not\ \neg b$ can be treated as an argument for b, if a is true. We are going to introduce a notion of argumentation structure, which encodes also such "conditional arguments".

Of course, some arguments can be treated as counterarguments against other arguments. If rules r_1 and r_2 belong to a program P and we accept the argument $not\ b$ (with the consequence $\neg b$), it can be treated as a counterargument to $not\ \neg b$ and vice versa. □

Definition 15 (Argument) Let P be a program.

A self-consistent set of default negations X is called an *argument* w.r.t. the program P for a consistent set of objective literals Y, given a set of objective literals Z iff

1. $pos(X) \cap Z = \emptyset$,
2. $Y \subseteq Cn_{\ll_{P \cup Z}}(X)$.

We will use the notation $\langle Y \hookleftarrow X; Z\rangle$ and the triple denoted by it is called an *argumentation structure* (w.r.t. P). □

A remark: we do not require a minimality of arguments (this is not an appropriate feature for answer sets specification).

If $Z = \emptyset$ also a shortened notation $\langle Y \hookleftarrow X\rangle$ can be used. We will omit sometimes the phrase "w.r.t. P" and speak simply about arguments (the corresponding program is always clear from the context).

We emphasize that only *self-consistent* arguments for *consistent* sets of objective literals are considered in this paper.

Derivation of argumentation structures

Some argumentation structures can be derived from another argumentation structures. The derivation is grounded on the basic argumentation structures, which correspond to the rules of the given program.

If $r \in P$ is a rule, then $\langle head(r) \hookleftarrow body^-(r); body^+(r)\rangle$ is a *basic* argumentation structure (w.r.t. P).

Only the argumentation structures derived from the basic argumentation structures using derivation rules from Definition 17 are of interest in the rest of this paper. Whenever we use the term "argumentation structure" below, we mean "argumentation structure derived from basic argumentation structures using derivation rules".

Example 16 Consider a program P – the rule r_3 is added to rules r_2 and r_1 from Example 14 (P is used as a running example in next paragraphs).

$$r_3 \qquad a \leftarrow not\ \neg a$$

The following (basic) argumentation structures correspond to rules of P: $\langle\{b\} \hookleftarrow \{not\ \neg b\}; \{a\}\rangle, \langle\{\neg b\} \hookleftarrow \{not\ b\}\rangle, \langle\{a\} \hookleftarrow \{not\ \neg a\}\rangle$ (let denote them by $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, respectively).

As examples of two kinds of derived argumentation structures can serve: The argumentation structure $\mathcal{A}_4 = \langle\{b\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle$ can be derived (by "unfolding") from $\mathcal{A}_1$ and $\mathcal{A}_3$ and $\mathcal{A}_5 = \langle\{b, a\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle$ from $\mathcal{A}_3$ and $\mathcal{A}_4$ (by joining arguments and sets of literals supported by the arguments).

Unfolding is used in order to obtain sets of argumentation structures, which are equivalents of negative programs. Joining is applied to those equivalents of negative programs in order to compose argumentation structures corresponding to answer sets. □

Derivation rules R1 and R2 are motivated in Example 16. The role of R3 is an extension of minimal arguments (assumptions). This feature is necessary for a generation of argumentation structures corresponding to answer sets.

Definition 17 (Derivation rules) Let P be a program.

R1 Let be $r_1, r_2 \in P$, $\mathcal{A}_1 = \langle\{head(r_1)\} \hookleftarrow X_1; Z_1\rangle$ and $\mathcal{A}_2 = \langle\{head(r_2)\} \hookleftarrow body^-(r_2); body^+(r_2)\rangle$ argumentation structures, $head(r_2) \in Z_1$ and $X_1 \cup body^-(r_2) \cup Z_1 \cup body^+(r_2) \cup \{head(r_1)\}$ be consistent.
Then also $\mathcal{A}_3 = \langle head(r_1) \hookleftarrow X_1 \cup body^-(r_2); (Z_1 \setminus \{head(r_2)\}) \cup body^+(r_2)\rangle$ is an argumentation structure.

R2 Let $\mathcal{A}_1 = \langle Y_1 \hookleftarrow X_1\rangle$ and $\mathcal{A}_2 = \langle Y_2 \hookleftarrow X_2\rangle$ be argumentation structures and $X_1 \cup X_2 \cup Y_1 \cup Y_2$ be consistent. Then also $\mathcal{A}_3 = \langle Y_1 \cup Y_2 \hookleftarrow X_1 \cup X_2\rangle$ is an argumentation structure.

R3 Let $\mathcal{A}_1 = \langle Y_1 \hookleftarrow X_1\rangle$ be an argumentation structure and W be an assumption.
If $X_1 \cup W \cup Y_1$ is consistent, then also $\mathcal{A}_2 = \langle Y_1 \hookleftarrow X_1 \cup W\rangle$ is an argumentation structure.

A *derivation* of $\mathcal{A}$ (w.r.t. P) is a sequence $\langle \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k \rangle$ of argumentation structures (w.r.t. P) such that $\mathcal{A}_1$ is a basic argumentation structure, $\mathcal{A} = \mathcal{A}_k$ and each $\mathcal{A}_i$, $1 < i \leq k$, is either a basic argumentation structure or it is obtained by R1 or R2 or R3 from preceding argumentation structures.

Argumentation structures of the form $\langle X \hookleftarrow Y; \emptyset \rangle$ are of the fundamental importance - they correspond to the negative rules. It is known from (Dimopoulos and Torres 1996) that each answer set of a program P can be represented as an answer set of the support equivalent negative reduced program, see Proposition 13. We are aiming at an identification of argumentation structures that generate answer sets and are not blocked in the sense defined below.

The following proposition shows that the rule R1 enables to derive argumentation structures, which are equivalents of the support equivalent negative reduced program.

Proposition 18 *Let P be a program, let be $L \in Obj$ and $W \subseteq Def$. If $L \in Cn_{\ll_P}(W)$ and W is a minimal support for L, then either $\langle \{L\} \hookleftarrow W \rangle$ is a basic argumentation structure or there is a derivation of $\langle \{L\} \hookleftarrow W \rangle$ and only the rule R1 is used in the derivation.*

Proof Sketch: Suppose that $\langle \{L\} \hookleftarrow W \rangle$ is not a basic argumentation structure.

Consider a sequence of rules $\langle r_1, \ldots, r_k \rangle$, where $k > 1$ and a rule $r_i, i > 1$. The argumentation structure corresponding to r_i is $\langle head(r_i) \hookleftarrow body^-(r_i); body^+(r_i) \rangle$. It holds that $body^-(r_i) \subseteq W$ and $body^+(r_i) \subseteq \{head(r_1), \ldots, head(r_{i-1})\} = Z^{i-1}$. Hence, after at most $i-1$ applications of R1 we obtain $\langle head(r_i) \hookleftarrow W \rangle$. Notice that $L = head(r_k)$ □

Therefore, to each rule from the negative equivalent of the given program P exists a corresponding argumentation structure (w.r.t. P).

On the other hand, a similar correspondence is from derived argumentation structures to the rules of the negative equivalent of P.

Proposition 19 *If there is a derivation of an argumentation structure $\mathcal{A} = \langle \{L\} \hookleftarrow Xs \rangle$ (w.r.t. P) using only R1, then Xs is a minimal support of L and there is $r \in R$ s.t. $head(r) = L$ and $body(r) = body^-(r) = Xs$.*

Proof Sketch: $L \in Cn_{\ll_P}(Xs)$, i.e. L is supported by Xs. If only R1 is used, then the support is minimal.

Consequence 20 *Let R be the negative equivalent of P. For each $r \in R$ there is a derivation (w.r.t. P) of the corresponding argumentation structure $\langle \{head(r)\} \hookleftarrow body(r) \rangle$.*

Argumentation structures of the form $\langle Y \hookleftarrow X; Z \rangle$, $Z \neq \emptyset$ may be derived only using R1 and they support only singletons, heads of some rules. Sets of literals, supported by an argument are constructed only from argumentation structures of the form $\langle \{L\} \hookleftarrow X \rangle$.

Consequence 21 *If there is a derivation of an argumentation structure $\langle Y \hookleftarrow X; Z \rangle$, $Z \neq \emptyset$, then $Y = \{L\}$ for some $L \in Obj$ and $L \in\subseteq Cn_{\ll_{P \cup Z}}(X)$.*

Prof Sketch: The rules R2, R3 can be applied only if $Z = \emptyset$. If only R1 is used in a derivation of an argumentation structure $\mathcal{A}_1 = \langle Y_1 \hookleftarrow X_1; Z_1 \rangle$, then Y_1 is a singleton.

Proposition 22 *If there is a derivation of an argumentation structure $\langle Y \hookleftarrow X \rangle$, then $Y \subseteq Cn_{\ll_P}(X)$.*

Proposition 23 *If S is an answer set of a program P, then there is an argumentation structure $\langle S^+ \hookleftarrow S^- \rangle$ derivable from the basic argumentation structures.*

Proof Sketch: $S = Cn_{\ll_P}(S^-)$ (see Proposition 9). According to Proposition 18 for each $L \in S^+$ there is a derivation of $\langle \{L\} \hookleftarrow W \rangle$, where $W \subseteq S^-$. The argumentation structure $\langle S^+ \hookleftarrow S^- \rangle$ is obtained using R2 and R3.

Attacks

Our approach to preferred answer sets is based on a solution of conflicts between argumentation structures. We distinguish three steps towards that goal. *Contradictions* between argumentation structures represent the elementary step. Only some contradictions are called attacks. The basic attacks are defined only for the basic argumentation structures. Consider two basic argumentation structures $\mathcal{A}_1$ and $\mathcal{A}_2$. If $\mathcal{A}_1$ contradicts $\mathcal{A}_2$ and corresponds to a more preferred rule, then it *attacks* $\mathcal{A}_2$. Attacks are propagated to other argumentation structures via some derivation rules. Finally, in the case of attacks between complete argumentation structures (corresponding to answer sets) we speak about *blocking*.

Definition 24 Consider argumentation structures $\mathcal{A} = \langle Y_1 \hookleftarrow X_1; Z_1 \rangle$ and $\mathcal{B} = \langle Y_2 \hookleftarrow X_2; Z_2 \rangle$.

If there is a literal $L \in Y_1$ such that $not\ L \in X_2$, it is said that the argument X_1 *contradicts* the argument X_2 and the argumentation structure $\mathcal{A}$ *contradicts* the argumentation structure $\mathcal{B}$.

It is said that X_1 is a *counterargument* to X_2. □

Two trivial observations (important for a generation of preferred answer sets):

The basic argumentation structures corresponding to the facts of the given program are not contradicted.

Let $r_1 = a \leftarrow$ be a fact, let r_2 be more preferred than r_1 and $not\ a \in body^-(r_2)$. Then any assumption that contains $body^-(r_2)$ is not self-consistent and, therefore, it is not an argument.

Example 25 In Example 16 $\mathcal{A}_1$ contradicts $\mathcal{A}_2$ and $\mathcal{A}_2$ contradicts $\mathcal{A}_1$. □

Example 25 shows that only some counterarguments are interesting: the rule r_1 is more preferred than the rule r_2, therefore the counterargument of $\mathcal{A}_2$ against $\mathcal{A}_1$ should not be "effectual". We are going to introduce a notion of *attack* in order to denote "effectual" counterarguments.

Similarly as for the case of argumentation structures, the basic attacks are defined first. A terminological convention: if $\mathcal{A}_1$ attacks $\mathcal{A}_2$, it is said that the pair $(\mathcal{A}_1, \mathcal{A}_2)$ is an attack.

Definition 26 Let be $r_2 \prec r_1$ and $\mathcal{A}_1 = \langle \{head(r_1)\} \hookleftarrow body^-(r_1); body^+(r_1)\rangle$ contradicts $\mathcal{A}_2 = \langle \{head(r_2)\} \hookleftarrow body^-(r_2); body^+(r_2)\rangle$.

Then $\mathcal{A}_1$ *attacks* $\mathcal{A}_2$ and it is said, that this attack is *basic*.

Attacks of argumentation structures "inherited" (propagated) from basic attacks are defined in terms of some derivation rules. The rules of that inheritance are motivated and defined below.

Example 27 Let us continue with Example 16.

Consider the basic argumentation structures $\mathcal{A}_1 = \langle \{b\} \hookleftarrow \{not\ \neg b\}; \{a\}\rangle$, $\mathcal{A}_2 = \langle \{\neg b\} \hookleftarrow \{not\ b\}\rangle$, $\mathcal{A}_3 = \langle \{a\} \hookleftarrow \{not\ \neg a\}\rangle$ and the derived argumentation structures $\mathcal{A}_4 = \langle \{b\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle$, $\mathcal{A}_5 = \langle \{b, a\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle$.

$(\mathcal{A}_1, \mathcal{A}_2)$ is the only basic attack.

A derivation of the attacks of $(\mathcal{A}_4, \mathcal{A}_2)$ and $(\mathcal{A}_5, \mathcal{A}_2)$ could be motivated as follows. $\mathcal{A}_4$ is derived from $\mathcal{A}_1$ and $\mathcal{A}_3$ using R1, the attack of $\mathcal{A}_1$ against $\mathcal{A}_2$ should be propagated to the attack $(\mathcal{A}_4, \mathcal{A}_2)$. Note that $\mathcal{A}_3$ is a neutral argumentation structure with respect to attacks.

Now, $\mathcal{A}_5$ is derived from $\mathcal{A}_3$ and $\mathcal{A}_4$. Again, the attack of $\mathcal{A}_4$ against $\mathcal{A}_2$ should be inherited by $(\mathcal{A}_4, \mathcal{A}_5)$.

To the contrary, $\mathcal{A}_2$ contradicts $\mathcal{A}_4$ and $\mathcal{A}_5$, but it is based on a less preferred rule, hence those contradictions are not considered as attacks. □

First we define two rules, which specifies inheritance of attacks "via unfolding". Second, if we have two attacks and the attacking sides are joined and also the attacked sides are joined (and some natural conditions are satisfied), then the former argumentation structure attacks the later. Finally, a class of similar cases: the "attacking side" is preserved, when both attacking and attacked argumentation structures are joined with a "neutral member".

A question, whether those derivation rules for attacks are sufficient and necessary arises in a natural way. Our only response to the question in this paper is that Principles I, III and IV are satisfied, when we use this notion of attack.

A technical remark: Attack derivation rules are designed in order to derive attacks from another attacks; however, if a new argumentation structure is used in the consequent of a rule, it is necessary to check, whether the argument of that structure is self-consistent (and, consequently, whether it is really an argumentation structure). Notice that G_P, defined below in Definition 29 contains all relevant argumentation structures as vertices.

Derivation rules, which propagate attacks are defined below. We introduce some conventions and shortcuts in order to simplify the presentation. We represent more (very similar) rules by some schemes of rules.

1. $u(\mathcal{A}_1, \mathcal{A}_2)$ denotes the result of "unfolding" of argumentation structures $\mathcal{A}_1$ and $\mathcal{A}_2$ (f.ex. the result of unfolding of $\mathcal{A}_1 = \langle \{head(r_1)\} \hookleftarrow X_1; Z_1\rangle$ by $\mathcal{A}_2 = \langle \{head(r_2)\} \hookleftarrow body^-(r_2); body^+(r_2)\rangle$ is $u(\mathcal{A}_1, \mathcal{A}_2) = \langle \{head(r_1)\} \hookleftarrow X_1 \cup body^-(r_2); (Z_1 \setminus \{head(r_2)\}) \cup body^+(r_2)\rangle$). It is assumed, that consequences of argumentation structures, involved in unfolding are singletons, heads of rules.
2. $\mathcal{A}_1 \cup \mathcal{A}_2$ means $\langle Y_1 \cup Y_2 \hookleftarrow X_1 \cup X_2\rangle$ for $\mathcal{A}_i = \langle Y_i \hookleftarrow X_i\rangle, i = 1, 2$; in this context we consider also "zero"-"argumentation structures" $\langle \emptyset \hookleftarrow \emptyset\rangle$, $\langle \emptyset \hookleftarrow W\rangle$ (only as a notational convention for this definition). Observe that $\langle Y \hookleftarrow \emptyset\rangle$ could be a regular argumentation structure. An occurrence of a zero-argumentation-structure should be clear in a given context.

   For example - $\mathcal{A}_1 \cup \mathcal{A}_2$ may represent also $\langle Y_1 \hookleftarrow X_1 \cup X_2\rangle$ or $\langle Y_1 \cup Y_2 \hookleftarrow X_1\rangle$ and $\mathcal{A}_1 \cup \mathcal{A}_2$ attacks $\mathcal{A}_3 \cup \mathcal{A}_4$ may represent also $\mathcal{A}_1$ attacks $\langle Y_3 \hookleftarrow X_3 \cup X_4\rangle$.

Definition 28 (Attack derivation rules) Basic attacks are attacks.

Q1 Let $\mathcal{A}_1$ attacks $\mathcal{A}_2 = \langle \{head(r_2)\} \hookleftarrow X_2; Z_2\rangle$, $\mathcal{A}_3 = \langle \{head(r_3)\} \hookleftarrow X_3; Z_3\rangle$ be an argumentation structure, which does not attack $\mathcal{A}_1$, $head(r_3) \in Z_2$.
If $u(\mathcal{A}_2, \mathcal{A}_3)$ is an argumentation structure[3], then $\mathcal{A}_1$ attacks $u(\mathcal{A}_2, \mathcal{A}_3)$.

Q2 Let $\mathcal{A}_1 = \langle \{head(r_1)\} \hookleftarrow X_1; Z_1\rangle$ attacks $\mathcal{A}_2$. Let $\mathcal{A}_3 = \langle \{head(r_3)\} \hookleftarrow X_3; Z_3\rangle$ be not attacked and $head(r_3) \in Z_1$.
If $u(\mathcal{A}_1, \mathcal{A}_3)$ is an argumentation structure, then $u(\mathcal{A}_1, \mathcal{A}_3)$ attacks $\mathcal{A}_2$.

Q3 Suppose that $\mathcal{A}_1 = \langle Y_1 \hookleftarrow X_1\rangle$ attacks $\mathcal{A}_2 = \langle Y_2 \hookleftarrow X_2\rangle$ and $\mathcal{A}_3 = \langle Y_3 \hookleftarrow X_3\rangle$ attacks $\mathcal{A}_4 = \langle Y_4 \hookleftarrow X_4\rangle$; neither $\mathcal{A}_3$, nor $\mathcal{A}_4$ attacks $\mathcal{A}_1$.
Then $\mathcal{A}_1 \cup \mathcal{A}_3$ attacks $\mathcal{A}_2 \cup \mathcal{A}_4$, if both are argumentation structures.

Q4 Suppose that $\mathcal{A}_1 = \langle Y_1 \hookleftarrow X_1\rangle$ attacks $\mathcal{A}_2 = \langle Y_2 \hookleftarrow X_2\rangle$. Let $\mathcal{B}_i$, $i = 1, 2$ be (possibly zero-) argumentation structures of the form $\langle U_i \hookleftarrow W_i\rangle$, both does not attack $\mathcal{A}_1$ and $\mathcal{B}_2$ does not attack $\mathcal{B}1$.
Then $\mathcal{A}_1 \cup \mathcal{B}_1$ attacks $\mathcal{A}_2 \cup \mathcal{B}_2$.

There are no other attacks except those specified above.

A *derivation of an attack* is a sequence $\mathcal{X}_1, \ldots, \mathcal{X}_k$, where each $\mathcal{X}_i$ is an attack (a pair of attacking and attacked argumentation structures), $\mathcal{X}_1$ is a basic attack and each $\mathcal{X}i$, $1 < i \leq k$ is either a basic attack or it is derived from the previous attacks using rules Q1, Q2, Q3, Q4.

We define below in Definition 29 a graph with argumentation structures as vertices and attacks as edges.

Definition 29 Let $(P, \prec, \mathcal{N})$ be a prioritized logic program. We define an oriented graph $G_P = (V, E)$, where vertices V are argumentation structures derived from the basic argumentation structures and edges E are attacks derived from the basic attacks. □

[3]The role of this condition – in all items of this definition – is to ensure the consistency of arguments and their consequences.

Preferred answer sets

Definition 30 (Complete arguments, blocked arguments) An argumentation structure $\langle Y \hookleftarrow X \rangle$ is called *complete* iff for each literal $L \in Obj$ it holds that $L \in Y$ or $not\ L \in X$.

A complete argumentation structure $\mathcal{A}_1$ is *blocked* iff there is a complete argumentation structure $\mathcal{A}_2$, which attacks $\mathcal{A}_1$ and $\mathcal{A}_2$ itself is not attacked by a complete argumentation structure. □

Example 31 Consider our running example. Remind all the relevant information: P is

$$\begin{array}{lrcl} r_1 & b & \leftarrow & a, not\ \neg b \\ r_2 & \neg b & \leftarrow & not\ b \\ r_3 & a & \leftarrow & not\ \neg a \end{array}$$

$\mathcal{A}_1 = \langle \{b\} \hookleftarrow \{not\ \neg b; \{a\}\}\rangle, \mathcal{A}_2 = \langle \{\neg b\} \hookleftarrow \{not\ b\}\rangle, \mathcal{A}_3 = \langle \{a\} \hookleftarrow \{not\ \neg a\}\rangle, \mathcal{A}_4 = \langle \{b\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle, \mathcal{A}_5 = \langle \{b, a\} \hookleftarrow \{not\ \neg b, not\ \neg a\}\rangle, \mathcal{A}_6 = \langle \{\neg b, a\} \hookleftarrow \{not\ \neg a, not\ b\}\rangle$. $\{(\mathcal{A}_1, \mathcal{A}_2), (\mathcal{A}_4, \mathcal{A}_2), (\mathcal{A}_5, \mathcal{A}_2), (\mathcal{A}_4, \mathcal{A}_6)(\mathcal{A}_5, \mathcal{A}_6)\} \subseteq E$. No pair $(\mathcal{U}, \mathcal{A}_5)$ such that $\mathcal{U}$ is derived from $\mathcal{A}_2$ is in E.

Hence: the complete argumentation structure $\mathcal{A}_6$ is blocked, $\mathcal{A}_5$ is complete and not blocked.

Observe that $\mathcal{A}_5$ contains an argument for $\{a, b\}$, an answer set of P, similarly $\mathcal{A}_6$ contains an argument for $\{a, \neg b\}$, which is another answer set of P.

We will prefer $\mathcal{A}_5$ over $\mathcal{A}_6$ (the later is blocked by the former). Consequently, we will consider $\{a, b\}$ as a preferred answer set of the given prioritized logic program. □

Definition 32 (Preferred answer set) An argumentation structure is *preferred* iff it is complete and not blocked.

$Y \cup X$ is a *preferred answer set* iff $\langle Y \hookleftarrow X \rangle$ is a preferred argumentation structure. □

The following example shows that the argumentation structure corresponding to the only answer set of a program is preferred, even if it is attacked (by an argumentation structure which is not complete).

Example 33

$$\begin{array}{lrcl} r_1 & b & \leftarrow & not\ a \\ r_2 & a & \leftarrow & not\ b \\ r_3 & c & \leftarrow & a \\ r_4 & c & \leftarrow & not\ c \end{array}$$

Let the basic argumentation structures are denoted by $\mathcal{A}_i$, $i = 1, \ldots, 4$. $(\mathcal{A}_1, \mathcal{A}_2), (\mathcal{A}_3, \mathcal{A}_4)$ are the basic attacks. $\mathcal{A}_1$ attacks $\mathcal{A}_5 = \langle \{c\} \hookleftarrow \{not\ b\}\rangle$ according to the rule Q1 and $\mathcal{A}_1$ attacks $\mathcal{A}_6 = \langle \{c, a\} \hookleftarrow \{not\ b\}\rangle$ according to the rule Q4. However, the complete argumentation structure $\mathcal{A}_6$ is not attacked by another complete argumentation structure (there is no such structure) and, consequently it is the preferred argumentation structure. □

Theorem 34 *If S is a preferred answer set of $(P, \prec, \mathcal{N})$, then S is an answer set of P.*

Proof Sketch: We assume that $\mathcal{A} = \langle S^+ \hookleftarrow S^- \rangle$ is a preferred argumentation structure. Then $\mathcal{A}$ is complete. Obviously, $S = Cn_{\ll_P}(S^-)$. □

Example 35 ((Brewka and Eiter 1999)) Consider the prioritized logic program, where r_i are names of the rules (occurring in the corresponding row) and the rule r_i is more preferred than the rule r_j whenever $i < j$.

$$\begin{array}{lrcl} r_1 & p & \leftarrow & not\ q \\ r_2 & q & \leftarrow & not\ \neg q \\ r_3 & \neg p & \leftarrow & not\ p \\ r_4 & p & \leftarrow & not\ \neg p \end{array}$$

$\mathcal{A}_3 = \langle \{\neg p\} \hookleftarrow \{not\ p\}\rangle$ is attacked by $\mathcal{A}_1 = \langle \{p\} \hookleftarrow \{not\ q\}\rangle$ and $\mathcal{A}_4 = \langle \{p\} \hookleftarrow \{not\ \neg p\}\rangle$ is attacked by $\mathcal{A}_3 = \langle \{\neg p\} \hookleftarrow \{not\ p\}\rangle$.

There are two complete argumentation structures in P: $\mathcal{A}_5 = \langle \{q, \neg p\} \hookleftarrow \{not\ \neg q, not\ p\}\rangle$ and $\mathcal{A}_6 = \langle \{q, p\} \hookleftarrow \{not\ \neg p, not\ \neg q\}\rangle$.

Notice that only $\mathcal{A}_6$ is blocked. $\mathcal{A}_5$ attacks $\mathcal{A}_6$ and $\mathcal{A}_1$ attacks $\mathcal{A}_5$, but $\mathcal{A}_1$ is not complete and there is no complete argumentation structure which attacks $\mathcal{A}_5$. □

Reconsider Example 4 of a prioritized program without BE-preferred answer set.

Example 36 ((Brewka and Eiter 1999))

$$\begin{array}{lrcl} r_1 & c & \leftarrow & not\ b \\ r_2 & b & \leftarrow & not\ a \end{array}$$

Remind that $r_2 \prec r_1$. There is no edge in G_P (there is no basic attack, hence no attack can be inherited). The only complete vertex is $\langle \{b\} \hookleftarrow \{not\ a, not\ c\}\rangle$. Hence, $\{b\}$ is a preferred answer set of $(P, \prec, \mathcal{N})$.

If we add the most preferred rule $r_0 = a \leftarrow c$ to P, then $\mathcal{A}_0 = \langle \{a\} \hookleftarrow \emptyset; \{c\}\rangle$ attacks $\mathcal{A}_2 = \langle b \hookleftarrow not\ a\rangle$ and it can be derived that the complete argumentation structure $\mathcal{A}_4 = \langle \{a, c\} \hookleftarrow \{not\ b\}\rangle$ attacks the complete argumentation structure $\mathcal{A}_5 = \langle \{b\} \hookleftarrow \{not\ a, not\ c\}\rangle$. Moreover, $\mathcal{A}_4$ is not attacked and not blocked. Hence, $\mathcal{A}_4$ is preferred. Therefore. $\{a, c\}$ is the preferred answer set. □

Principle III is satisfied:

Theorem 37 *Let $\mathcal{P} = (P, \prec, \mathcal{N})$ be a prioritized logic program and $SM(P) \neq \emptyset$.*
Then there is a preferred answer set of $\mathcal{P}$.

Proof Sketch: Assume that for each $S \in SM(P)$ holds that the complete argumentation structure $\langle S^+ \hookleftarrow S^- \rangle$ is blocked. Consider $S_i \in SM(P)$. If $\mathcal{A}_i = \langle S_i^+ \hookleftarrow S_i^- \rangle$ is blocked, there is a complete argumentation structure $\mathcal{A}_j = \langle S_j^+ \hookleftarrow S_j^- \rangle$ which attacks $\mathcal{A}_i$ and itself is not attacked. A contradiction. □

Theorem 38 *Principle I is satisfied*

Proof Sketch: Let $A_1 \neq A_2$ be answer sets of a program P. Let $R \subset P$ be a set of rules and $d_1, d_2 \notin R$ are rules. Let A_1^+, A_2^+ be generated by the rules $R \cup \{d_1\}$ and $R \cup \{d_2\}$, respectively.

It holds that $head(d1) \in A_1$ and $head(d_2) \in A_2$, $A_1 \not\models body(d_2)$ and $A_2 \not\models body(d_1)$ (otherwise $A_1 = A_2$ or either A_1 or A_2 is not an answer set). Suppose that $d_2 \prec d_1$. It means, $\langle head(d_1) \hookleftarrow body^-(d_1); body^+(d_1)\rangle$ attacks $\langle head(d_2) \hookleftarrow body^-(d_2); body^+(d_2)\rangle$

If both $body^+(d_1)$ and $body^+(d_2)$ are empty sets, $\langle A_1^+ \hookleftarrow A_1^-\rangle$ attacks $\langle A_2^+ \hookleftarrow A_2^-\rangle$ according to Q4.

Otherwise, at least one argumentation structure from $\mathcal{B}_1 = \langle head(d_1) \hookleftarrow W_1\rangle$ and $\mathcal{B}_2 = \langle head(d_2) \hookleftarrow W_2\rangle$ is obtained using the derivation rule R1 and $\mathcal{B}_1$ attacks $\mathcal{B}_2$. Hence, also $\langle A_1^+ \hookleftarrow A_1^-\rangle$ attacks $\langle A_2^+ \hookleftarrow A_2^-\rangle$.

Therefore, A_2 is not a preferred answer set. □

Theorem 39 *Principle IV is satisfied.*

Proof Sketch: Let S_1 and S_2 be answer sets of a program P. Suppose that r_1 is a member of a generating set of S_1 and r_2 is in each set of generating rules of S_2. It is also assumed that $\mathcal{A}_2 = \langle head(r_2) \hookleftarrow body^-(r_2); body^+(r_2)\rangle$ is attacked by $\mathcal{A}_1 = \langle \{head(r_1)\} \hookleftarrow body^-(r_1); body^+(r_1)\rangle$.

Therefore, $\langle S_2^+ \hookleftarrow S_2^-\rangle$ is blocked by $\langle S_1^+ \hookleftarrow S_1^-\rangle$ and S_2 is not a preferred answer set. □

Conclusions

Results An argumentation framework has been constructed, which enables to transfer attacks of rules to attacks of argumentation structures and, consequently, to attacks of answer sets. Preferred answer sets are not attacked by another answer sets.

This construction enables a selection of a preferred answer set whenever there is a non-empty set of standard answer sets of a program.

We did not accept the second principle from (Brewka and Eiter 1999), on the other hand a new principle, which reflects the role of blocking, has been proposed. According to prescriptive approaches to prioritized logic programs, no rule can be fired before a more preferred rule, unless the more preferred rule is blocked. Programs with standard answer sets and without preferred answer sets is a consequence of that attitude. We stress the role of blocking – in our approach, rules can be blocked by more preferred rules, but the rules which are not blocked are handled in a declarative style.

Preferred answer set is not blocked by another answer set. It means, that a *desirable set of rules*, in our approach, which generates a preferred answer set, is such a set of rules, that no rule of the set can be attacked by a rule from a set of rules generating another answer set (unless that answer set is blocked).

Open problems, future research Of course, a kind of fine-tuning of our approach is intended.

Further, a more detailed comparison of our approach with other approaches to prioritized logic programs is among our plans. Also approaches not referenced in this paper are of interest (f.ex. works by Zhang and his colleagues). A comparison to defeasible logic programming of (García and Simari 2004), to other defeasible logics and argumentation frameworks is challenging for us, similarly as a comparison to dynamic logic programming.

It is assumed that from the computational complexity point of view our approach generate problems complete on the second level of polynomial hierarchy. The technical result is among the plans for the future research.

References

Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence* 109(1-2):297–356.

Delgrande, J. P., and Schaub, T. 2000. Expressing preferences in default logic. *Artificial Intelligence* 123(1-2):41–87.

Delgrande, J.; 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.; Schaub, T.; and Tompits, H. 2003. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming* 3(2):129–187.

Dimopoulos, Y., and Torres, A. 1996. Graph theoretical structures in logic programs and default theories. *Theor. Comput. Sci.* 170(1-2):209–244.

García, A. J., and Simari, G. R. 2004. Defeasible logic programming: An argumentative approach. *TPLP* 4(1-2):95–138.

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. 2001. A comparative study of logic programs with preference. In *IJCAI*, 597–602.

Sefranek, J. 2006. Rethinking semantics of dynamic logic programs. In *Proc. of the Workshop NMR*.

Wang, K.; Zhou, L.; and Lin, F. 2000. Alternating fixpoint theory for logic programs with priority. In *Computational Logic*, 164–178.

On Planning with Preferences in HTN*

Shirin Sohrabi and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada.
{shirin,sheila}@cs.toronto.edu

Abstract

In this paper, we address the problem of generating preferred plans by combining the procedural control knowledge specified by Hierarchical Task Networks (HTNs) with rich qualitative user preferences. The outcome of our work is a language for specifying user preferences, tailored to HTN planning, together with a provably optimal preference-based planner, **HTNPLAN**, that is implemented as an extension of **SHOP2**. To compute preferred plans, we propose an approach based on forward-chaining heuristic search. Our heuristic uses an admissible evaluation function measuring the satisfaction of preferences over partial plans. Our empirical evaluation demonstrates the effectiveness of our **HTNPLAN** heuristics. We prove our approach sound and optimal with respect to the plans it generates by appealing to a situation calculus semantics of our preference language and of HTN planning. While our implementation builds on **SHOP2**, the language and techniques proposed here are relevant to a broad range of HTN planners.

1 Introduction

Hierarchical Task Network (HTN) planning is a popular and widely used planning paradigm, and many domain-independent HTN planners exist (e.g., **SHOP2**, SIPE-2, I-X/I-PLAN, O-PLAN) (Ghallab, Nau, and Traverso 2004). In HTN planning, the planner is provided with a set of tasks to be performed, possibly together with constraints on those tasks. A plan is then formulated by repeatedly decomposing tasks into smaller and smaller subtasks until primitive, executable tasks are reached. A primary reason behind HTN's success is that its task networks capture useful procedural control knowledge—advice on how to perform a task—described in terms of a decomposition of subtasks. Such control knowledge can significantly reduce the search space for a plan while also ensuring that plans follow one of the stipulated courses of action. However, while HTNs specify a family of satisfactory plans, they are, for the most part, unable to distinguish high-quality plans.

In this paper, we address the problem of generating preferred plans by augmenting HTN planning problems with rich qualitative user preferences. User preferences can be arbitrarily complex, often involving combinations of conditional, interacting, and mutually exclusive preferences that can range over multiple states of a plan. This makes finding an optimal plan hard. There are two aspects to addressing the problem of preference-based planning with HTNs. The first is to propose a preference specification language that is tailored to HTN planning. The second, is to generate preferred, and ideally optimal, plans efficiently.

To specify user preferences, we augment a rich qualitative preference language, $\mathcal{LPP}$, proposed in (Bienvenu, Fritz, and McIlraith 2006) with HTN-specific constructs. $\mathcal{LPP}$ specifies preferences in a variant of linear temporal logic (LTL). Among the HTN-specific properties that we add to our language, $\mathcal{LPH}$, is the ability to express preferences over how tasks in our HTN are decomposed into subtasks, preferences over the parameterizations of decomposed tasks, and a variety of temporal and nontemporal preferences over the task networks themselves.

To compute preferred plans, we propose an approach based on forward-chaining heuristic search. Key to our approach is a means of evaluating the (partial) satisfaction of preferences during HTN plan generation based on progression. The optimistic evaluation of preferences yields an admissible evaluation function which we use to guide search. We implemented our planner, **HTNPLAN**, as an extension to the **SHOP2** HTN planner. Our empirical evaluation demonstrates the effectiveness of **HTNPLAN** heuristics in finding high-quality plans. We provide a semantics for our preference language in the situation calculus (Reiter 2001) and appeal to this semantics to prove the soundness and optimality of our planner with respect to the plans it generates.

In Section 2, we review HTN planning, situation calculus, Golog, ConGolog, and provide an encoding of a preference-based HTN planning problem. In Section 3, we provide the syntax and the semantics for our preference language. In Section 4, we turn our attention to computing preferred plans describing how we evaluate the satisfaction of preferences over partial plans using progression. In Section 5, we describe the implementation of our HTN preference-based planner, **HTNPLAN** that is built on top of **SHOP2**, and provide empirical results that establish the effectiveness of our evaluation function in guiding search. We conclude with a summary and discussion of related work.

*A shorter version of this paper appears in the Proceedings of the Fourth Multidisciplinary Workshop on Advances in Preference Handling (MPref 2008).

2 HTN Planning

In this section, we provide a brief overview of both HTN planning, following (Ghallab, Nau, and Traverso 2004), and our situation calculus encoding of preference-based HTN planning.

Travel Example: Consider a simple HTN planning problem to address the task of arranging travel. This task can be decomposed into arranging transportation, accommodations, and local transportation. Each of these tasks can again be decomposed based on alternative modes of transportation and accommodations, reducing eventually to primitive actions that can be executed in the world. Further constraints can be imposed to restrict decompositions.

Definition 1 (HTN Planning Problem) *An HTN planning problem is a 3-tuple $\mathcal{P} = (s_0, w, D)$ where s_0 is the initial state, w is a task network called the initial task network, and D is the HTN planning domain. $\mathcal{P}$ is a total-order planning problem if w and D are totally ordered; otherwise it is said to be partially ordered.*

A *task* consists of a task symbol and a list of arguments. A task is primitive if its task symbol is an operator name and its parameters match, otherwise it is *nonprimitive*. In our example, *arrange-trans* and *arrange-acc* are nonprimitive tasks, while *book-flight* and *book-car* are primitive tasks.

Definition 2 (Task Network) *A task network is a pair w=(U, C) where U is a set of task nodes and C is a set of constraints. Each task node u ∈ U contains a task t_u. If all of the tasks are ground then w is ground; If all of the tasks are primitive, then w is called primitive; otherwise is called nonprimitive. Task network w is totally ordered if C defines a total ordering of the nodes in U.*

In our example, we could have a task network (U, C) where $U = \{u_1, u_2\}$, u_1 =*book-car*, and u_2= *pay*, and C is a precedence constraint such that u_1 must occur before u_2 and a before-constraint such that at least one car is available for rent before u_1.

A domain is a pair $D = (O, M)$ where O is a set of operators and M is a set of methods. Operators are essentially primitive actions that can be executed in the world. They are described by a triple *o =(name(o), pre(o), eff(o))*, corresponding to the operator's name, preconditions and effects. Preconditions are restricted to a set of literals, and effects are described as STRIPS-like Add and Delete lists. An operator o can accomplish a ground primitive task in a state s if their names match and o is applicable in s. In our example, ignoring the parameters, operators might include: *pay, book-train, book-car, book-hotel*, and *book-flight*.

A method, m, is a 4-tuple (*name(m), task(m),subtasks(m), constr(m))* corresponding to the method's name, a nonprimitive task and the method's task network, comprising subtasks and constraints. A method is *totally ordered* if its task network is *totally ordered*. A domain is a total-order domain if every $m \in M$ is *totally ordered*. Method m is relevant for a task t if there is a substitution σ such that $\sigma(t)$ =*task*(m). Several different methods can be relevant to a particular nonprimitive task t, leading to different decompositions of t. In our example, the method with *name by-flight-trans* can be used to decompose the *task arrange-trans* into the *subtasks* of booking a flight and paying, with the constraint (*constr*) that the booking precede payment.

Definition 3 (Solution to HTN Planning Problem) *Given HTN planning problem $\mathcal{P} = (s_0, w, D)$, a plan $\pi = (o_1, ..., o_k)$ is a solution for $\mathcal{P}$, depending on these two cases: 1) if w is primitive, then there must exist a ground instance (U′, C′) of (U, C) and a total ordering $(u_1, ..., u_k)$ of the nodes in U' such that for all $1 \leq i \leq k$, name(o_i) = t_{u_i}, the plan π is executable in the state s_0, and all the constraints hold, 2) if w is nonprimitive, then there must exist a sequence of task decompositions that can be applied to w to produce a primitive task network w', where π is a solution for w'.*

Finally, we define the HTN preference-based planning problem. This definition appeals to two concepts that are not yet well-defined and which we defer to later sections: definitions of the form and content of the the formula Φ_{htn} that captures user preferences for HTN planning as well as and the precise definition of *more preferred* appears in Section 3.

Definition 4 (Preference-based HTN Planning) *An HTN planning problem with user preferences is described as a 4-tuple $\mathcal{P} = (s_0, w, D, \Phi_{htn})$ where Φ_{htn} is a formula describing user preferences. A plan π is a solution to $\mathcal{P}$ if and only if: π is a plan for $\mathcal{P}' = (s_0, w, D)$ and there does not exists a plan π' such that π' is more preferred than π with respect to the preference formula Φ_{htn}.*

2.1 Situation Calculus Specification of HTN

We now have a definition of preference-based HTN planning. Later in the paper, we propose an approach to computing preferred plans, together with a description of our implementation. To prove the correctness and optimality of our algorithm, we appeal to an existing situation calculus encoding of HTN planning, which we augment and extend to provide an encoding of preference-based HTN planning. Since the situation calculus has a well-defined semantics, we have a semantics for our encoding which we use in our proofs. In this section, we review the salient features of this encoding.

The Situation Calculus is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular *situation* s, e.g., $F(\vec{x}, s)$. A situation s is a *history* of the primitive actions, $a \in \mathcal{A}$, performed from a distinguished initial situation S_0. The function $do(a, s)$ maps a situation and an action into a new situation thus inducing a tree of situations rooted in S_0. A *basic action theory* in the situation calculus $\mathcal{D}$ includes *domain independent foundational axioms*, and *domain dependent axioms*. A situation s' precedes a situation s, i.e., $s' \sqsubset s$, means that the sequence s' is a proper prefix of sequence s.

Golog (Reiter 2001) is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation

calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions (*programs*) δ. Example complex actions include action sequences, if-then-else, while loops, nondeterministic choice of actions and action arguments, and procedures. These complex actions serve as constraints upon the situation tree. ConGolog (De Giacomo, Lespérance, and Levesque 2000) is the concurrent version of Golog in which the language can additionally deal with execution of concurrent processes, interrupts, prioritized concurrency, and exogenous actions.

A number of researchers have pointed out the connection between HTN and ConGolog. Following Gabaldon (Gabaldon 2002), we map an HTN state to a situation calculus *situation*. Consequently, the initial HTN state s_0 is encoded as the initial situation, S_0. The HTN domain description maps to a corresponding situation calculus domain description, $\mathcal{D}$, where for every operator o there is a corresponding primitive action a, such that the preconditions and the effects of o are axiomatized in $\mathcal{D}$. Every method and nonprimitive task together with constraints is encoded as a ConGolog procedure. For the purposes of this paper, the set of procedures in a ConGolog domain theory is referred to as $\mathcal{R}$.

We use a predicate *badSituation*(s) proposed by Reiter (Reiter 2001) to encode the constraints in a task network. The purpose of these constraints is to prune part of a search space similar to using temporal constraints.

To deal with partially ordered task networks, we add two new primitive actions *start*$(P(\vec{v}))$, *end*$(P(\vec{v}))$, and two new fluents *executing*$(P(\vec{v}), s)$ and *terminated*(X, s), where $P(\vec{v})$ is a ConGolog procedure and X is either $P(\vec{v})$ or an action $a \in \mathcal{A}$. *executing*$(P(\vec{v}), s)$ states that $P(\vec{v})$ is executing in situation s, *terminated*(X, s) states that X has terminated in s. *executing*(a, s) where $a \in \mathcal{A}$ is defined to be false. The successor state axioms for these fluents follow. They show how the actions *start*$(P(\vec{v}))$, *end*$(P(\vec{v}))$ change the truth value of these fluents:

$$\begin{aligned} &\textit{executing}(P(\vec{v}), do(a, s)) \equiv a = \textit{start}(P(\vec{v})) \vee \\ &\quad \textit{executing}(P(\vec{v}), s) \wedge a \neq \textit{end}(P(\vec{v})) \\ &terminated(X, do(a, s)) \equiv X = a \vee \\ &\quad (X \in \mathcal{R} \wedge a = \textit{end}(X)) \vee \textit{terminated}(X, s) \end{aligned}$$

where $\mathcal{R}$ is the set of ConGolog procedures in our domain.

Definition 5 (Preference-based HTN in Situation Calculus) *An HTN planning problem with user preferences described as a 4-tuple $\mathcal{P} = (s_0, w, D, \Phi_{htn})$ is encoded in situation calculus as a 5-tuple $(\mathcal{D}, \mathcal{C}, \Delta, \delta_0, \Phi_{sc})$ where $\mathcal{D}$ is the basic action theory, $\mathcal{C}$ is the set of ConGolog axioms,Δ is the sequence of procedure declarations for all ConGolog procedures in $\mathcal{R}$, δ_0 is an encoding of the initial task network in ConGolog, and Φ_{sc} is a mapping of the preference formula Φ_{htn} in situation calculus. A plan $\vec{a}$ is a solution to the encoded preference-based HTN problem if and only if:*

$$\begin{aligned} \mathcal{D} \cup \mathcal{C} \models\ & (\exists s) Do(\Delta; \delta_0, S_0, s) \wedge s = do(\vec{a}, S_0) \\ & \wedge \neg \textit{badSituation}(s) \wedge \nexists s'.[Do(\Delta; \delta_0, S_0, s') \\ & \wedge \neg \textit{badSituation}(s') \wedge pref(s', s, \Phi_{sc})] \end{aligned}$$

where $pref(s', s, \Phi_{sc})$ denotes that the situation s' is preferred to situation s with respect to the preference formula Φ_{sc}, and $Do(\delta, S_0, do(\vec{a}, S_0))$ denotes that the ConGolog program δ, starting execution in S_0 will legally terminate in situation $do(\vec{a}, S_0)$. Removing all the *start*$(P(\vec{v}))$ and $end(P(\vec{v}))$ actions from $\vec{a}$ to obtain $\vec{b} = (b_1, ..., b_n)$, a preferred plan for the original HTN planning problem $\mathcal{P}$ is a plan $\pi = (o_1, ..., o_n)$ where for all $1 \leq i \leq n$, name(o_i)= b_i.

3 HTN Preference Specification

In this section, we describe how to specify the preference formula Φ_{htn}. Our preference language, $\mathcal{LPH}$, modifies and extends the $\mathcal{LPP}$ qualitative preference language proposed in (Bienvenu, Fritz, and McIlraith 2006) to capture HTN-specific preferences.

Our $\mathcal{LPH}$ language has the ability to express preferences over certain parameterization of a task (e.g., preferring one task grounding to another), over a certain decomposition of nonprimitive tasks (i.e., prefer to apply a certain method over another), and a soft version of the before, after, and in between constraints. A soft constraint is defined via a preference formula whose evaluation determines when a plan is *more preferred* than another. However, unlike the task network constraints which will prune or eliminate those plans that have not satisfied them, not meeting a soft constraint simplify deems a plan to be of poorer quality.

Definition 6 (Basic Desire Formula (BDF)) *A basic desire formula is a sentence drawn from the smallest set $\mathcal{B}$ where:*

1. *If l is a literal, then $l \in \mathcal{B}$ and **final**$(l) \in \mathcal{B}$*
2. *If t is a task, then **occ**$(t) \in \mathcal{B}$*
3. *If m is a method, and $n = name(m)$, then **apply**$(n) \in \mathcal{B}$*
4. *If t_1, and t_2 are tasks, and l is a literal, then **before**(t_1, t_2), **holdBefore**(t_1, l), **holdAfter**(t_1, l), **holdBetween**(t_1, l, t_2) are in $\mathcal{B}$.*
5. *If φ_1 and φ_2 are in $\mathcal{B}$, then so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$,$(\exists x)\varphi_1$, $(\forall x)\varphi_1$, **next**(φ_1), **always**(φ_1), **eventually**(φ_1), and **until**(φ_1, φ_2).*

final(l) states that the literal l holds in the final state, **occ**(t) states that the task t occurs in the present state, and **next**(φ_1), **always**(φ_1), **eventually**(φ_1), and **until**(φ_1, φ_2) are basic LTL constructs. **apply**(n) states that a method whose name is n is applied to decompose a nonprimitive task. **before**(t_1, t_2) states a precedence ordering between two tasks. **holdBefore**(t_1, l), **holdAfter**(t_1, l), **holdBetween**(t_1, l, t_2) state a soft constraint over when the fluent l is preferred to hold. (i.e., **holdBefore**(t_1, l) state that l must be true right before the last operator descender of t_1 occurs). Combining **occ**(t) with the rest of $\mathcal{LPH}$ language enables the construction of preference statements over parameterizations of tasks.

BDFs establish properties of different states within a plan. By combining BDFs using boolean and temporal connectives, we are able to express other properties of state. The

following are a few examples from our travel domain[1].

$$(\exists c).\mathbf{occ}'(\textit{book-car}(c, Enterprise)) \quad \text{(P1)}$$
$$\mathbf{apply}'(\textit{by-car-local(SUV, Avis)}) \quad \text{(P2)}$$
$$\mathbf{before}(\textit{arrange-trans}, \textit{arrange-acc}) \quad \text{(P3)}$$
$$\mathbf{holdBefore}(hotelReservation, \textit{arrange-trans}) \quad \text{(P4)}$$
$$\mathbf{always}(\neg(\mathbf{occ}'(\textit{pay}(Mastercard)))) \quad \text{(P5)}$$
$$(\exists h, r).\mathbf{occ}'(\textit{book-hotel}(h, r)) \wedge \textit{starsGE}(r, 3) \quad \text{(P6)}$$
$$(\exists c).\mathbf{occ}'(\textit{book-flight}(c, \textit{Economy}, \textit{Direct}, \textit{WindowSeat})) \wedge \textit{member}(c, \textit{StarAlliance}) \quad \text{(P7)}$$

P1 states that at some point the user books a car with Enterprise. P2 states that at some point, the *by-car-local* method is applied to book an SUV from Avis. P3 states that the *arrange-trans* task occurs before the *arrange-acc* task. P4 states that the hotel is reserved before transportation is arranged. P5 states that the user never pays by Mastercard. P6 states that at some point the user books a hotel that has a rating of 3 or more. P7 states that at some point the user books a direct economy window-seated flight with a Star Alliance carrier.

To define a preference ordering over alternative properties of states, *Atomic Preference Formulae* (APFs) are defined. Each alternative comprises two components: the property of the state, specified by a BDF, and a *value* term which stipulates the relative strength of the preference.

Definition 7 (Atomic Preference Formula (APF)) *Let $\mathcal{V}$ be a totally ordered set with minimal element v_{min} and maximal element v_{max}. An atomic preference formula is a formula $\varphi_0[v_0] \gg \varphi_1[v_1] \gg ... \gg \varphi_n[v_n]$, where each φ_i is a BDF, each $v_i \in \mathcal{V}$, $v_i < v_j$ for $i < j$, and $v_0 = v_{min}$. When $n = 0$, atomic preference formulae correspond to BDFs.*

While one could let $\mathcal{V} = [0, 1]$, you could choose a strictly qualitative set like $\{$*best* $<$ *good* $<$ *indifferent* $<$ *bad* $<$ *worst*$\}$ to express preferences over alternatives.

Now here are a few APF examples from the travel domain.

$$P2[0] \gg \mathbf{apply}'(\textit{by-car-local(SUV, National)})[0.3] \quad \text{(P8)}$$
$$\mathbf{apply}'(\textit{by-car-trans})[0] \gg \mathbf{apply}'(\textit{by-flight})[0.4] \quad \text{(P9)}$$
$$\mathbf{occ}'(\textit{book-train})[0] \gg \mathbf{occ}'(\textit{book-car})[0.4] \quad \text{(P10)}$$

P8 states that the user prefers that the *by-car-local* method rents an SUV and that the rental car company Avis is preferred to National. P9 states that the user prefers to decompose the *arrange-trans* task by the method *by-car-trans* rather than the *by-flight* method. Note that the task is implicit in the definition of the method. P10 states that the user prefers travelling by train over renting a car.

To allow the user to specify more complex preferences and to aggregate preferences, General Preference Formulae (GPFs) extend the language to conditional, conjunctive, and disjunctive preferences.

[1]To simplify the examples many parameters have been suppressed, and we abbreviate **eventually**(**occ**(φ)) by **occ**$'$, **eventually**(**apply**(φ)) by **apply**$'$ and refer to preferences by their labels.

Definition 8 (General Preference Formula (GPF)) *A formula Φ is a GPF if one of the following holds:*

- Φ *is an APF*
- Φ *is* $\gamma : \Psi$, *where* γ *is a BDF and* Ψ *is a GPF [Conditional]*
- Φ *is one of* $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ *[General Conjunction]* *or* $\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n$ *[General Disjunction]*

where $n \geq 1$ and each Ψ_i is a GPF.

General conjunction (resp.general disjunction) refines the ordering defined by $\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ (resp. $\Psi_0|\Psi_1|...|\Psi_n$) by sorting indistinguishable states using the lexicograping ordering. Continuing our example:

$$\mathbf{occ}(\textit{arrange-trans}) : (\exists c).\mathbf{occ}'(\textit{book-car}(c, Avis)) \quad \text{(P11)}$$
$$\mathbf{occ}(\textit{arrange-local-trans}) : P1 \quad \text{(P12)}$$
$$drivable : P10[0] \gg \mathbf{occ}'(\textit{book-flight})[0.3] \quad \text{(P13)}$$
$$P4 \,\&\, P6 \,\&\, P7 \,\&\, P8 \,\&\, P9 \,\&\, P10 \,\&\, P12 \,\&\, P13 \quad \text{(P14)}$$

P11 states that if inter-city transportation is being arranged then the user prefers to rent a car from Avis. P12 states that if local transportation is being arranged the user prefers Enterprise. P13 states that if the distance between the origin and the destination is drivable then the user prefers to book a train over booking a car over booking a flight. P14 aggregates preferences into one formula.

Again, and only for the purpose of proving properties, we provide an encoding of the HTN-specific terms of $\mathcal{LPH}$ in the situation calculus. As such, for any preference formula Φ_{htn} there is a corresponding formula Φ_{sc} where every HTN-specific term is replaced as follows: each literal l is mapped to a fluent or non-fluent relation in the situation calculus, as appropriate; each primitive task t is mapped to an action $a \in \mathcal{A}$; and each nonprimitive task t and each method m is mapped to a procedure $P(\vec{v}) \in \mathcal{R}$ in ConGolog.

3.1 The Semantics

The semantics of $\mathcal{LPH}$ is achieved through assigning a weight to a situation s with respect to a GPF, Φ, written $w_s(\Phi)$. This weight is a composition of its constituents. For BDFs, a situation s is assigned the value v_{min} if the BDF is satisfied in s, v_{max} otherwise. Similarly, given an APF, and a situation s, s is assigned the weight of the best BDF that it satisfies within the defined APF. Finally GPF semantics follow the natural semantics of boolean connectives. As such General Conjunction yields the minimum of its constituent GPF weights and General Disjunction yields the maximum.

Similar to (Gabaldon 2004) and following $\mathcal{LPP}$, we use the notation $\varphi[s', s]$ to denote that φ holds in the sequence of situations starting from s' and terminating in s. Next, we will show how to interpret BDFs in the situation calculus.

If f is a fluent, we will write $f[s', s] = f[s']$ since fluents are represented in situation-suppressed form. If r is a non-fluent, we will have $r[s', s] = r$ since r is already a situation calculus formula. Furthermore, we will write $\mathbf{final}(f)[s', s] = f[s]$ since $\mathbf{final}(f)$ means that the fluent f must hold in the final situation.

The BDF $\mathbf{occ}(X)$ states the occurrence of X which can be either an action or a procedure. written as:

$$\textbf{occ}(X)[s',s] = \begin{cases} do(X,s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(start(X),s') \sqsubseteq s & \text{if X} \in \mathcal{R} \end{cases}$$

The BDF **apply**$(P(\vec{v}))$ will be interpreted as follows:

$$\textbf{apply}(P(\vec{v}))[s',s] = do(start(P(\vec{v})),s') \sqsubseteq s$$

Boolean connectives and quantifiers are already part of the situation calculus and require no further explanation here. The LTL constructs are interpreted in the same way as in (Gabaldon 2004). We interpret the rest of the connectives as follows [2].

$$\textbf{before}(X_1,X_2)[s',s] = (\exists s_1,s_2 : s' \sqsubseteq s_1 \sqsubseteq s_2 \sqsubseteq s) \{\textit{terminated}(X_1)[s_1] \wedge \neg \textit{executing}(X_2)[s_1] \wedge \neg \textit{terminated}(X_2)[s_1] \wedge \textbf{occ}(X_2)[s_2,s]\}$$

$$\textbf{holdBefore}(X,f)[s',s] = (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s) \{f[s_1] \wedge \textbf{occ}(X)[s_1,s]\}$$

$$\textbf{holdAfter}(X,f)[s',s] = (\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s) \{terminated(X)[s_1] \wedge f[s_1]\}$$

$$\textbf{holdBetween}(X_1,f,X_2)[s',s] = (\exists s_1,s_2 : s' \sqsubseteq s_1 \sqsubseteq s_2 \sqsubseteq s) \{\textit{terminated}(X_1)[s_1] \wedge \neg \textit{executing}(X_2)[s_1] \wedge \neg \textit{terminated}(X_2)[s_1] \wedge \textbf{occ}(X_2)[s_2,s]\} \wedge (\forall s_i : s_1 \sqsubseteq s_i \sqsubseteq s_2) f[s_i]$$

From here, the semantics follows that of $\mathcal{LPP}$.

Definition 9 (Basic Desire Satisfaction) *Let $\mathcal{D}$ be an action theory, and let s' and s be situations such that $s' \sqsubseteq s$. The situations beginning in s' and terminating in s satisfy φ just in the case that $\mathcal{D} \models \varphi[s',s]$. We define $w_{s',s}(\varphi)$ to be the weight of the situations originating in s' and ending in s wrt BDF φ. $w_{s',s}(\varphi) = v_{min}$ if φ is satisfied, otherwise $w_{s',s}(\varphi) = v_{max}$.*

Note that for readability we are going to drop s' from the index, i.e., $w_s(\varphi) = w_{s',s}(\varphi)$ in the special case of $s' = S_0$.

Definition 10 (Atomic Preference Satisfaction) *Let s be a situation and $\Phi = \varphi_0[v_0] \gg \varphi_1[v_1] \gg ... \gg \varphi_n[v_n]$ be an atomic preference formula. Then $w_s(\Phi) = v_i$ if $i = min_j\{\mathcal{D} \models \varphi_j[S_0,s]\}$, and $w_s(\Phi) = v_{max}$ if no such i exists.*

Definition 11 (General Preference Satisfaction) *Let s be a situation and Φ be a general preference formula. Then $w_s(\Phi)$ is defined as follows:*

- $w_s(\varphi_0 \gg \varphi_1 \gg ... \gg \varphi_n)$ *is defined above*
- $w_s(\gamma : \Psi) = \begin{cases} v_{min} & \text{if } w_s(\gamma) = v_{max} \\ w_s(\Psi) & \text{otherwise} \end{cases}$
- $w_s(\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n) = \max\{w_s(\Psi_i) : 1 \le i \le n\}$
- $w_s(\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n) = min\,\{w_s(\Psi_i) : 1 \le i \le n\}$

The following definition dictates how to compare two situations (and thus two plans) with respect to a GPF. This preference relation $pref$ is used to compare HTN plans in Definition 5 and provides the semantics for *more preferred* in Definition 4.

Definition 12 (Preferred Situations) *A situation s_1 is at least as preferred as a situation s_2 with respect to a GPF Φ, written $pref(s_1,s_2,\Phi)$ if $w_{s_1}(\Phi) \le w_{s_2}(\Phi)$.*

[2] We use the following abbreviations:
$(\exists s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\Phi = (\exists s_1)\{s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s \wedge \Phi\}$
$(\forall s_1 : s' \sqsubseteq s_1 \sqsubseteq s)\Phi = (\forall s_1)\{[s' \sqsubseteq s_1 \wedge s_1 \sqsubseteq s] \subset \Phi\}$

4 Computing Preferred Plan

To compute a preferred plan, we proposed a heuristic-search, forwarding-chaining planner that searchs for the *most preferred* terminating state that satisfies the HTN planning problem. The search is guided by an admissible evaluation function that evaluates partial plans with respect to preference satisfaction. We use *progression* to evaluate the preference formula satisfaction over partial plans.

4.1 Progression

Given a situation and a temporal formula, progression evaluates it with respect to the state of a situation to generate a new formula representing those aspects of the formula that remain to be satisfied. In this section, we define the progression of the constructs we added/modified from $\mathcal{LPP}$ and show that progression preserves the semantics of preference formulae. To define the progression, similar to (Bienvenu, Fritz, and McIlraith 2006) we add the propositional constants TRUE and FALSE to both the situation calculus and to our set of BDFs, where $\mathcal{D} \models$ TRUE and $\mathcal{D} \not\models$ FALSE for every action theory $\mathcal{D}$. We also add the BDF **occNext**(X), and **applyNext**$(P(\vec{v}))$ to capture the progression of **occ**(X) and **apply**$(P(\vec{v}))$. Below we show the progression of the added constructs.

Definition 13 (Progression) *Let s be a situation, and let φ be a BDF. The progression of φ through s, written $\rho_s(\varphi)$, is given by:*

- *If φ=**occ**(X) then*
  $\rho_s(\varphi)$ = ***occNext***$(X) \wedge$***eventually***$(terminated(X))$
- *If $\varphi =$ **occNext**(X) , then*
  $\begin{cases} \textit{TRUE} & \textit{if } X \in \mathcal{A} \wedge \mathcal{D} \models \exists s'.s = do(X,s') \\ \textit{TRUE} & \textit{if } X \in \mathcal{R} \wedge \mathcal{D} \models \exists s'.s = do(start(X),s') \\ \textit{FALSE} & \textit{otherwise} \end{cases}$
- *If $\varphi =$ **apply**$(P(\vec{v}))$, then*
  $\rho_s(\varphi) =$ ***applyNext***$(P(\vec{v})) \wedge$***eventually***$(terminated(P(\vec{v})))$
- *If $\varphi =$ **applyNext**$(P(\vec{v}))$, then*
  $\rho_s(\varphi) = \begin{cases} \textit{TRUE} & \textit{if } \mathcal{D} \models \exists s'.s = do(start(P(\vec{v})),s') \\ \textit{FALSE} & \textit{otherwise} \end{cases}$
- *If $\varphi =$ **before**(X_1,X_2), **holdBefore**(X,f), **holdAfter**(X,f), or **holdBetween**$(X_1,f,X2)$, then*
  $\rho_s(\varphi) = \begin{cases} \textit{TRUE} & \textit{if } w_s(\varphi) = v_{min} \\ \textit{FALSE} & \textit{otherwise} \end{cases}$

To see how the other constructs are progressed please refer to (Bienvenu, Fritz, and McIlraith 2006).

4.2 Admissible Evaluation Function

In this section, we describe an admissible evaluation function using the notion of *optimistic* and *pessimistic* weights that provide a bound on the best and worst weights of any successor situation with respect to a GPF Φ. Optimistic (resp. pessimistic) weights, $w_s^{opt}(\Phi)$ (resp. $w_s^{pess}(\Phi)$) are defined based on optimistic (resp. pessimistic) satisfaction of BDFs. Optimistic satisfaction ($\varphi[s',s]^{opt}$) assumes that any parts of the BDF not yet falsified will eventually be satisfied. Pessimistic satisfaction ($\varphi[s',s]^{pess}$) assumes the opposite. The following definitions highlight the key differences between this work and the definitions in (Bienvenu, Fritz, and McIlraith 2006).

$$\textbf{occ}(X)[s', s]^{opt} \stackrel{\text{def}}{=} \begin{cases} do(X, s') \sqsubseteq s \vee s' = s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s \vee s' = s & \text{if X} \in \mathcal{R} \end{cases}$$

$$\textbf{occ}(X)[s', s]^{pess} \stackrel{\text{def}}{=} \begin{cases} do(X, s') \sqsubseteq s & \text{if X} \in \mathcal{A} \\ do(start(X), s') \sqsubseteq s & \text{if X} \in \mathcal{R} \end{cases}$$

$$\textbf{apply}(P(\vec{v}))[s', s]^{opt} \stackrel{\text{def}}{=} do(start(P(\vec{v})), s') \sqsubseteq s \vee s' = s$$

$$\textbf{apply}(P(\vec{v}))[s', s]^{pess} \stackrel{\text{def}}{=} do(start(P(\vec{v})), s') \sqsubseteq s$$

If $\varphi =$ **before**(X_1, X_2), **holdBefore**(X, f), **holdAfter**(X, f) **holdBetween**(X_1, f, X_2), then

$$\varphi[s', s]^{opt} \stackrel{\text{def}}{=} \varphi[s', s]^{pess} \stackrel{\text{def}}{=} w_{s',s}(\varphi)$$

Theorem 1 *Let $s_n = do([a_1, ..., a_n], S_0), n \geq 0$ be a collection of situations, φ be a BDF, Φ a general preference formula, and $w_s^{opt}(\Phi)$, $w_s^{pess}(\Phi)$ be the optimistic and pessimistic weights of Φ with respect to s. Then for any $0 \leq i \leq j \leq k \leq n$,*

1. $\mathcal{D} \models \varphi[s_i]^{pess} \Rightarrow \mathcal{D} \models \varphi[s_j]$, $\mathcal{D} \not\models \varphi[s_i]^{opt} \Rightarrow \mathcal{D} \not\models \varphi[s_j]$,

2. $\left(w_{s_i}^{opt}(\Phi) = w_{s_i}^{pess}(\Phi)\right) \Rightarrow w_{s_j}(\Phi) = w_{s_i}^{opt}(\Phi) = w_{s_i}^{pess}(\Phi)$,

3. $w_{s_i}^{opt}(\Phi) \leq w_{s_j}^{opt}(\Phi) \leq w_{s_k}(\Phi), w_{s_i}^{pess}(\Phi) \geq w_{s_j}^{pess}(\Phi) \geq w_{s_k}(\Phi)$

Theorem 1 states that the optimistic weight is non-decreasing and never over-estimates the real weight. Thus, f_Φ is admissible and when used in best-first search, the search is optimal.

Definition 14 (Evaluation function) *Let $s = do(\vec{a}, S_0)$ be a situation and let Φ be a general preference formula. Then $f_\Phi(s) \stackrel{\text{def}}{=} w_s(\Phi)$ if $\vec{a}$ is a plan, otherwise $f_\Phi(s) \stackrel{\text{def}}{=} w_s^{opt}(\Phi)$.*

5 Implementation and Results

In this section, we describe our best-first search, ordered-task-decomposition planner. Figure 1 outlines the algorithm. **HTNPLAN** takes as input $\mathcal{P} = (s_0, w, D, pref)$ where s_0 is the initial state, w the initial task network, D is the HTN planning domain, and *pref* the general preference formula, and returns a sequence of ground primitive operators, i.e. a plan, and the weight of that plan.

The *frontier* is a list of nodes of the form [*optW, pessW,* w*, partialP, s, pref*], sorted by optimistic weight, pessimistic weight, and then by plan length. The frontier is initialized to the initial task network w, the empty partial plan, its *optW, pessW,* and *pref* corresponding to the progression and evaluation of the input preference formula in the initial state.

On each iteration of the **while** loop, **HTNPLAN** removes the first node from the frontier and places it in *current*. If w is empty (i.e., U is an empty set), the situation associated with this node is a terminating situation. Then **HTNPLAN** returns *current*'s partial plan and weight. Otherwise, it calls the function **EXPAND** with *current's* node as input.

EXPAND returns a new list of nodes that need to be added to the frontier. The new nodes are sorted by *optW, pessW,* and merged with the remainder of the frontier. If *w* is *nil* then the frontier is left as is. Otherwise, it generates a new set of nodes of the form [*optW, pessW, newW, newPartialP, newS, newProgPref*], one for each legal ground operator that can be reached by performing w using a partial-order forward decomposition procedure (PFD) (Ghallab, Nau, and Traverso 2004). Currently **HTNPLAN** uses **SHOP2** (Nau et al. 2003) as its PFD. Hence, the current implementation of

HTNPLAN(s_0, *w, D, pref*)
frontier ← **INITFRONTIER**(s_0, *w, pref*)
while *frontier* ≠ ∅
  current ← **REMOVEFIRST**(*frontier*)
  % establishes values of *w, partialP, s, progPref*
  if *w*= ∅ and *optW=pessW* then **return** *partialP, optW*
  neighbours ← **EXPAND**(*w, D, partialP, s, progPref*)
  frontier ← **SORTNMERGE**(*neighbours, frontier*)
return [], ∞

Figure 1: A sketch of the **HTNPLAN** algorithm.

HTNPLAN is an implementation of **SHOP2** with user preferences. For each primitive task leading to terminating states, **EXPAND** generates a node of the same form but with *optW* and *pessW* replaced by the actual weight. If we reach the empty frontier, we return the empty plan.

Theorem 2 (Soundness and Optimality) *Let $\mathcal{P}$=(s_0, w, D, Φ) be a HTN planning problem with user preferences. Let π be the plan returned by* **HTNPLAN** *from input $\mathcal{P}$. Then π is a solution to the preference based HTN problem $\mathcal{P}$*

Proof sketch: We prove that the algorithm terminates appealing to the fact that the PFD procedure is sound and complete. We prove that the returned plan is optimal, by exploiting the correctness of progression of preference formula, and admissibility of our evaluation function.

5.1 Experiments

We implemented our preference-based HTN planner, **HTNPLAN**, on top of the LISP implementation of **SHOP2** (Nau et al. 2003). All experiments were run on a Pentium 4 HT, 3GHZ CPU, and 1 GB RAM, with a time limit of 1800 seconds (30 min). Since the optimality of **HTNPLAN**-generated plans was established in Theorem 2, our objective was to evaluate the effectiveness of our heuristics in guiding search towards the optimal plan, and to establish benchmarks for future study, since none currently exist.

We tested **HTNPLAN** with ZenoTravel and Logistics domains, which were adapted from the International Planning Competition (IPC). The ZenoTravel domain involves transporting people on aircrafts that can fly at two alternative speeds between locations. In the numeric variant the planes consume fuel at different rates according to the speed of travel, and distances between locations vary. The simple-time variant combines the speed of travel with the associated costs. We used both. The Logistics domain involves transporting packages to different destinations using trucks for delivery within cities and planes for between cities. Some of the preferences we used in the evaluations are as follows: we prefer that the high priority packages be delivered first, we prefer to use trucks with lower gas consumptions, and we prefer certain truck routes to another. The problems become harder as the number of objects and/or number of tasks in the domain increases.

In order to evaluate the effectiveness of **HTNPLAN** it would have been appealing to evaluate our planner with a preference-based planner that also makes use of procedural

P	SHOP2			HTNPLAN			
#	# Plan	NE	Time	NE	NC	Time	PL
1	12	172	0.61	78	88	1.19	22
2	155	1628	8.60	448	547	9.45	26
3	230	2234	11.15	76	97	1.05	23
4	230	2234	11.10	361	413	4.67	23
5	485	6331	74.10	240	276	8.14	38
6	487	6226	113.20	1084	1218	63.60	46
7	720	6724	50.46	211	250	4.63	31
8	720	6724	50.90	699	808	13.63	28
9	851	9152	165.22	2689	3066	142.7	40
10	2069	23200	205.10	2290	2733	91.25	34
11	2875	27022	369.20	609	704	17.20	30
12	3956	35789	275.30	304	361	5.10	22
13	>8K	>104K	>1800	150	167	5.64	63
14	>13K	>143K	>1800	2153	2922	80.01	35
15	>13K	>136K	>1800	1624	1910	36.02	29
16	>31K	>293K	>1800	1510	1848	24.80	21

(a) ZenoTravel domain

P	SHOP2			HTNPLAN			
#	# Plan	NE	Time	NE	NC	Time	PL
1	80	1297	1.27	73	93	0.64	14
2	90	540	0.28	19	24	0.20	12
3	808	4597	4.00	301	404	2.22	18
4	1024	10665	79.95	1626	1820	49.56	42
5	1024	10665	79.95	98	115	2.30	42
6	1260	6320	4.66	130	172	1.04	14
7	2178	15104	17.20	27	32	0.22	20
8	2520	14728	12.47	29	40	0.33	16
9	21776	114548	119.1	866	1163	9.44	15
10	>28K	>264K	>1800	1062	1437	13.21	19
11	>28K	>239K	>1800	1767	2417	32.76	14
12	>30K	>118K	>1800	1417	1925	21.07	20
13	>42K	>368K	>1800	2398	2968	82.62	42
14	>54K	>407K	>1800	858	1088	19.26	33
15	>65K	>428K	>1800	37	48	0.46	24
16	>67K	>376K	>1800	451	618	5.14	22

(b) Logistic domain

Figure 2: Our criteria for comparisons are number of Nodes Expanded (NE), number of applied operators; number of Nodes Considered (NC), the number of nodes that were added to the frontier, and time measured in seconds. Note NC is equal to NE for **SHOP2**. PL is the Plan Length and # Plan is the total number of plans.

control knowledge. But since no comparable planner exists, and it would not have been fair to compare **HTNPLAN** with a preference-based planner that does not use control knowledge, we compared **HTNPLAN** with **SHOP2**, using a brute-force technique for **SHOP2** to determine the optimal plan. In particular, as is often done with Markov Decision Processes, **SHOP2** generated all plans that satisfied the HTN specification and then evaluated each to find the optimal plan. Note that the times reported for **SHOP2** do not actually include the time for posthoc preference evaluation, so they are lower bounds on the time to compute the optimal plan.

Figure 2 reports our experimental results for ZenoTravel and the Logistics domain. The problems varied in preference difficulty and are shown in the order of difficulty with respect to number of possible plans (# Plan) that satisfy the HTN control.

The results show that, in all but the first two cases of the ZenoTravel domain, **SHOP2** required more time to find the optimal plan, and expanded more nodes. In particular note that in a number of problems, for example problems 13 and 14 **SHOP2** ran out of time (1800 seconds) while **HTNPLAN** found the optimal plan well within the time limit. Also note that **HTNPLAN** expands far fewer nodes in comparison to **SHOP2**, illustrating the effectiveness of our evaluation function in guiding search.

6 Summary and Related Work

In this paper, we addressed the problem of generating preferred plans by combining the procedural control knowledge of HTNs with rich qualitative user preferences. The most significant contributions of this paper include: $\mathcal{LPH}$, a rich HTN-tailored preference specification language, developed as an extension of a previously existing language; an approach to (preference-based) HTN planning based on forward-chaining heuristic search, that exploits progression to evaluate the satisfaction of preferences during planning; a sound and optimal implementation of an ordered-task-decomposition preference-based HTN planner; and leveraging previous research, an encoding of HTN planning with preferences in the situation calculus, that enabled us to prove our theoretical results. While the implementation we present here exploits **SHOP2**, the language and techniques proposed are relevant to a broad range of HTN planners.

In previous work, we addressed the problem of integrating user preferences into Web service composition (Sohrabi, Prokoshyna, and McIlraith 2006). To that end, we developed a Golog-based composition engine that also exploits heuristic search. It similarly uses an optimistic heuristic. The language used in that work was $\mathcal{LPP}$ and had no Web-service or Golog-specific extensions for complex actions. This paper's HTN-tailored language and HTN-based planner are significantly different.

Preference-based planning has been the subject of much interest in the last few years, spurred on by an International Planning Competition (IPC) track on this subject. A number of planners were developed, all based on the the competition's PDDL3 language (Gerevini and Long 2005). Our work is distinguished in that it exploits *procedural* (action-centric) domain control knowledge in the form of an HTN, and action-centric and state-centric preferences in the form of $\mathcal{LPH}$. In contrast, the preferences and domain control in PDDL3 and its variants are strictly state-centric. Further, $\mathcal{LPH}$ is *qualitative* whereas PDDL3 is quantitative, appealing to a numeric objective function. We contend that qualitative, action- or task-centric preferences are often more compelling and easier to elicit that their PDDL3 counterparts.

While no other HTN planner can perform true preference-based planning, **SHOP2** (Nau et al. 2003) and **ENQUIRER** (Kuter et al. 2004) handle some simple user constraints. In particular the order of methods and sorted preconditions in a domain description specifies a user preference over which method is more preferred to decompose a task. Hence users may write different versions of a domain description to specify simple preferences. However, unlike

HTNPLAN the user constraints are treated as hard constraints and (partial) plans that do not meet these constraints will be pruned from the search space. Further, there is no way to handle temporally extended hard or soft constraints in **SHOP2**. We used progression in our approach to planning precisely to deal with these interesting preferences. Were we limiting the expressive power of preferences to **SHOP2**-like method ordering, we would have created a different planner. Interestingly, **SHOP2** method ordering can still be exploited in our approach, but requires a mechanism that is beyond the scope of this paper.

Finally, the **ASPEN** planner (Rabideau, Engelhardt, and Chien 2000) performs a simple form of preference-based planning, focused mainly on preferences over resources and with far less expressivity than our preference language. Moreover, unlike our planner ASPEN will not perform well on problems where preferences are interacting, nested, and not local to any specific activity. Nevertheless, **ASPEN** has the ability to plan with HTN-like task decomposition, and as such, this work is related in spirit, though not in approach to our work.

Acknowledgements: We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Ministry of Research and Innovation Early Researcher Award.

References

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR)*, 134–144.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Gabaldon, A. 2002. Programming hierarchical task networks in the situation calculus. In *AIPS'02 Workshop on On-line Planning and Scheduling*.

Gabaldon, A. 2004. Precondition control and the progression algorithm. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR)*, 634–643. AAAI Press.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Hierarchical Task Network Planning. Automated Planning: Theory and Practice*. Morgan Kaufmann.

Kuter, U.; Sirin, E.; Nau, D. S.; Parsia, B.; and Hendler, J. A. 2004. Information gathering during planning for web service composition. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, 335–349.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Rabideau, G.; Engelhardt, B.; and Chien, S. A. 2000. Using generic preferences to incrementally improve plan quality. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 236–245.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Sohrabi, S.; Prokoshyna, N.; and McIlraith, S. A. 2006. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC)*, 597–611.

Special Session on Foundations of NMR and Uncertainty

The abilities to handle uncertain information and to reason nonmonotonically from incomplete knowledge are crucial features of intelligent behaviour in complex and dynamic environments. This is why reasoning under uncertainty and nonmonotonic reasoning are major research areas in Artificial Intelligence. The aim of this specialized workshop at NMR'2008 is to bring together researchers working in the intersection of both fields. This subworkshop will especially welcome papers that explore the relationship between formalisms developed within the two fields and that improve understanding and cross-fertilization between researchers active in reasoning under uncertainty and nonmonotonic reasoning.

Session Chairs

Alberto Finzi, University of Naples "Federico II", Italy
Frank Wolter, University of Liverpool, UK

Program Committee

Salem Benferhat, Université d'Artois, France
Marc Denecker, Katholieke Universiteit Leuven, Belgium
Lluis Godo, Institut d'Investigacó en Intelligència Artificial (IIIA), Spain
Gabriele Kern-Isberner, University of Dortmund, Germany
Thomas Lukasiewicz, Oxford University, UK
Fiora Pirri, University of Rome "La Sapienza", Italy
Choh Man Teng, University of West Florida, USA
Leon Van der Torre, University of Luxembourg, Luxembourg
Ronald R. Yager, Iona College, USA
Emil Weydert, Luxembourg University of Applied Sciences, Luxembourg
Mary-Anne Williams, University of Technology Sydney, Australia

Simple Generalized Default Theories

Alexander Bochman
Computer Science Department,
Holon Institute of Technology, Israel

Abstract

We single out a special class of what we call *simple* default theories in generalized default logic that involve only monotonic inference rules and unconditional ('supernormal') defaults. The resulting reasoning framework constitutes a generalization of an assumption-based framework for nonmonotonic reasoning suggested in (Bondarenko *et al.* 1997). On the other hand, it will be shown that there exists a polynomial translation of arbitrary generalized default theories to simple default theories.

Introduction

Default logic has been born as just one of a number of alternative formalizations of nonmonotonic reasoning. In the course of its development, however, it has become increasingly clear that it occupies a special place in nonmonotonic reasoning, both with respect to its representation capabilities, and in its relations to other nonmonotonic formalisms. The main objective of this study consists in confirming and elaborating the claim that default logic can serve as a general formalism for nonmonotonic reasoning, sufficient to deal with the majority of the tasks and problems posed to the latter by AI.

The plan of the paper is as follows. First, we describe a powerful generalization of the original, Reiter's default logic to disjunctive default rules that may contain, in addition, justifications in heads. Essentially, this generalization has been suggested first in (Lin and Shoham 1992; Lifschitz 1994) in a modal logical framework, and it subsumes disjunctive default logic of (Gelfond *et al.* 1991). Next, we describe a natural class of generalized default theories that contain only monotonic inference rules and default assumptions (aka supernormal defaults). By reformulating such theories in a different language, we will demonstrate that simple default theories constitute a principal instantiation of the general assumption-based framework for nonmonotonic reasoning suggested in (Bondarenko *et al.* 1997). Finally, we will show that the generalized default logic in its full generality is polynomially reducible to the formalism of simple default theories.

In what follows, we will invariably use upper case letters $A, B, \ldots$ for denoting propositions, while lower case letters $a, b, \ldots$ will denote sets of propositions.

Reiter's Default Logic

Originally, default theory was defined in (Reiter 1980) as a pair (W, D), where W is a set of classical propositions (the axioms), and D a set of default rules of the form $A : b/C$, where A, C are propositions and b a finite set of propositions. Very informally, a rule $A : b/C$ was intended to state something like: "If A is believed, and each $B \in b$ can be consistently assumed, then C should be believed".

The default rules were intended to act as meta-rules for extending the initial knowledge base W beyond what is strictly known. Accordingly, the nonmonotonic semantics of default logic was defined by determining a set of *extensions* of a default theory. An extension was defined by a fixed point construction: for a set u of propositions, let $\Gamma(u)$ be the least deductively closed set that includes W and satisfies the following condition:

- If $A : b/C$ is a default rule, $A \in \Gamma(u)$ and $\neg B \notin u$, for any $B \in b$, then $C \in \Gamma(u)$.

Then a set s is an *extension* of the default theory if and only if $\Gamma(s) = s$.

As can be seen, default claims were represented in default logic as inference *rules* affecting our beliefs. In this respect, Reiter's default logic has been largely inspired by the need to provide logical foundations for the procedural approach to nonmonotonicity found in deductive databases, logic programming and Doyle's truth maintenance. This representation avoided some of the problems arising with formula-based interpretations of defaults (such as contraposition of default claims). On the other hand, this representation has made default logic an inherently epistemic formalism. Namely, it primarily described our beliefs and knowledge, unlike the extensional classical logic normally used for a direct representation of objective facts about the world. This epistemic understanding can be clearly discerned from the original description given in (Reiter 1980). As Reiter put it, default rules function as meta-rules allowing us to extend our (incomplete) knowledge base with reasonable beliefs. In fact, the very notation for default rules initially used by Reiter, namely $A{:}\mathbf{M}B_1, \ldots, \mathbf{M}B_n/C$, involved a rudimen-

tary modal operator $\mathbf{M}$ employed for designating justifications. Actually, this operator was a 'remnant' of an alternative, modal approach to nonmonotonic reasoning developed by McDermott and Doyle in (McDermott and Doyle 1980). However, one of the main objectives of Reiter was to avoid modal logic or any modal concepts entirely and work only within a first order logical framework. In accordance with this, the operator $\mathbf{M}$ has been dropped later as syntactically unnecessary, though without changing the original epistemic understanding.

(Marek and Truszczyński 1989) have suggested a more logical description of default logic using the notion of a context-depended proof as a way of formalizing Reiter's operator Γ. This representation has been developed in (Marek *et al.* 1990) to a general theory of nonmonotonic rule systems (see also (Marek and Truszczyński 1993)).

Given a set s of propositions (the 'context'), let us consider the set $\mathcal{D}(s)$ of all propositions that are derivable from W using the classical entailment and the following ordinary inference rules that are allowed by the context:

$$\{A \vdash C \mid A : b/C \in D \ \& \ \neg B \notin s, \text{ for any } B \in b\}.$$

Then s is an extension of the default theory if and only if $s = \mathcal{D}(s)$.

It can be easily verified that the operator $\mathcal{D}$ coincides, in effect, with the operator Γ. Nevertheless, the above representation makes it vivid that a large part of reasoning in default logic involves ordinary rule-based inference, the only distinction from traditional inference systems being that the very set of rules allowed in the inference process is determined by the (assumptions made in the) context. In particular, an extension of a default theory can be viewed as a set of propositions that are provable (= justified) on the basis of taking itself as an assumption context.

As a starting point of the present study, let us point out to a certain logical discrepancy that was present in the overall design of Reiter's default logic. Namely, the monotonic, factual information was encoded in default logic primarily as classical propositions (included in the set W of axioms), while the inference rules were employed solely to deal with nonmonotonic extensions of the theory. As we will argue below, however, inference *rules* are actually essential for an adequate description of epistemic information in general, monotonic or not (at least if we don't want to use an explicit modal language). Consequently, we suggest that default logic should be re-constructed on a uniform basis of inference rules, both monotonic and nonmonotonic ones. More precisely, instead of a pair (W, D) of propositional axioms and default inference rules, we will begin with representing a default theory uniformly as a set of rules $A : b/C$, where b may be an empty set (in which case the default rule represents an ordinary monotonic inference rule $A \vdash B$).

Obviously, there is no loss of expressivity in this move, since any axiom A is expressible as an inference rule $\vdash A$. Moreover, this reformulation actually does not exceed the expressivity of the original default logic, since monotonic inference rules $A \vdash B$ are expressible, for example, as default rules $A : \mathbf{t}/B$[1]. The reformulation makes it clear, however, that the real informational basis of default logic is not just a set of propositions, but a set of inference rules. Actually, one of the important claims of this study is that this rule-based character of default logic is essential for its adequate use in applications. This shift in view allows us to explain, for instance, why the restriction of default logic to *normal* default theories (that was originally advocated by Reiter in (Reiter 1980)) is problematic from the point of view of expressivity, not so much because of the restriction of proper default rules to normal rules $A : B/B$, but rather because monotonic rules are completely eliminated as a by-product of this restriction. In fact, the main result of this study will amount to showing that, once the monotonic, rule-based background of default logic is preserved, its nonmonotonic 'overhead' is reducible even to super-normal defaults of the form $: A/A$.

But before this, the above rule-based reformulation of default logic suggests also its natural generalization which is based on extending its logical background from ordinary Tarski inference rules to more expressive disjunctive inference rules described in the next section.

Disjunctive (Scott) consequence relations

We will briefly describe below a well-known generalization of a theory of logical inference based on using disjunctive, multiple-conclusion rules $a \vdash b$, where a and b are sets of propositions. A generic informal interpretation of a disjunctive inference rule $a \vdash b$ is

If all propositions from a hold, then at least one proposition from b should hold.

Thus, the conclusions of such rules are interpreted as sets of alternatives implied by the premises. It turns out that such a disjunctive generalization of inference rules is essential for adequate representation of many important concepts and problem situations.

The importance of disjunctive rules for describing logical inference has been realized already by Gerhard Gentzen, the father of the sequent calculus. From a logical point of view, the main advantage of such disjunctive rules consists in providing more adequate and transparent encoding of semantic descriptions, and hence of the information we may have about the world or a problem. Indeed, taken in a negative form, a rule $a \vdash b$ says that it is impossible that all propositions from a hold, and no proposition from b holds. As can be seen from this description, such rules allow us to express equally well not only positive information about what holds in the situation, but also negative information about what does not hold. Of course, in a classical, world-oriented setting, such a negative information can be reduced to a single proposition (using the classical disjunction), and that is why ordinary, Tarski inference rules $a \vdash A$ are sufficient in such a semantic framework for almost all purposes. Furthermore, in such a setting we basically do not need rules at all for representing factual information, since any rule is reducible to the corresponding material implication. Things become more complex, however, in epistemic contexts where the semantic objects (such as belief states) are inherently partial.

[1] Or, alternatively as default rules $A : \ /B$ with an empty set of justifications.

In such contexts the information that A does not hold (i.e., is not believed) is weaker, in general, than the assertion that $\neg A$ holds (that is, $\neg A$ is believed), and consequently the inference rule $A \vdash B$ expresses a weaker claim than the material implication $A \supset B$ in the sense that $\vdash A \supset B$ implies $A \vdash B$, but not vice versa. Moreover, $A \vee B$ is weaker, in turn, than the claim that either A holds, or B holds. Consequently, the classical logical language turns out to be insufficient in such contexts for making some essential semantic distinctions, and it is here that the full expressive capabilities of disjunctive inference rules become a necessity.

The theory of disjunctive inference is now a well-developed part of the general logical theory. In a most abstract setting, it is provided by a theory of Scott consequence relations (see (Scott 1974) and (Gabbay 1976; 1981)), known also under the name multiple-conclusion consequence relations (cf. (Shoesmith and Smiley 1978; Segerberg 1982; Wojcicki 1988)).

We will restrict our description of disjunctive consequence relations below to what will be strictly necessary for what follows. Further details can be found in the above mentioned literature, as well as in (Bochman 2001; 2005).

As a preparation, let us recall that the usual *Tarski consequence relation* can be defined as a set of rules of the form $a \vdash A$ that satisfy the postulates

(Reflexivity) $A \vdash A$.

(Monotonicity) If $a \vdash A$ and $a \subseteq a'$, then $a' \vdash A$;

(Cut) If $a \vdash A$ and $a, A \vdash B$, then $a \vdash B$.

A Tarski consequence relation can also be described using the associated *provability operator* Cn defined as follows: $\mathrm{Cn}(u) = \{A \mid u \vdash A\}$. A *theory* of a Tarski consequence relation is a set of propositions u such that $u = \mathrm{Cn}(u)$. As is well-known, the set of theories of a Tarski consequence relation is closed with respect to arbitrary intersections, so any consistent set u of propositions is included in a unique least theory $\mathrm{Cn}(u)$.

Now, as a starting point, we take a disjunctive rule to be an expression of the form $a \vdash b$, where a and b are *finite* sets of propositions. A set of disjunctive rules is said to form a *Scott consequence relation* if it satisfies the following 'symmetric' generalization of the above postulates:

(Reflexivity) $A \vdash A$.

(Monotonicity) If $a \vdash b$ and $a \subseteq a'$, $b \subseteq b'$, then $a' \vdash b'$;

(Cut) If $a \vdash b, A$ and $a, A \vdash b$, then $a \vdash b$.

Though defined initially only for finite sets of propositions, the notion of a disjunctive rule can be extended to infinite sets of premises and conclusions by requiring that, for any sets of propositions u and v,

(Compactness) $u \vdash v$ if and only if $a \vdash b$, for some finite $a \subseteq u, b \subseteq v$.

This extension retains all the rules of a Scott consequence relation. Moreover, the resulting extended consequence relations could be characterized alternatively as Scott consequence relations on arbitrary sets of formulas satisfying the Compactness requirement.

In what follows, $\overline{u}$ will denote the complement of the set u of propositions. The next definition describes the basic notion of a theory for a Scott consequence relation.

Definition. A set u of propositions is a *theory* of a Scott consequence relation if $u \nvdash \overline{u}$.

Thus, theories are defined as sets of propositions that do not imply propositions outside them. The following lemma shows in what sense theories can also be seen as sets of propositions that are *closed* with respect to the rules of a disjunctive consequence relation.

Lemma 1. *A set u is a theory of a Scott consequence relation $\vdash$ if and only if, for any rule $a \vdash b$ from $\vdash$, if $a \subseteq u$, then $u \cap b \neq \emptyset$.*

Theories of a Scott consequence relation naturally generalize theories of a Tarski consequence relation, but they do not have all the properties of the latter. Most importantly, intersections of Scott theories are not in general theories. Nevertheless, due to compactness, we still have existence of inclusion minimal theories containing some set of propositions, and inclusion maximal theories disjoint from a set of propositions:

Theorem 2. *1. If u is a theory containing a set v of propositions, then u contains a minimal theory u' including v.*

2. If u is a theory included in a set v of propositions, then u is contained in a maximal theory u' included in v.

As a special case, we obtain the following useful property:

Corollary 3. *Any theory of $\vdash$ is included in a maximal theory and contains a minimal theory of $\vdash$.*

Any family of sets of propositions $\mathcal{T}$ determines a Scott consequence relation $\vdash_{\mathcal{T}}$ defined as follows:

$$a \vdash_{\mathcal{T}} b \quad \equiv \quad \text{For any } u \in \mathcal{T}, \text{ if } a \subseteq u, \text{ then } b \cap u \neq \emptyset. \tag{GS}$$

Actually, the above construction is quite general, since any Scott consequence relation can be generated in this way by the set of its theories – this is precisely the basic result about Scott consequence relations, called Scott Completeness Theorem in (Gabbay 1981).

Theorem 4. (**Scott Completeness Theorem**) *If $\mathcal{T}_{\vdash}$ is the set of all theories of a Scott consequence relation $\vdash$, then $\vdash$ coincides with $\vdash_{\mathcal{T}_{\vdash}}$.*

An important consequence of the above representation theorem is that Scott consequence relations are uniquely determined by their theories. Moreover, for more expressive languages the above representation theorem can serve as a basis of constructing full-fledged semantics. In this case the set of theories of a consequence relation will serve, eventually, as its canonical model.

By a *sequent theory* we will mean an arbitrary set of disjunctive rules. Due to the 'Horn' form of the postulates characterizing a Scott consequence relation, intersection of a set of Scott consequence relations is again a Scott consequence relation. This implies, in particular, the following

Lemma 5. *For any sequent theory there exists a least Scott consequence relation containing it.*

The least Scott consequence relation that contains a sequent theory Δ will be denoted by $\vdash_\Delta$. The latter consists of all the disjunctive rules that can be inferred from Δ using the postulates of Scott consequence relation. Hence, it describes, in a sense, the logical content of Δ. Accordingly, we will extend the notion of a theory to arbitrary sequent theories:

Definition. A set of propositions will be called a *(propositional) theory of a sequent theory* Δ, if it is a theory of $\vdash_\Delta$.

The next, almost obvious, result says that propositional theories of a sequent theory Δ are precisely the sets of propositions that are closed with respect to the rules from Δ.

Lemma 6. *A set u of propositions is a theory of Δ if and only if, for any rule $a \vdash b$ from Δ, if $a \subseteq u$, then $b \cap u \neq \emptyset$.*

Since any sequent theory determines a unique Scott consequence relation, we acquire a useful tool for constructing special consequence relations by restricting the corresponding sets of generating rules. Thus, the next definition introduces a class of Scott consequence relations that correspond to usual Tarski consequence relations.

Definition. A Scott consequence relation will be called *singular*, if it is generated by rules of the form $a \vdash A$, where A is a proposition.

The following lemma describes such consequence relations in terms of their theories.

Lemma 7. *A Scott consequence relation is singular if and only if the set of its theories is closed with respect to arbitrary intersections.*

As a consequence of the above result, any singular Scott consequence relation will always have a least theory.

Scott consequence relations provide an abstract description of disjunctive inference. Default logic also presupposes, however, classical first order inference as part of its logical machinery. In accordance with this, we should 'upgrade' our abstract notion of inference to consequence relations that subsume classical entailment. Such *supraclassical* consequence relations will turn out to be suitable for describing the logical basis of default logic.

From now on we will consider consequence relations that are formulated in a classical language containing the ordinary classical connectives $\{\vee, \wedge, \neg, \supset\}$. As usual, $\models$ will denote the classical entailment relation with respect to these connectives, and Th its associated classical derivability operator.

Definition. A Scott consequence relation in a classical language will be called *supraclassical*, if it satisfies:

Supraclassicality If $a \models A$, then $a \vdash A$.

Falsity $\mathbf{f} \vdash$.

By the first condition, theories of a supraclassical Scott consequence relation should be deductively closed. Falsity amounts, in addition, to exclusion of inconsistent theories from consideration. Accordingly, a Scott consequence relation will be supraclassical if and only if all its theories are consistent deductively closed sets.

Supraclassicality allows for replacement of classically equivalent formulas in premises and conclusions of disjunctive rules, as well as replacement of sets of premises by their classical conjunctions: $a \vdash b$ will be equivalent to $\bigwedge a \vdash b$. Disjunctive conclusions, however, cannot be replaced in this way by their classical disjunctions. Taking the simplest case, a rule $\vdash B, C$ is not reducible to $\vdash B \vee C$; the latter says that any theory should contain $B \vee C$, while the former asserts a stronger constraint that any theory should contain either B or C. As a result, the disjunctive character of our inference rules is not eliminated by supraclassicality. Speaking more generally, supraclassical Scott consequence relations are only *supra*-classical, which means, in particular, that the deduction theorem, contraposition, and disjunction in the antecedent are in general not valid for them.

In what follows, we will be especially interested in minimal theories of a disjunctive consequence relation. The following result provides a syntactic description of such theories.

Lemma 8. *A set u of propositions is a minimal theory of a supraclassical Scott consequence relation if and only if $u = \{A \mid \vdash \overline{u}, A\}$.*

This description will be used in what follows.

Weak provability and relativization

As we already mentioned, the main formal difference between disjunctive inference and ordinary Tarski consequence relations amounts to the fact that theories of a disjunctive consequence relation are no longer closed with respect to intersections. As a consequence, we no longer have that every consistent set of propositions always has a logical closure - a single least theory containing it. What we have instead is that any such set is included, in general, in a number of minimal theories (cf. Theorem 2). As a special case, instead of a single least theory of a Tarski consequence relation (that accumulates the set of provable propositions), we have multiple minimal theories of a disjunctive consequence relation. This means that the very concept of provability of propositions should be appropriately generalized, or relaxed, to the disjunctive case.

The following notion of a weak provability provides a starting point for the required generalization.

Definition. Proposition A is *weakly derivable* from a set a of propositions (notation $a \mathrel{|\!\approx} A$) in a Scott consequence relation $\vdash$ if A belongs to some minimal theory of $\vdash$ that includes a.

The following lemma gives a syntactic description of this notion of weak provability.

Lemma 9. *$a \mathrel{|\!\approx} A$ iff $a \vdash A, b$, for some finite set b such that $a \nvdash b$.*

It should be noted that the above notion of a weak derivability does not have all the properties of ordinary provability. To begin with, this notion of derivation is *credulous*, so $a \mathrel{|\!\approx} A$ and $a \mathrel{|\!\approx} B$ do not imply $a \mathrel{|\!\approx} A \wedge B$. This happens because A and B may well belong to different minimal theories containing a. Thus, care should be exercised in an

attempt to extend this notion to derivability of a set of propositions:

Definition. A set v of propositions is *weakly derivable* from a set u in a Scott consequence relation $\vdash$ if v is included in some minimal theory of $\vdash$ that contains u.

The next lemma gives a corresponding syntactic description of this generalization.

Lemma 10. *v is weakly derivable from u iff there exists a set w such that $u \nvdash w$ and $u \vdash A, w$, for any $A \in v$.*

Note that the set w in the above description cannot be restricted in general to a finite set.

As an illustration, the following result provides an alternative description of minimal theories in terms of weak derivability.

Lemma 11. *A set u of propositions is a minimal theory of $\vdash$ if and only if it is a maximal weakly derivable set (from $\emptyset$).*

The above description can be easily generalized to minimal theories containing a given set of propositions.

Corollary 12. *A set u is a minimal theory of $\vdash$ containing a given set v of propositions if and only if u is a maximal set that is weakly derivable from v.*

By the above descriptions, weak derivability amounts to ordinary derivability under a supposition that some propositions *do not* hold. A more systematic description of this idea can be obtained by using the notion of a relativization, or restriction, of a consequence relation with respect to a given set of propositions.

Given a Scott consequence relation $\vdash$ and a set u of propositions, we will define the following consequence relation $\vdash^u$:

$$a \vdash^u b \;\equiv\; a \cap b \neq \emptyset \text{ or } a \cap u \vdash b, \overline{u}.$$

It can be easily verified that $\vdash^u$ is also a Scott consequence relation. It will be called a *restriction*, or *relativization*, of $\vdash$ with respect to u.

The effect of relativization of consequence relations amounts to restricting the inferences to propositions from u. Thus, the following lemma shows that only the latter propositions matter in determining theories of $\vdash^u$.

Lemma 13. *A set v of propositions is a theory of $\vdash^u$ if and only if $v \cap u$ is a theory of $\vdash$.*

It should be noted that the restricted consequence relation is not a sub-relation of the source consequence relation. In fact, the two consequence relations are in general incomparable with respect to set inclusion. The relativization can be viewed, however, as a 'negative' conditionalization based on non-acceptance of some propositions.

The above relativization can be extended to sequent theories. Thus, the *restriction (or relativization) of a sequent theory* Δ with respect to a set u of propositions will be defined as the following sequent theory Δ^u:

$$\Delta^u = \{a \vdash b \cap u \mid a \vdash b \in \Delta \;\&\; a \subseteq u\}.$$

The above relativization of a sequent theory amounts to restricting its rules to propositions from u. Note, in particular, that u is a theory of Δ if and only if Δ^u does not contain rules of the form $a \vdash$.

The following lemma shows that the relativization of sequent theories agrees with our earlier definition of relativization for consequence relations.

Lemma 14. *$\vdash_{\Delta^u}$ coincides with the restriction of $\vdash_\Delta$ with respect to u.*

In view of the above lemma, Δ^u provides an adequate constructive description of relativization. In particular, we immediately obtain that a set v is a theory of Δ^u if and only if $u \cap v$ is a theory of Δ.

Using the above notion of relativization, we can give a more illuminating description of minimal theories of a consequence relation. Indeed, if u is a minimal theory of $\vdash$, it will be a least theory of $\vdash^u$ (see Lemma 13). Consequently, we immediately obtain

Corollary 15. *A theory u of a Scott consequence relation $\vdash$ is minimal if and only if $\vdash^u A$, for any $A \in u$.*

On this description, a minimal theory is a set of propositions that are provable upon the assumption that no proposition outside this set holds.

Default Logic Generalized

Now we are going to describe a generalization of default logic grounded on a theory of disjunctive inference described in the preceding section.

As a matter of fact, an appropriate generalization of default logic based on disjunctive inference rules has already been proposed in (Gelfond *et al.* 1991), guided by the need to provide a logical basis for disjunctive logic programming, as well as more perspicuous ways of handling disjunctive epistemic information[2].

A disjunctive default theory is a set of *disjunctive defaults*, rules of the form

$$a : b/c,$$

where a, b, c are finite sets of propositions. An informal meaning of such rules is 'If all propositions from a are believed, and each $B \in b$ can be consistently assumed, then at least one proposition from c should be believed'.

For a set s of propositions, let $\mathcal{D}(s)$ denote the set of all minimal deductively closed theories that are closed also with respect to the following ordinary disjunctive rules:

$$\{a \vdash c \mid a : b/c \in D \;\&\; \neg B \notin s, \text{ for any } B \in b\}.$$

Then s is said to be an *extension* of a disjunctive default theory if $s \in \mathcal{D}(s)$.

In accordance with the above description, an extension of a disjunctive default theory is a deductively closed set u which happens to be a minimal theory of the set of disjunctive inference rules obtained by taking u itself as an assumption context. It should be clear that when all the rules of the default theory are actually singular, Tarski rules that involve single conclusions in their heads, then the above definition reduces to Reiter's definition of extensions. In fact, many of the properties of Reiter's extensions remain valid also in the disjunctive case (see below).

[2] such as Poole's 'broken hand' counterexample discussed in that paper.

Just as the original notion of an extension, also the above construction can be viewed as a particular instantiation of a bi-context reasoning in which the assumption context imposes constraints on the use of inference rules in the main context. In this respect, the switch to disjunctive default rules provides enhanced expressive capabilities with respect to the main context. Still, even disjunctive default rules give us only limited ways of expressing the assumption information. Namely, in writing default rules we can constrain the assumption context only negatively by saying what does not belong to it (what is consistent to assume), but we cannot directly state that some proposition *should be* assumed. Fortunately, this last expressive shortcoming can also be fixed by a further elaboration of the syntactic form of default rules.

Again, an appropriate final generalization of interest for our study has already been suggested in (Lin and Shoham 1992; Lifschitz 1994) in an attempt to construct a unified formalism for nonmonotonic reasoning and logic programming. Both of these latter formal systems were formulated in a modal framework with two modal operators. It has been shown in (Bochman 1995), however, that this formalism can also be expressed in a purely non-modal setting by using rules of the form $a : b/c : d$, where a, b, c, d are sets of classical propositions. Such generalized default rules could be read as follows:

If all propositions from a are believed, and each proposition from b can be consistently assumed, then at least one proposition from c should be believed, or else at least one proposition from d can be consistently assumed.

Thus, compared with the preceding generalization to disjunctive default rules, the new default rules are disjunctive rules that may involve justifications not only in bodies, but also in their heads. It should also be clear that such default rules provide full expressivity with respect to both the main and assumption contexts.

Below, for a set u of propositions, $\neg u$ will denote the set $\{\neg A \mid A \in u\}$. Then, in full analogy with the preceding constructions, the corresponding generalized default logic can be described as follows.

Definition. A generalized default theory is a set of rules

$$a : b/c : d,$$

where a, b, c, d are sets of classical propositions.

For a set s of propositions, let $\mathcal{D}(s)$ denote the set of minimal deductively closed theories that are closed also with respect to the rules

$$\{a \vdash c \mid a : b/c : d \in D \,\&\, \neg b \cap s = \emptyset \,\&\, \neg d \subseteq s.\}$$

Then s is an *extension* of a generalized default theory if $s \in \mathcal{D}(s)$.

In what follows, we will use a more explicit description of extensions for generalized default theories provided by the next lemma.

Lemma 16. *A set s of propositions is an extension of a generalized default theory D if and only if it satisfies the following conditions:*

- *s is deductively closed;*
- *s is closed with respect to the rules of D: for any rule $a : b/c : d$ from D, if $a \subseteq s$ and $\neg b \cap s = \emptyset$, then $c \cap s \neq \emptyset$, or $\neg d \nsubseteq s$;*
- *There is no smaller deductively closed set $u \subset s$ that is closed with respect to the set of rules*

$$\{a \vdash c \mid a : b/c : d \in D \,\&\, \neg b \cap s = \emptyset \,\&\, \neg d \subseteq s.\} \quad (*)$$

The second condition states that s itself is closed with respect to the above rules, while the third condition secures that s is a minimal such set. Thus, the above description is obviously equivalent to the 'official' definition of extensions.

It turns out that generalized default logic still supports a suitable generalization of the fact established in (Reiter 1980, Theorem 2.5) that extensions are uniquely determined by conclusions of their generating rules.

Given an extension s, let us define the set D_s of *generating rules* of s as follows:

$$D_s = \{a : b/c : d \in D \mid a \subseteq s \,\&\, \neg b \cap s = \emptyset \,\&\, \neg d \subseteq s\}$$

By the above description, generating rules of an extension are default rules that are 'active' with respect to the extension, namely rules in which the justifications agree with the extension, and the premises are satisfied. Then the next lemma states that the extension is a logical closure of some of the conclusions of such rules.

Lemma 17. *If D_s is the set of generating rules of an extension s, then $s = \mathrm{Th}(s_D)$, where*

$$s_D = \{C \in s \mid a : b/c : d \in D_s \,\&\, C \in c\}.$$

Generalized default logic is of course a much more expressive formalism than the original default logic. Thus, in addition to the already mentioned advantages of disjunctive default rules, it has been shown, in effect, in (Lin and Shoham 1992) that an autoepistemic logic is directly representable in this formalism by using rules with justifications in heads. In addition, it has been suggested first in (Lifschitz and Woo 1992) that generalized rules of this kind might be useful also in logic programming. And indeed, it has been shown in (Inoue and Sakama 1998) that program rules of the form

$$A, \mathbf{not}\ A \leftarrow$$

provide a faithful description of abducibles, so they can be used for a formal representation of abductive logic programming. Finally, the causal calculus of McCain-Turner-Lifschitz (see (McCain and Turner 1997a; Bochman 2004)) is also subsumed by this formalism. More precisely, the causal rules $A \Rightarrow B$ of causal theories are representable as default rules $: A/B$ (cf. (McCain and Turner 1997b)) of default theories augmented with an additional axiom

$$: A, \neg A/\mathbf{f}.$$

The latter axiom restrict the extensions to worlds. Moreover, the entire formalism of default theories under this restriction turns out to be equivalent to Turner's logic of universal causation (UCL) from (Turner 1999).

Summing up all these developments, we can confidently claim today that generalized default logic constitutes an important stage in constructing a general, uniform framework for representing nonmonotonic reasoning.

Simple Default Theories

In accordance with the preceding discussion, the logical basis of default logic is formed by (disjunctive) inference rules $a \vdash b$ that are determined in each case by the appropriate assumption context. If we take this logical basis for granted, then a simplest 'nonmonotonic overhead' would consist of most simple default rules of the form $:A/A$, often called *supernormal defaults*. This naturally leads us to the following notion of a simple default theory:

Definition. A generalized default theory will be called *simple* if it includes only rules of the following two kinds:

- Monotonic rules $a : /c :$, and
- Supernormal defaults $: A/A :$.

To simplify the notation, the monotonic rules $a:/c$: having no justifications neither in bodies nor in heads will be written below as ordinary inference rules a/c. Furthermore, a supernormal default $:A/A$ states, in effect, that the proposition A can be used as a default assumption in the corresponding monotonic derivation. As a result, we can get rid of completely from the traditional notation for default rules and identify a simple default theory as a pair $(D, \mathcal{A})$, where D is a set of monotonic rules (a sequent theory), and $\mathcal{A}$ a distinguished set of propositions called *assumptions*. Then the following lemma gives a straightforward reformulation of the definition of extensions for such default theories.

Lemma 18. *A set s of propositions is an extension of a simple default theory $(D, \mathcal{A})$ if and only if it satisfies the following conditions:*

- *s is a minimal deductively closed theory of D that contains $s \cap \mathcal{A}$;*
- *s decides the assumption set: for any $A \in \mathcal{A}$, either $A \in s$, or $\neg A \in s$;*

The first condition in the above description combines two requirements: (i) u is a theory of D, and (ii) there is no smaller theory of D that includes the same assumptions as s. This description implies that extensions of simple default theories are determined in some (weak) sense by their corresponding sets of assumptions. More precisely, by Lemma 11, we have

Corollary 19. *Any extension s of a simple default theory $(D, \mathcal{A})$ is a maximal set that is weakly derivable from $s \cap \mathcal{A}$.*

Not every assumption set, however, generates an extension in this sense. The following definition gives a necessary and sufficient condition.

Definition. A set $\mathcal{A}_0$ of assumptions will be called *stable* if it is consistent and weakly derives $\neg(\mathcal{A} \setminus \mathcal{A}_0)$.

Then the following result shows that extensions are precisely theories that are generated by stable sets of assumptions.

Lemma 20. *A set $\mathcal{A}_0$ of assumptions is stable if and only if $\mathcal{A}_0 = u \cap \mathcal{A}$, for some extension u.*

As a partial converse of the above result, an extension can be characterized as a minimal theory s containing a stable set of assumptions and such that it includes $\neg A$, for any assumption not in s. It should be noted, however, that in the general disjunctive case extensions are not determined uniquely by their sets of assumptions. Indeed, even in the extreme case when the set of assumptions $\mathcal{A}$ is empty, a default theory may have multiple extensions (= minimal theories).

Of course, the above descriptions can be greatly simplified for the non-disjunctive case of Tarski inference rules. As we will see later, the original Reiter's default theories will be reducible to this case.

Let us define a *simple Tarski default theory* as a simple default theory $(D, \mathcal{A})$ such that all inference rules in D are Tarski rules of the form $a \vdash A$. For a set u of propositions, we will denote by $\mathrm{Cn}_D(u)$ the logical closure of u with respect to D, namely the set of propositions that are provable from u using the rules from D and the classical entailment. Then we have

Corollary 21. *If $(D, \mathcal{A})$ is a simple Tarski default theory, then*

- *A set $\mathcal{A}_0$ of assumptions is stable if and only if it is consistent and refutes any assumption not in $\mathcal{A}_0$:*

$$\neg A \in \mathrm{Cn}_D(\mathcal{A}_0), \text{ for any } A \in \mathcal{A} \setminus \mathcal{A}_0.$$

- *A set s of propositions is an extension if and only if $s = \mathrm{Cn}_D(\mathcal{A}_0)$, for some stable set of assumptions $\mathcal{A}_0$.*

Simple default theories provide a most transparent and natural description of default reasoning. There is an obvious correspondence between this framework and two other general approaches to nonmonotonic reasoning, namely Poole's abductive theory (Poole 1988) and the assumption-based framework of (Bondarenko *et al.* 1997). In Poole's Theorist system a default theory is also described as a pair $(T, \mathcal{A})$, where T is a classical theory (a set of propositions), and $\mathcal{A}$ a set of assumptions. In this framework, an extension is defined as a logical closure of a maximal consistent set of assumptions. It should be clear that Poole's theories correspond precisely to simple default theories that contain only rules of the form $\vdash A$. Moreover, even the full version of this system, namely theories with constraints, is subsumed (for a finite set of constraints) by simple Tarski default theories that may contain also constraints, that is, rules of the form $a \vdash \mathbf{f}$. Already this latter modification creates, however, a difference between extensions of default logic and Poole's extensions. The difference stems, ultimately, from a logical fact that, for supraclassical consequence relations, the inconsistency $a, A \vdash \mathbf{f}$ does not necessarily imply refutation $a \vdash \neg A$. As a result, not every maximal consistent set of assumptions will be stable, so not every Poole's extension will be an extension in our sense. In fact, the stability condition on the set of assumptions can be seen as a hallmark of default logic as opposed to simpler systems of nonmonotonic reasoning. According to this condition, a proper set of assumptions not only should be a maximal consistent set, it should also *explain why* other assumptions should not be accepted (by refuting them).

A more general account of default reasoning has been given in (Bondarenko *et al.* 1997). Bringing it closer to our present terminology, and without loss of generality, a

default theory in this framework can also be defined as a simple Tarski default theory $(D, \mathcal{A})$, plus a mapping that assigns every assumption A its *contrary* $\overline{A}$. Then a set $\mathcal{A}_0$ of assumptions is said to *attack* an assumption A if it implies its contrary: $\overline{A} \in \mathrm{Cn}_D(\mathcal{A}_0)$. Also, a set $\mathcal{A}_0$ of assumptions is *closed*, if $A \in \mathcal{A}_0$, for any assumption $A \in \mathrm{Cn}_D(\mathcal{A}_0)$. Finally, a set of assumptions is *stable*, if it is closed, does not attack itself, but attacks each assumption outside the set.

As can be seen from the above description, for the case of Tarski rules, our constructions are a special case of the above framework when the contrary $\overline{A}$ of an assumption A is just $\neg A$. Still, it will be shown in the next section that this simplification is nevertheless sufficient for covering the full generalized default logic. In other words, it will be shown that default logic can be described entirely in the standard language of rule-based classical logical reasoning.

The reduction

We are going to describe now a translation of arbitrary default theories to simple ones. To begin with, we will extend the source propositional language $\mathcal{L}$ with new propositional atoms A°, for any classical proposition A in $\mathcal{L}$. For a set u of propositions from $\mathcal{L}$, u° will denote the set of new atoms $\{A^\circ \mid A \in u\}$.

Next, if D is a default theory in $\mathcal{L}$, then D° will denote the following set of plain disjunctive rules in the extended language:

$$\{a, b^\circ / c, d^\circ \mid a : b / c : d \in D\} \tag{1}$$

plus the following two rules for any formula A from $\mathcal{L}$ that appears as a justification in the rules from D:

$$\neg A / \neg A^\circ \qquad \text{and} \qquad : A^\circ / A^\circ \tag{2}$$

Clearly, D° is a simple default theory. Also, it can be easily seen that the above translation is polynomial and modular. Moreover, the following theorem shows that this translation is also faithful, so it is actually a PFM translation in the sense of (Janhunen 1999).

Theorem 22. *A set u is an extension of D if and only if there is a unique extension u_0 of D° such that $u = u_0 \cap \mathcal{L}$.*

As a first consequence of the above reduction, we obtain that justifications in heads of generalized default rules can be eliminated, so we have

Corollary 23. *Any generalized default theory is PFM-reducible to a disjunctive default theory.*

Moreover, if the source default theory is actually a Reiter's default theory that contains only non-disjunctive rules, then the above reduction produces a simple Tarski default theory. Hence we have

Corollary 24. *Any Reiter's default theory (W, D) is PFM-reducible to a simple Tarski default theory.*

It should also be noted that the simple default theories obtained by the above reduction have many useful special properties that do not hold for simple theories in general. Thus, the assumption set of such theories is a set of atoms. Moreover, the language of assumptions is completely disjoint from the 'main' language in the sense that assumption atoms do not occur in other formulas of the language. Hopefully, these additional properties might be exploited in more elaborated results for default logic, or even in devising more efficient computational algorithms.

As many other essential results in nonmonotonic reasoning, the above theorem is also not completely new and has numerous precedents.

- From a technical side, the proof of the above result generalizes the proof of the corresponding result for disjunctive logic programs with negations in heads given in (Janhunen 2001).
- The first expression of this kind of reduction can be discerned from the representation of Reiter's default logic in terms of argument systems suggested in (Lin and Shoham 1989). In this representation, default rules $A{:}B_1, \ldots, B_n / C$ were represented as monotonic rules

  $$A, \neg ab(B_1), \ldots, \neg ab(B_n) / C.$$

  In addition, an argument system was required to contain monotonic rules of the form $\neg A / ab(A)$, plus *nonmonotonic* rules $\mathbf{t} \Rightarrow \neg ab(B)$, the latter rules having the same functionality as supernormal defaults

  $$: \neg ab(B) / \neg ab(B).$$

  As can be seen, for the case of Reiter's default logic our translation is just a notational variant of this representation.
- Our translation is also quite similar to the representation of default logic in the assumption-based framework, described in (Bondarenko *et al.* 1997). On this representation, default rules $A{:}B_1, \ldots, B_n / C$ were translated to monotonic rules $A, \mathbf{M}B_1, \ldots, \mathbf{M}B_n / C$, where propositions of the form $\mathbf{M}B$ were taken to be the assumptions, and $\neg B$ was considered a contrary to the assumption $\mathbf{M}B$. In some sense, however, our reduction complements these representation results by showing that default logic itself constitutes a primary logical framework for nonmonotonic reasoning (at least if we adhere to the stable semantics of extensions).

Conclusions

It goes without saying that the ultimate aim of nonmonotonic formalisms consists in providing computational tools for solving the actual problems arising in Artificial Intelligence. Nevertheless, as in many other areas of scientific research, it should also be clear that our formalisms will have a chance to fulfil this aim only to the extent they will manage to provide an adequate, concise and versatile framework for *representing* such problems. In this sense, default logic can primarily be viewed as a framework for representing *defeasible knowledge*. The accumulated body of results on this subject indicates that default logic constitutes, ultimately, a proper unified framework for this task.

It is also well known that default logic is a computationally difficult formalism. It seems, however, that (as in any other honest business) this happens because we have to pay

for the ability to handle and act on the basis of additional, defeasible knowledge that is not available in ordinary, monotonic logical reasoning. Moreover, the representation problems of nonmonotonic reasoning should to some reasonable extent be separated from the computability questions. In other words, we should know *what* ought to be expressed, or represented, even before we know *how* it could be computed. There should be no difference in attitude here between the theory of nonmonotonic reasoning and Logic in general.

Finally, it should also be noted that default logic, even the generalized one described in this study, still does not resolve by itself the actual problems of AI, just as the differential calculus does not resolve by itself the problems of physics. In this sense default logic is only a framework for dealing with nonmonotonicity, in which real and useful kinds of nonmonotonic reasoning can hopefully be expressed and studied. Despite obvious successes, much work still should be done for a rigorous and adequate representation of the AI universum in all its actual scope and diversity.

Acknowledgment. Thanks to the referees for their useful comments and suggestions (including the new title of the paper).

References

A. Bochman. On bimodal nonmonotonic logics and their unimodal and non-modal equivalents. In *Proc. IJCAI'95*, pages 1518–1524, 1995.

A. Bochman. *A Logical Theory of Nonomonotonic Inference and Belief Change*. Springer, 2001.

A. Bochman. A causal approach to nonmonotonic reasoning. *Artificial Intelligence*, 160:105–143, 2004.

A. Bochman. *Explanatory Nonmonotonic Reasoning*. World Scientific, 2005.

A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic framework for default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

D. M. Gabbay. *Investigations in Modal and Tense Logics*. D. Reidel, 1976.

D. M. Gabbay. *Semantical Investigations in Heyting's Intuitionistic Logic*. D. Reidel, 1981.

M. Gelfond, V. Lifschitz, H. Przymusińska, and M. Truszczyński. Disjunctive defaults. In *Proc. Second Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'91*, pages 230–237, Cambridge, Mass., 1991.

K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.

T. Janhunen. On the intertranslatability of non-monotonic logics. *Annals of Math. and Art. Intel.*, 27:791–828, 1999.

T. Janhunen. On the effect of default negation on the expressiveness of disjunctive rules. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning, LPNMR 2001*, volume 2173 of *LNAI*, pages 93–106. Springer, 2001.

V. Lifschitz and T. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In *Proc. Third Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'92*, pages 603–614. Morgan Kauffman, 1992.

V. Lifschitz. Minimal belief and negation as failure. *Artificial Intelligence*, 70:53–72, 1994.

F. Lin and Y. Shoham. Argument systems: A uniform basis for nonmonotonic reasoning. In *Proceedings of 1st Intl. Conference on Principles of Knowledge Representation and Reasoning*, pages 245–255, Stanford, CA, 1989.

F. Lin and Y. Shoham. A logic of knowledge and justified assumptions. *Artificial Intelligence*, 57:271–289, 1992.

W. Marek and M. Truszczyński. Relating autoepistemic and default logics. In *Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'89*, pages 276–288, San Mateo, Calif., 1989. Morgan Kaufmann.

W. Marek and M. Truszczyński. *Nonmonotonic Logic, Context-Dependent Reasoning*. Springer, 1993.

W. Marek, A. Nerode, and J. Remmel. A theory of nonmonotonic rule systems. *Annals of Mathematics and Artificial Intelligence*, 1:241–273, 1990.

N. McCain and H. Turner. Causal theories of action and change. In *Proceedings AAAI-97*, pages 460–465, 1997.

N. McCain and H. Turner. On relating causal theories to other formalisms. Unpublished ms., 1997.

D. McDermott and J. Doyle. Nonmonotonic logic. *Artificial Intelligence*, 13:41–72, 1980.

D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

D. Scott. Completeness and axiomatizability in many-valued logic. In *Proc. Symp. In Pure Math., No. 25*, pages 431–435, 1974.

K. Segerberg. *Classical Propositional Operators*. Clarendon Press, 1982.

D. J. Shoesmith and T. J. Smiley. *Multiple-Conclusion Logic*. Cambridge University Press, 1978.

H. Turner. A logic of universal causation. *Artificial Intelligence*, 113:87–123, 1999.

R. Wojcicki. *Theory of Logical Calculi*, volume 199 of *Synthese Library*. Kluwer Ac. Publ., 1988.

Specificity Principle in Querying Databases with Preferences

Souhila Kaci
Université Lille-Nord de France, Artois
CRIL, CNRS UMR 8188 - IUT de Lens
F-62307, France
kaci@cril.univ-artois.fr

Rui da Silva Neves
DSVP
5 allées A. Machado, 31058
Toulouse Cedex 0, France
neves@univ-tlse2.fr

abstract

Everybody is aware that preferences are present in many domains of our daily life. Databases community has rapidly perceived the importance of preferences in queries. As users generally express comparative preference statements of the form "I prefer p to q", existing works that integrate preferences in database queries are based on this preference format with however different semantics. We propose in this paper a simple and unified framework to reason about preference queries based on comparative preference statements where different semantics may cohabit. We show that it recovers existing works when only one semantics is considered. Our framework is based on specificity principle which makes it highly attractive from computational point of view.

Introduction

Preferences are present in many domains in our daily life. As such, many works in psychology, since the pioneering work of Kahneman and Tversky have focused on the study of preferences in human decision making and judgment. It has been shown for example that preferences comparisons are intransitive (Tversky 1969; Sjoberg 1975) and a very robust preference reversal phenomena (e.g., (Lichtenstein and Slovic 1971; Lindman 1971)). Such a phenomena has been observed for example in choices regarding monetary outcomes, both hypothetical and real, and in questions pertaining to the loss of human lives. These findings led to a number of empirical investigations, some of which attempted to eliminate preference reversals but instead replicated and extended them.

Databases community has rapidly perceived the importance of preferences in queries. Suppose that a user is looking for a hotel in Sydney. Due to the large number of hotels there, providing more knowledge about user's preferences in the query, for e.g., hotels that are in the downtown and cheap, will help to eliminate irrelevant hotels and rank-order the relevant ones. Indeed preferences can be viewed as a filter which focusses on relevant answers only and returns them ranked according to their relevance. The integration of preferences in database queries comes however with many complex issues regarding the representation of preferences and reasoning about them. Advances in this topic started with quantitative representations (Chang 1976; Motro 1986) which consist in associating a score to each tuple expressing how much we like it. However the expressive power of quantitative preferences has been proved to be limited (Fishburn 1999). Moreover the question where do the numbers come from represents a serious limitation of these approaches. Indeed different proposals have been made to introduce qualitative preferences in database queries. The pionner work was proposed by Börzsönyi et al. (2001) who make use of skyline operation. Following skyline, tuples are compared on the basis of several criteria. A tuple dominates another tuple if it is as good or better w.r.t. all criteria and better w.r.t. at least one criteria. Then the skyline is defined as the tuples which are not dominated by any other tuple. However skyline queries are too limited in their expressiveness and supporting user's preferences requires more sophisticated preference models in database queries. Kiessling (2002) promoted this idea and proposed a preference model in which preferences are represented by a strict partial order. The result of a preference query is then retrieved using the BMO ("Best Matches Only") query model which returns all tuples that are the closest to user's preferences. However the elicitation of an explicit partial order over tuples is not an easy task. Indeed Chomicki (2002; 2003) also used the notion of strict partial orders as preference models in database queries but preferences are defined in a concise way as a logical formula. Then the partial order can be computed from the logical formula and the result of a preference query is computed using the winnow operator which coincides with BMO query model. In the same line of research, Brafman and Domshlak (2004) represent the partial order by a CP-net which is a graphical representation of preferences (Boutilier et al. 1999). Although the two approaches differ on the way to reason about preferences, they agree that preferences are generally expressed in terms of comparative statements, for example "I prefer fish to meat", "I prefer cake to ice_cream", "If fish is served then I prefer white wine to red wine", etc. However these preferences expressed in natural language may have different interpretations. Following Chomicki (2003) when I prefer fish to meat, this means that any menu composed of fish is preferred to any menu composed of meat, regardless what is served with fish or meat (for example wine and dessert). This se-

mantics, rather strong when different preference statements are expressed, has been criticized by Brafman and Domshlak (2004) who proposed a weaker semantics based on ceteris paribus principle. Following this principle, the comparative preference statement "I prefer fish to meat" is interpreted as a menu composed of fish is preferred to a menu composed of meat when the two menus are otherwise exactly the same. While each approach argues for a unique semantics and deals with, we believe that both semantics are equally good and may be present together.

Moreover representing and reasoning about comparative preference statements has been widely investigated in Artificial Intelligence (Boutilier et al. 1999; Benferhat, Dubois, and Prade 2001; Benferhat and Kaci 2001; Wilson 2004; Boutilier et al. 2004; Kaci and van der Torre 2008). Unfortunately, these works have not been used in preference queries from databases if we except (Brafman and Domshlak 2004).

The aim of this paper is twofolds: (i) we present a framework for preference queries based on comparative preference statements where two semantics cohabit, (ii) we also show that the use of specificity principle developed in non-monotonic reasoning (Yager 1983) in ranking alternatives w.r.t. preference queries has an attractive complexity.

After giving a necessary background on each approach, we present a unified framework where both strong and ceteris paribus semantics can cohabit. Our framework is based on specificity principle and runs in a polynomial time complexity. We then show how our framework captures Chomicki's and Brafman&Domshlak's approaches. Lastly we conclude.

Compact qualitative preference models for querying a database

Basic definitions

A relational scheme is denoted $\mathcal{R}(X)$, where $\mathcal{R}$ is the name of the relation and $X = A_1, \cdots, A_l$ is a set of l attributes. Each attribute A_i takes its values in a domain D_i denoted $Dom(A_i)$. A tuple t over a scheme $\mathcal{R}(X)$ associates with each attribute A_i in X a value taken from D_i. The notation $t.A_i$ refers to the value given to the attribute A_i in the tuple t. An instance r over $\mathcal{R}$ is a non-empty finite set of tuples over $\mathcal{R}(X)$.

Preferences over $\mathcal{R}(X)$ are expressed in qualitative way by specifying that a tuple t_1 is preferred to a tuple t_2 (Chomicki 2003; Kiessling 2002). A qualitative preference relation (preference relation for short) $\succeq$ is a preorder i.e., a reflexive and transitive binary relation over $D_1 \times \cdots \times D_l$ such that $t_1 \succeq t_2$ stands for "t_1 is at least as preferred as t_2". $t_1 = t_2$ means that t_1 and t_2 are equally preferred (i.e., both $t_1 \succeq t_2$ and $t_2 \succeq t_1$ hold), and $t_1 \sim t_2$ means that t_1 and t_2 are incomparable (i.e., neither $t_1 \succeq t_2$ nor $t_2 \succeq t_1$ holds).

A strict preference relation $\succ$ is an order i.e., an irreflexive and transitive binary relation over $D_1 \times \cdots \times D_l$ such that $t_1 \succ t_2$ stands for "t_1 is strictly preferred to t_2"[1]. We also write $t_1 \sim t_2$ w.r.t. $\succ$ when neither $t_1 \succ t_2$ nor $t_2 \succ t_1$ holds. Given a preference relation $\succeq$ on $D_1 \times \cdots \times D_l$, we derive a strict preference relation $\succ$ as follows: $t_1 \succ t_2$ iff $t_1 \succeq t_2$ holds but $t_2 \succeq t_1$ does not.

For simplicity and only when there is no ambiguity, we will omit the term "strict" and refer to both $\succeq$ and $\succ$ as preference relations.

$\succeq$ (resp. $\succ$) is cyclic if and only if there exist t_1 and t_2 such that $t_1 \succ \cdots \succ t_2 \succ t_1$. A preference relation $\succeq$ (resp. $\succ$) is complete if and only if $\forall t_1, t_2 \in D_1 \times \cdots \times D_l$, either $t_1 \succeq t_2$ or $t_2 \succeq t_1$ (resp. $t_1 \succ t_2$ or $t_2 \succ t_1$) holds.

A complete preference relation can be represented by means of a set of ordered (non-empty) equivalence classes $(E_1, \cdots, E_n)$ such that:

- $E_1 \cup \cdots \cup E_n$ is the set of all possible tuples,
- $E_i \cap E_j = \emptyset$ for $i \neq j$.
- Tuples belonging to the same equivalence class are equally preferred w.r.t. the associated preference relation and $\forall t_1, t_2 \in D_1 \times \cdots \times D_l$, with $t_1 \in E_i$ and $t_2 \in E_j$ we have $i < j$ iff $t_1 \succ t_2$.

Complete preference relations can be compared following specificity principle (Yager 1983):

Definition 1 (Minimal/Maximal specificity principle) *Let $\succeq$ and $\succeq'$ be two complete preorders represented by ordered partitions $(E_1, \cdots, E_n)$ and $(E'_1, \cdots, E'_{n'})$ respectively. We say that $\succeq$ is less specific than $\succeq'$, written as $\succeq \sqsubseteq \succeq'$, iff $\forall t \in D_1 \times \cdots \times D_l$, if $t \in E_i$ and $t \in E'_j$ then $i \leq j$. $\succeq$ belongs to the set of the least (resp. most) specific preorders among a set of preorders $\mathcal{O}$ if there is no $\succeq' \in \mathcal{O}$ such that $\succeq' \sqsubset \succeq$, i.e., $\succeq' \sqsubseteq \succeq$ holds but $\succeq \sqsubseteq \succeq'$ (resp. $\succeq \sqsubset \succeq'$) does not.*

Querying a database with logical formulas

Chomicki (2003) represents the preference relation as a strict partial order over tuples in a compact way by means of a first-order logical formula denoted $C(t_1, t_2)$[2] and called *a preference formula*. $C(t_1, t_2)$ defines a preference relation $\succ_C$ over $\mathcal{R}(X)$ in the following way:

$$t_1 \succ_C t_2 \text{ iff } C(t_1, t_2).$$

Chomicki (2003) also proposed two ways to compose preference relations: unidimensional and multidimensional compositions. In the former, preference relations over the same scheme are composed producing a new preference relation over the same scheme. In the latter, preference relations over several schemas are composed producing a new preference relation over the cartesian product of these schemas. For simplicity we will consider in this paper unidimensional composition only. Extension to multidimensional composition is left for a future work. Examples of unidimensional composition are booelan (intersection, union and difference) and prioritized composition.

[1] We also say that t_1 dominates t_2.

[2] Sometimes we write C for short.

Let $\succ_{C_1}$ and $\succ_{C_2}$ be two preference relations associated to the preference formulas C_1 and C_2 respectively. The intersection (resp. union, difference) of $\succ_{C_1}$ and $\succ_{C_2}$, denoted $\succ_C = \succ_{C_1} \cap \succ_{C_2}$ (resp. $\succ_C = \succ_{C_1} \cup \succ_{C_2}$, $\succ_C = \succ_{C_1} - \succ_{C_2}$) is defined by the formula $C = C_1 \wedge C_2$ (resp. $C = C_1 \vee C_2$, $C = C_1 \wedge \neg C_2$). The prioritized composition $\succ_C$ of $\succ_{C_1}$ and $\succ_{C_2}$ giving priority to $\succ_{C_1}$ has the following reading: prefer according to $\succ_{C_2}$ unless $\succ_{C_1}$ is applicable. It is defined as

$$t_1 \succ_C t_2 \text{ iff } (t_1 \succ_{C_1} t_2) \vee ((t_1 \sim_{C_1} t_2) \wedge (t_1 \succ_{C_2} t_2)).$$

Lastly the winnow operator has been defined (Chomicki 2003). It selects the set of undominated tuples given a preference relation $\succ_C$ derived from a preference formula C. Formally the winnow operator, denoted $W_C(r)$, is defined as

$$W_C(r) = \{t_1 | \nexists t_2 \in r, t_2 \succ_C t_1\},$$

where r is an instance of $\mathcal{R}(X)$.
Note that if $W_C(r)$ is empty then $\succ_C$ is necessarily cyclic but the converse is not true as illustrated in Example 1.
Different algorithms have been proposed to compute the result of the winnow operator (Börzsönyi, Kossmann, and Stocker 2001; Chomicki 2003; Torlone and Ciaccia 2002). They all share the same principle. We recall the nested-loop algorithm (Chomicki 2003) which is the simplest algorithm to understand but not the most efficient one. Let r be an instance of a relational scheme $\mathcal{R}$[3].

Algorithm 1: Computation of $W_C(r)$.

Data: $r, \succ_C$
Result: $W_C(r)$
begin
  $W_C(r) = \emptyset$
  1. open a scan S_1 on r
  2. **for** *each tuple* t_1 *in* S_1 **do**
    a. open a scan S_2 on r
    b. **for** *each tuple* t_2 *in* S_2 **do**
      if $t_2 \succ_C t_1$ **then** go to 2.d
    c. $W_C(r) = W_C(r) \cup \{t_1\}$
    d. close S_2
  3. close S_1
  return $W_C(r)$.
end

Example 1 *Let* $Meal(dish, wine)$ *with* $Dom(dish) = \{meat, fish\}$ *and* $Dom(wine) = \{red, white, rosé\}$. *Let* $r = Dom(dish) \times Dom(wine) = \{meat - red, meat - white, meat - rosé, fish - red, fish - white, fish - rosé\}$. *Let* $\succ_{C_1}$ *be defined by a preference formula* C_1 *as follows:* $t \succ_{C_1} t'$ *iff* $((t.dish = meat \wedge t.wine = red \wedge t'.dish = meat \wedge t'.wine = white) \vee (t.dish = fish \wedge t.wine = white \wedge t'.dish = fish \wedge t'.wine = red))$. $\succ_{C_1}$ *is depicted in Figure 1.a. An edge from* t *to* t' *stands for "*t *is preferred to* t'*". We have* $W_{C_1}(r) = \{meat - red, fish - white, meat - rosé, fish - rosé\}$.

[3] a scan is a copy.

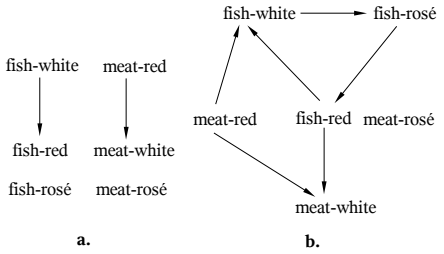


Figure 1: Preference relations $\succ_{C_1}$ and $\succ_{C_2}$.

Let $\succ_{C_2}$ *be a binary preference relation over* $Meal$ *defined by a preference formula* C_2 *as follows:* $t \succ_{C_2} t'$ *iff* $((t.wine = red \wedge t'.wine = white) \vee (t.dish = fish \wedge t.wine = white \wedge t'.dish = fish \wedge t'.wine = rosé) \vee (t.dish = fish \wedge t.wine = rosé \wedge t'.dish = fish \wedge t'.wine = red))$. *The preference relation* $\succ_{C_2}$ *is depicted in Figure 1.b. We have* $W_{C_2}(r) = \{meat - red, meat - rosé\}$. *Note that* $W_{C_2}(r) \neq \emptyset$ *even if* $\succ_{C_2}$ *is cyclic.*

Along this paper, we restrict ourselves to preference formulas which are constructed from atomic formulas using logical connectives $\wedge, \vee, \neg$. An atomic formula has the form $A = u$ or $A \neq u$ where A is an attribute and u is a constant. This restriction is made without any loss of generality. In fact preferences of the form "I prefer book b to book b' when $b.price < b'.price$" can be modelled in our setting by comparing all possible prices.

Since the purpose of this paper is to combine two frameworks, we need to standardize the notations. In Chomicki's approach presented in this section, preference queries are based on a preference formula which is a first order formula C such that t is preferred to t' if and only if $C(t, t')$. We know that any preference formula can be written under the form $C_1 \vee \cdots \vee C_l$ such that each C_i is of the form $c_{i,1} \wedge \cdots \wedge c_{i,m}$, where $c_{i,j}$ is an atomic formula i.e., of the form $A = u$ or $A \neq u$, where A is an attribute and u is a constant.

Lemma 1 *Let* C *be a preference formula such that* $C = C_1 \vee \cdots \vee C_l$. *Then,* $\succ_C = \succ_{C_1} \cup \cdots \cup \succ_{C_l}$.

Let $p > q$ denote a comparative preference statement which expresses a preference for p over q, where p and q are formulas. The notation $t \models p$ means that t makes the formula p true.
Given the formula $C = C_1 \vee \cdots \vee C_l$, we construct a set of comparative preference statements $\mathcal{P}_C = \{(s_1) : p_1 > q_1, \cdots, (s_l) : p_l > q_l\}$ such that (s_i) is the name of the comparative preference statement and p_i (resp. q_i) is the part of C_i concerning the tuple t (resp. t'). More precisely $(s_i) : p_i > q_i$ is built from C_i as follows:

- Recall that C_i is a first order formula concerning two tuples t and t'. Let $S_t \subset \{c_{i,1}, \cdots, c_{i,m}\}$ be the set of atomic formulas in C_i concerning t.

- For each atomic formula $c_{i,j}$ in S_t we define $f(c_{i,j})$ as follows:

$$f(c_{i,j}) = \begin{cases} u_A & \text{if } c_{i,j} \text{ is of the form } A = u \\ \neg u_A & \text{if } c_{i,j} \text{ is of the form } A \neq u \end{cases}$$

  When there is no ambiguity we omit A from u_A and $\neg u_A$.
- Then $p_i = f(c_{i,1}) \wedge \cdots \wedge f(c_{i,m})$.

We define q_i in similar way for t'.

Lemma 2 *Let $C = C_1 \vee \cdots \vee C_l$ be a preference formula and $\{(s_1) : p_1 > q_1, \cdots, (s_l) : p_l > q_l\}$ be the set of comparative preference statements constructed from C. Then, for $i = 1, \cdots, l$,*

$$\forall t, t' \in r, t \succ_{C_i} t' \text{ iff } t \models p_i \text{ and } t' \models q_i.$$

Example 2 *Consider again the preference formula C_1 given in Example 1. We have $t \succ_{C_1} t'$ iff $((t.dish = meat \wedge t.wine = red \wedge t'.dish = meat \wedge t'.wine = white) \vee (t.dish = fish \wedge t.wine = white \wedge t'.dish = fish \wedge t'.wine = red))$. Then $C_1 = C_{11} \vee C_{12}$, where $C_{11} = (t.dish = meat) \wedge (t.wine = red) \wedge (t'.dish = meat) \wedge (t'.wine = white)$ and $C_{12} = (t.dish = fish) \wedge (t.wine = white) \wedge (t'.dish = fish) \wedge (t'.wine = red)$. So $\mathcal{P}_{C_1} = \{(s_1) : meat \wedge red > meat \wedge white, (s_2) : fish \wedge white > fish \wedge red\}$.*
We can check that $t \succ_{C_{11}} t'$ iff $t \models meat \wedge red$ and $t' \models meat \wedge white$. Also $t \succ_{C_{12}} t'$ iff $t \models fish \wedge white$ and $t' \models fish \wedge red$.

The rewritting of a first order formula C as a set of comparative preference statements will make discussions and results of the next sections easier to follow.

Querying a relational database with a CP-net

The standard interpretation of preferences of the form "I prefer p to q" in database community is "*each tuple satisfying p is preferred to each tuple satisfying q*" (Chomicki 2003). This semantics is called totalitarian in (Brafman and Domshlak 2004), strong in (Benferhat and Kaci 2001; Wilson 2004) and careful in (Kaci and van der Torre 2008). In this paper, we use the terminology "totalitarian". However, as noticed in (Benferhat and Kaci 2001; Brafman and Domshlak 2004), such semantics is too strong as the set of results may be empty when several comparative preference statements are dealt with.

Example 3 *(Example 1 continued)*
Let $\succ_{C_3}$ be defined over $Meal$ by a preference formula as follows:
$t \succ_{C_3} t'$ iff $((t.wine = red \wedge t'.wine \neq red) \vee (t.dish = fish \wedge t.wine = white \wedge t'.dish = fish \wedge t'.wine \neq white))$.
We have $\mathcal{P}_{C_3} = \{(s_1) : red > \neg red, (s_2) : fish \wedge white > fish \wedge \neg white\}$. Then $\succ_{C_3} = \succ_{s_1} \cup \succ_{s_2}$. Following (s_1) we have $meat - red \succ_{s_1} meat - white$, $meat - red \succ_{s_1} meat - rosé$, $meat - red \succ_{s_1} fish - white$, $meat - red \succ_{s_1} fish - rosé$, $fish - red \succ_{s_1} meat - white$, $fish - red \succ_{s_1} meat - rosé$, $fish - red \succ_{s_1} fish - white$ and $fish - red \succ_{s_1} fish - rosé$. Following (s_2) we have $fish - white \succ_{s_2} fish - red$ and $fish - white \succ_{s_2} fish - rosé$. Since $\succ_{C_3} = \succ_{s_1} \cup \succ_{s_2}$ we have therefore both $fish - red \succ_{C_3} fish - white$ and $fish - white \succ_{C_3} fish - red$.

In order to overcome the above problem raised by the union composition, the authors of (Brafman and Domshlak 2004) proposed a weaker preference semantics based on ceteris paribus semantics together with CP-nets. This semantics compares two tuples which are completed in the same way besides specified preferences. CP-nets (Boutilier et al. 2004) are graphical representation of preferences based on ceteris paribus semantics. A CP-net $\mathcal{N}$ over $A_1, \cdots, A_n$ is a directed graph in which $A_1, \cdots, A_n$ are nodes. A direct edge from A_i to A_j means that preference over values of A_j is conditioned on the value of A_i. Preferences are locally specified at the level of each node A_i with a conditional preference table $CPT(A_i)$ which associates a complete order on values of A_i for each instantiation of parents of A_i. A partial strict order, denoted $\succ_{\mathcal{N}}$, is associated to $\mathcal{N}$ resulting from the application of ceteris paribus semantics on local preferences.

Example 4 *Let $\mathcal{N}$ be a CP-net over* dish *and* wine *depicted in Figure 2.a, with $Dom(dish) = \{fish, meat\}$ and $Dom(wine) = \{white, red, rosé\}$. $\mathcal{N}$ induces the partial*

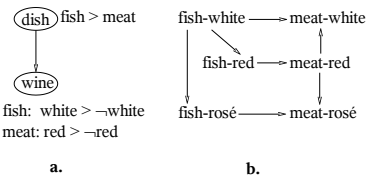


Figure 2: A CP-net and its associated order.

order (depicted in Figure 2.b):

- $fish-white \succ meat-white$, $fish-red \succ meat-red$ *and* $fish - rosé \succ meat - rosé$ *(due to $fish > meat$)*
- $fish-white \succ fish-red$, $fish-white \succ fish-rosé$ *(due to $fish : white > \neg white$)*
- $meat-red \succ meat-white$, $meat-red \succ meat-rosé$ *(due to $meat : red > \neg red$)*

Note however that the acyclicity of preference relations induced by an acyclic CP-net is due to the hierarchical structure of preferences i.e., preferences over values of a child node depend on the values of its parent nodes. The use of ceteris paribus semantics without being together with CP-nets does not guarantee the acyclicity of the preference relation.

Example 5 *(Example 3 continued)*
Following ceteris paribus semantics, (s_1) leads to $fish - red \succ_{s_1} fish - white$, $fish - red \succ_{s_1} fish - rosé$, $meat-red \succ_{s_1} meat-white$ and $meat-red \succ_{s_1} meat-rosé$. (s_2) leads to $fish - white \succ_{s_2} fish - red$ and $fish - white \succ_{s_2} fish - rosé$. Therefore we have both $fish-red \succ fish-white$ and $fish-white \succ fish-red$.

While CP-nets approach guarantees acyclic preference relations, the computation of the winnow operator is however

expensive from computational point of view. This is because the winnow operator ($W_{\mathcal{N}}(r) = \{t_1 | \nexists t_2 \in r, t_2 \succ_{\mathcal{N}} t_1\}$) is based on pairwise comparisons, called dominance queries, between tuples which are NP-hard in CP-nets (Boutilier et al. 2004). In order to overcome this computational limitation of CP-nets, the authors of (Brafman and Domshlak 2004) have weakened dominance queries into ordering queries. The idea consists in replacing the partial order $\succ_{\mathcal{N}}$ associated to $\mathcal{N}$ by a complete order $\succ_t$ such that $\forall t_1, t_2 \in D_1 \times \cdots \times D_l$, if $t_1 \succ_{\mathcal{N}} t_2$ then $t_1 \succ_t t_2$.

Our framework

As discussed in the previous sections, two main preference semantics of comparative preference statements have been defended in databases framework: totalitarian and ceteris paribus semantics. However existing preference representations in database queries use *one semantics only at time* which makes preference queries very restrictive in their expressiveness.

Example 6 *Let $\mathcal{R}(dish, wine, dessert)$ be a relational scheme modeling menus such that $Dom(dish) = \{fish, meat\}$, $Dom(wine) = \{white, red\}$ and $Dom(dessert) = \{cake, ice\_cream\}$. Let $r = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ be an instance of $\mathcal{R}$ with $t_0 : fish - white - ice\_cream$, $t_1 : fish - white - cake$, $t_2 : fish - red - ice\_cream$, $t_3 : fish - red - cake$, $t_4 : meat - white - ice\_cream$, $t_5 : meat - white - cake$, $t_6 : meat - red - ice\_cream$, $t_7 : meat - red - cake$. Clara expresses three comparative preference statements "$(s_1) : fish > meat$", "$(s_2) : red \wedge cake > white \wedge ice\_cream$" and "$(s_3) : fish \wedge white > fish \wedge red$". Moreover we know that Clara is vegetarian so s_1 should be interpreted following totalitarian semantics i.e., any menu composed of fish is preferred to any menu composed of meat. Let $\succ$ be the partial order associated to $\{(s_1), (s_2), (s_3)\}$. Using totalitarian semantics on (s_1), (s_2) and (s_3) is too strong and leads to cyclic preferences since we have both $t_0 \succ t_3$ and $t_3 \succ t_0$ for example. Now using ceteris paribus semantics on (s_1), (s_2) and (s_3) leads to the partial order depicted in Figure 3.a. However this semantics makes $t_3 : fish - red - cake$ and $t_5 : meat - white - cake$ (also t_0 and t_7; t_2 and t_4) incomparable while we would expect that t_3 is preferred to t_5 since Clara is vegetarian. In order to capture Clara's preferences regarding fish and meat, it is more suitable to use totalitarian semantics with the comparative preference statement (s_1). It is intuitively arguable to apply ceteris paribus semantics to (s_2) and (s_3) since no further information is given about these preferences. Figure 3.b depicts the partial order associated to $(s_1), (s_2)$ and (s_3) with (s_1) obeying totalitarian semantics and (s_2) and (s_3) obeying ceteris paribus semantics. We have $W_C(r) = \{t_1\}$.*

One may treat the comparative preference statement (s_1) and the fact that Clara is vegetarian by simply dropping the tuples t_4, t_5, t_6 and t_7 as they are menus composed of meat. However this is not the right way to deal with such a preference since it may be the case that the restaurant have no fish-based menus and Clara is hungry and would absolutly like to eat something. So she may accept a meat-based menu then she eats dessert and drinks wine without eating meat. So $meat - red - cake$ should be preferred to $meat - white - ice\_cream$ following (s_2).

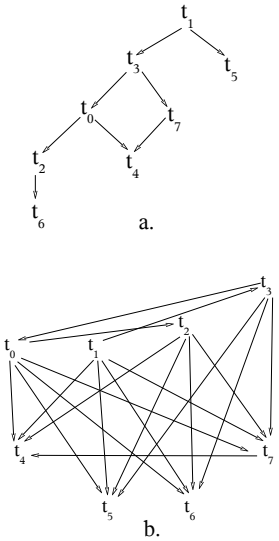


Figure 3:

Indeed a good preference system should not use a unique semantics for all comparative preference statements, as it is the case in current preference-based queries systems, but allows the user to express her preferences w.r.t. different semantics at the same time. Such an approach is consistent with psychological results. Indeed, preference reversals have generated considerable theoretical attention from psychologists (Lichtenstein and Slovic 1971). Several rival psychological accounts of preference reversals have been proposed: expression theory, change-of-process theory, etc. According to change-of-process theory, preference reversals are attributed to variations in the decision strategies used to combine information. In this section we propose an algorithm which handles totalitarian and ceteris paribus semantics used simultaneously.

The algorithm has as an input a set of comparative preference statements of the form $p > q$ interpreted following totalitarian or ceteris paribus semantics or both. Unlike existing approaches the alogorithm does not compute the partial order induced by the set of comparative preference statements $\mathcal{P}$ but a completion of that order following the least specific principle (Yager 1983) inducing a (complete) preorder on r which satisfies each comparative preference statement in $\mathcal{P}$. Let $p >_T q$ (resp. $p >_{CP} q$) denote the preference for p over q following totalitarian (resp. ceteris paribus) semantics.

Definition 2 (Semantics) *Let $\succeq$ be a complete preorder. Let p and q be two formulas and r be an instance of a relational scheme $\mathcal{R}$.*

- *$\succeq$ satisfies $p >_T q$ iff $\forall t, t' \in r$, $t \models p \wedge \neg q$, $t' \models \neg p \wedge q$ we have that $t \succ t'$.*

- $\succeq$ *satisfies* $p >_{CP} q$ *iff* $\forall t, t' \in r$, $t \models p \wedge \neg q$, $t' \models \neg p \wedge q$ *we have that* $t \succ t'$, *where* t *and* t' *are assigned the same values w.r.t. variables that are not involved in* p *and* q.

$\succeq$ *satisfies a set of comparative preference statements* $\mathcal{P}$ *if and only if it satisfies each statement in* $\mathcal{P}$.

We will show that the preorder induced by our algorithm is faithful w.r.t. preference queries since it represents an iterated version of the winnow operator.

The algorithm

Let $\mathcal{P} = \mathcal{P}_T \cup \mathcal{P}_{CP}$ with $\mathcal{P}_T = \{(s_i) : p_i >_T q_i\}$ and $\mathcal{P}_{CP} = \{(s_j) : p_j >_{CP} q_j\}$. $\mathcal{P}_T$ (resp. $\mathcal{P}_{CP}$) is the set of comparative preference statements interpreted following totalitarian (resp. ceteris paribus) semantics. We associate to each comparative preference statement $(s) : p >_x q$, with $x \in \{T, CP\}$, a pair $(L(s), R(s))$ such that $L(s) = \{t | t \in r, t \models p \wedge \neg q\}$ and $R(s) = \{t | t \in r, t \models \neg p \wedge q\}$. We write $t \rhd_s t'$ when t is preferred to t' w.r.t. the comparative preference statement $(s) : p >_{CP} q$.
Let $\mathcal{L}(\mathcal{P}) = \{(L(s_k), R(s_k)) | s_k \in \mathcal{P}\}$. Note that $L(s_k)$ or $R(s_k)$ (or both) may be empty when r is incomplete. In such a case, we remove the pair $(L(s_k), R(s_k))$ from $\mathcal{L}(\mathcal{P})$. We put $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}_T) \cup \mathcal{L}(\mathcal{P}_{CP})$, where $\mathcal{L}(\mathcal{P}_T) = \{(L(s_i), R(s_i)) | s_i \in \mathcal{P}_T\}$ and $\mathcal{L}(\mathcal{P}) = \{(L(s_j), R(s_j)) | s_j \in \mathcal{P}_{CP}\}$.
Due also to the possible incompleteness of the database, comparative preference statements interpreted following ceteris paribus semantics need a pre-treatment since they are based on pairwise comparisons between tuples t and t'. Let $(s_j) : p_j >_{CP} q_j$. Suppose that t belongs to $L(s_j)$. Then t is preferred to t', with $t' \in R(s_j)$, only if $t \rhd_{s_j} t'$. Now due to the possible incompleteness of the database, some tuples t' in $R(s_j)$ are not dominated by any tuple t in $L(s_j)$ and thus should be removed from $R(s_j)$. The same reasoning is followed when a tuple t in $L(s_j)$ does not dominate any tuple t' in $R(s_j)$.

Example 7 *(Example 6 continued)*
Let $r = \{t_0, t_2, t_3, t_6\}$ *with* $t_0 = fish - white - ice\_cream$, $t_2 : fish - red - ice\_cream$, $t_3 : fish - red - cake$ *and* $t_6 : meat - red - ice\_cream$. *Let* $(s) : white >_{CP} red$. *We have* $(L(s), R(s)) = (\{t_0\}, \{t_2, t_3, t_6\})$. *We have that* t_0 *is preferred to* t_2 *w.r.t.* $white >_{CP} red$. *However there is not tuple in* $L(s)$ *which is preferred to* t_3 *and* t_6 *w.r.t.* $white >_{CP} red$. *So they should be removed from* $R(s)$ *since they are not dominated w.r.t.* $white >_{CP} red$. *Indeed* $(L(s), R(s))$ *is replaced by* $(\{t_0\}, \{t_2\})$.

We suppose that such pre-treatment is done before we apply Algorithm 2.

At each iteration of the algorithm, we first select the set of the best tuples, those that are not dominated by any tuple. Undominated tuples are those that do not appear in the right hand side of any pair $(L(s_k), R(s_k))$ in $\mathcal{L}(\mathcal{P})$ (line 2). Then we update the set $\mathcal{L}(\mathcal{P})$. Comparative preference statements (s_i) interpreted following totalitarian semantics are updated by removing tuples of E_l from $L(s_i)$. This is done in line 5. Lastly we remove satisfied comparative preference statements; those with an empty $L(s_i)$ (line 6).

Algorithm 2: A preorder associated to $\mathcal{P} = \mathcal{P}_T \cup \mathcal{P}_{CP}$.

Data: r, $\mathcal{P}$
Result: $\succeq$
begin
  $l = 0$
  while $r \neq \emptyset$ **do**
    1. $l = l + 1$
    2. $E_l = \{t | t \in r, \nexists (L(s_k), R(s_k)) \in \mathcal{L}(\mathcal{P}), t \in R(s_k)\}$
    3. **if** $E_l = \emptyset$ **then**
      stop (inconsistent preference statements)
    4. **for** *each* t *in* E_l **do**
      replace each $(L(s_j), R(s_j))$ in $\mathcal{L}(\mathcal{P}_{CP})$ by $(L(s_j) \backslash \{t\}, R(s_j) \backslash \{t'\})$ when $t \in L(s_j)$, $t' \in R(s_j)$ and $t \rhd_{s_j} t'$
    5. replace each $(L(s_i), R(s_i))$ in $\mathcal{L}(\mathcal{P}_T)$ by $(L(s_i) \backslash E_l, R(s_i))$
    6. remove $(L(s_k), R(s_k))$ with empty $L(s_k)$
    7. $r = r \backslash E_l$
  return $\succeq = (E_1, \cdots, E_l)$.
end

When the set of comparative preference statements $\mathcal{P}$ is consistent i.e., its associated partial order is acyclic, Algorithm 2 returns a complete preorder which satisfies $\mathcal{P}$. Otherwise it returns $(E_1, \cdots, E_l)$ with $E_1 \cup \cdots \cup E_{l-1} \subset r$ and $E_l = \emptyset$. Moreover some comparative preference statements are not satisfied by the preorder. See Example 11.

What about the winnow operator?

Proposition 1 states that the best alternatives w.r.t. the preorder returned by Algorithm 2 are exactly the result of the winnow operator.

Proposition 1 *Let* $\mathcal{P} = \mathcal{P}_T \cup \mathcal{P}_{CP}$. *Let* $\succeq = (E_1, \cdots, E_m)$ *be the preorder associated to* $\mathcal{P}$ *following Algorithm 2. Let* r *be an instance of* $\mathcal{R}(X_1, \cdots, X_n)$. *Then,*

$$W_{\mathcal{P}}(r) = E_1.$$

Example 8 *(Example 6 continued)*
We have $\mathcal{P} = \mathcal{P}_T \cup \mathcal{P}_{CP}$ *with* $\mathcal{P}_T = \{(s_1) : fish >_T meat\}$ *and*
$\mathcal{P}_{CP} = \{(s_2) : red \wedge cake >_{CP} white \wedge ice\_cream, (s_3) : fish \wedge white >_{CP} fish \wedge red\}$.
$\mathcal{L}(\mathcal{P}) = \{(L(s_1), R(s_1))\} \cup \{(L(s_2), R(s_2)), (L(s_3), R(s_3))\} = \{(\{t_0, t_1, t_2, t_3\}, \{t_4, t_5, t_6, t_7\})\} \cup \{(\{t_3, t_7\}, \{t_0, t_4\}), (\{t_0, t_1\}, \{t_2, t_3\})\}$.
We have $E_1 = \{t_1\}$. *We first run "For each" loop. We replace* $(L(s_3), R(s_3))$ *by* $(\{t_0\}, \{t_2\})$ *since* $t_1 \in L(s_3)$, $t_3 \in R(s_3)$ *and* $t_1 \rhd_{s_3} t_3$. *Then we replace* $(L(s_1), R(s_1))$ *by* $(\{t_0, t_2, t_3\}, \{t_4, t_5, t_6, t_7\})$. *We get* $\mathcal{L}(\mathcal{P}) = \{(\{t_0, t_2, t_3\}, \{t_4, t_5, t_6, t_7\})\} \cup \{(\{t_3, t_7\}, \{t_0, t_4\}), (\{t_0\}, \{t_2\})\}$. *We repeat the same reasoning and get* $E_2 = \{t_3\}$, $E_3 = \{t_0\}$,

$E_4 = \{t_2\}$, $E_5 = \{t_5, t_6, t_7\}$ *and* $E_6 = \{t_4\}$. *Indeed* $\succeq = (\{t_1\}, \{t_3\}, \{t_0\}, \{t_2\}, \{t_5, t_6, t_7\}, \{t_4\})$. *We can check that* $W_{\mathcal{P}}(r) = E_1 = \{t_1\}$.

The result of the winnow operator is the set of the best tuples w.r.t. agent's preferences. However it may be the case that these tuples are no longer feasible or available and the computation of the winnow operator should then be applied again on the remaining tuples. Our algorithm has a nice property; it is anytime as at each step of the algorithm it computes the best tuples w.r.t. the current set of tuples. Formally we have the following result which generalizes Proposition 1:

Proposition 2 *Let* $\mathcal{P} = \mathcal{P}_T \cup \mathcal{P}_{CP}$. *Let* $\succeq = (E_1, \cdots, E_m)$ *be the preorder associated to* $\mathcal{P}$ *following Algorithm 2. Let* r *be an instance of* $\mathcal{R}(X_1, \cdots, X_n)$. *Then,*

$$\forall i = 1, \cdots, m, E_i = W^i_{\mathcal{P}}(r),$$

where $W^1_{\mathcal{P}}(r) = W_{\mathcal{P}}(r)$ *and* $W^{n+1}_{\mathcal{P}}(r) = W_{\mathcal{P}}(r \setminus \bigcup_{i=1,\cdots,n} W^i_{\mathcal{P}}(r))$.

Indeed $W^1_{\mathcal{P}}(r)$ is the set of the best tuples, $W^2_{\mathcal{P}}(r)$ is the set of next immediate best tuples, and so on.

Example 9 *(Example 6 continued)*
Let $r' = r \setminus W_{\mathcal{P}}(r) = \{t_0, t_2, t_3, t_4, t_5, t_6, t_7\}$. *We can check that* $W_{\mathcal{P}}(r') = W^2_{\mathcal{P}}(r) = \{t_3\} = E_2$. *Also* $W^3_{\mathcal{P}}(r) = E_3$, $W^4_{\mathcal{P}}(r) = E_4$, $W^5_{\mathcal{P}}(r) = E_5$ *and* $W^6_{\mathcal{P}}(r) = E_6$.

It follows from Proposition 2 that Algorithm 2 returns a complete preorder which is a completion of the partial order induced by the set of comparative preference statements $\mathcal{P}$. There may be several complete preorders which satisfy $\mathcal{P}$. However Algorithm 2 returns a unique complete preorder which is called the *least specific* (Yager 1983) in nonmonotonic reasoning since each tuple has been put in the highest possible equivalence class in the preorder (see Definition 1). So if we move any tuple to a higher equivalence class then the new preorder will violate some comparative preference statements. Let us consider the preorder $\succeq$ computed in Example 8. If we move t_5 from E_5 to E_4 then we get $\succeq' = (\{t_1\}, \{t_3\}, \{t_0\}, \{t_2, t_5\}, \{t_6, t_7\}, \{t_4\})$ which violates $fish >_T meat$. Algorithm 2 is an extended version of an algorithm proposed in (Benferhat, Dubois, and Prade 2001) for handling totalitarian preferences only.

Lastly it is important to notice that the computation of the best tuples (i.e., the winnow operator) following Algorithm 2 is highly attractive from complexity point of view.

Proposition 3 *Let* r *be a relational database and* $\mathcal{P}$ *be a set of comparative preference statements. Let* $\succeq = (E_1, \cdots, E_l)$ *be the preorder associated to* r *and* $\mathcal{P}$ *following Algorithm 2. The computation of* $\succeq$ *is achieved in polynomial time in the size of* r *(i.e., the number of tuples in* r*) and the number of comparative preference statements in* $\mathcal{P}$.

Indeed our algorithm offers better complexity than existing algorithms to compute the winnow operator (Börzsönyi, Kossmann, and Stocker 2001; Chomicki 2003; Torlone and Ciaccia 2002). This is because the best tuples are undominated ones and our algorithm computes them without resorting to dominance queries between all pairs of tuples which is very expensive in complexity.

Encoding database preference queries

In this section, we highlight the expressive power of our framework and show how it recovers existing approaches when reduced to a single semantics.

Preference queries based on CP-nets

As presented before, CP-nets are based on comparative preference statements of the form $p : q > r$ which obey ceteris paribus semantics i.e., $t \succ_{\mathcal{N}} t'$ iff $t \models p \wedge q$ and $t' \models p \wedge r$ and t, t' are the same otherwise. Each preference statement $p : q > r$ is translated in our framework into $p \wedge q >_{CP} p \wedge r$. So $\mathcal{P} = \mathcal{P}_{CP} = \{(s_j) : p_j \wedge q_j >_{CP} p_j \wedge r_j\}$, where $p_j : q_j > r_j$ is expressed in preference tables associated with the CP-net that we intend to encode. Since $\mathcal{P}$ is composed of ceteris paribus preferences only, Algorithm 2 is reduced to Algorithm 3.

Algorithm 3: A preorder associated to $\mathcal{P}_{CP}$.

Data: r, $\mathcal{P}_{CP}$
Result: $\succeq$
begin
  $l = 0$
  while $r \neq \emptyset$ **do**
    $l = l + 1$
    $E_l = \{t | t \in r, \nexists (L(s_j), R(s_j)) \in \mathcal{L}(\mathcal{P}_{CP}), t \in R(s_j)\}$
    if $E_l = \emptyset$ **then**
      stop (inconsistent preference statements)
    for *each* t *in* E_l **do**
      replace each $(L(s_j), R(s_j))$ in $\mathcal{L}(\mathcal{P}_{CP})$ by $(L(s_j) \setminus \{t\}, R(s_j) \setminus \{t'\})$ when $t \in L(s_j)$, $t' \in R(s_j)$ and $t \rhd_{s_j} t'$
    - remove $(L(s_j), R(s_j))$ with empty $L(s_j)$
    - $r = r \setminus E_l$
  return $\succeq = (E_1, \cdots, E_l)$.
end

Then we have the following result:

Proposition 4 *Let* $\mathcal{N}$ *be a CP-net and* $\mathcal{P}_{CP}$ *be the set of comparative preference statements built from* $\mathcal{N}$. *Let* $\succeq = (E_1, \cdots, E_m)$ *be the preorder associated to* $\mathcal{P}_{CP}$ *following Algorithm 3. Then,*

$$\forall i = 1, \cdots, m, W^i_{\mathcal{N}}(r) = E_i.$$

Example 10 *(Brafman and Domshlak 2004)*
Let $\mathcal{N}$ *be a CP-net built on three attributes* dish, wine *and* dessert *taking their values in* $\{fish, meat\}$, $\{white, red\}$ *and* $\{cake, ice\_cream\}$ *respectively. The CP-net* $\mathcal{N}$ *and its associated partial order are depicted in Figure 4. We have* $\mathcal{P}_{CP} = \{fish >_{CP} meat, fish \wedge white >_{CP} fish \wedge red, meat \wedge red >_{CP} meat \wedge white, white \wedge cake >_{CP} white \wedge ice\_cream, red \wedge ice\_cream >_{CP} red \wedge cake\}$. *Then* $\mathcal{L}(\mathcal{P}_{CP}) = \{(\{t_0, t_1, t_2, t_3\}, \{t_4, t_5, t_6, t_7\}), (\{t_0, t_1\}, \{t_2, t_3\}), (\{t_6, t_7\}, \{t_4, t_5\}), (\{t_1, t_5\}, \{t_0, t_4\}), (\{t_2, t_6\}, \{t_3, t_7\})\}$. *Let us now apply Algorithm 3 on* $\mathcal{P}_{CP}$. *We get* $\succeq = (E_1, E_2, E_3, E_4, E_5, E_6, E_7)$ *with* $E_1 = \{t_1\}$, $E_2 = \{t_0\}$, $E_3 = \{t_2\}$, $E_4 = \{t_3, t_6\}$, $E_5 = \{t_7\}$, $E_6 = \{t_5\}$ *and* $E_7 = \{t_4\}$. *We can check that* $E_i = W^i_{\mathcal{N}}(r)$ *for* $i = 1, \cdots, 7$.

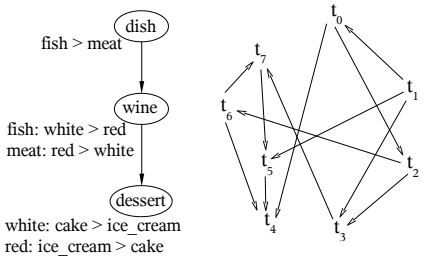


Figure 4: A CP-net $\mathcal{N}$ and its associated partial order.

Preference queries based on formulas

We first present the encoding of a unique preference relation $\succ_C$. As noticed before, comparative preference statements are interpreted in Chomicki's approach following totalitarian semantics. So given a formula C, we construct a set of totalitarian preferences $\mathcal{P}_T = \{p_1 >_T q_1, \cdots, p_l >_T q_l\}$ as previously described. Since we are dealing with totalitarian preferences only, Algorithm 2 is reduced to Algorithm 4.

Algorithm 4: A preorder associated to $\mathcal{P}_T$.

Data: r, $\mathcal{P}_T$
Result: $\succeq$
begin
  $l = 0$
  while $r \neq \emptyset$ **do**
    $l = l + 1$
    $E_l = \{t | t \in r, \nexists (L(s_i), R(s_i)) \in \mathcal{L}(\mathcal{P}_T), t \in R(s_i)\}$
    if $E_l = \emptyset$ **then**
      stop (inconsistent preference statements)
    - replace each $(L(s_i), R(s_i))$ in $\mathcal{L}(\mathcal{P}_T)$ by $(L(s_i) \backslash E_l, R(s_i))$
    - remove $(L(s_i), R(s_i))$ with empty $L(s_i)$
    - $r = r \backslash E_l$
  return $\succeq = (E_1, \cdots, E_l)$.
end

Proposition 5 *Let r be an instance of $\mathcal{R}(X_1, \cdots, X_n)$. Let C be a preference formula and $\mathcal{P}_T$ be the set of totalitarian preferences built from C. Let $\succeq = (E_1, \cdots, E_m)$ be the preorder associated to $\mathcal{P}_T$ following Algorithm 4. Then,*

$$\forall i = 1, \cdots, m, W^i_C(r) = E_i.$$

Example 11 *(Example 1 continued)*
Let us consider $\succ_{C_1}$. We have $\mathcal{P}_1 = \{meat \wedge red >_T meat \wedge white, fish \wedge white >_T fish \wedge red\}$. Applying Algorithm 4 returns (E_1, E_2) with $E_1 = \{meat - red, fish - white, meat - rosé, fish - rosé\}$ and $E_2 = \{meat - white, fish - red\}$. Indeed, $E_1 = W_{C_1}(r)$ and $E_2 = W^2_{C_1}(r)$.
We now consider $\succ_{C_2}$. We have $\mathcal{P}_2 = \{red >_T white, fish \wedge white >_T fish \wedge rosé, fish \wedge rosé >_T fish \wedge red\}$. Following Algorithm 4, $\mathcal{P}_2$ is inconsistent and the associated preorder is (E_1, E_2) with $E_1 = \{meat - red, meat - rosé\}$ and $E_2 = \emptyset$. So we have $W_{C_2}(r) = E_1$ and $W^2_{C_2}(r) = E_2$.

Let us now present the encoding of the composition of preference queries. Let C_1 and C_2 be two preference formulas and $\succ_{C_1}$ and $\succ_{C_2}$ be two preference relations over the same relation schema. As noticed before, we distinguish two types of composition: boolean and prioritized. Boolean composition is repesented by intersection, union and difference which are defined by $C_1 \wedge C_2$, $C_1 \vee C_2$ and $C_1 \wedge \neg C_2$ respectively. Since any logical formula can be written in a disjunctive form, we can simply use the encoding presented above to encode boolean composition.
Let us now give the encoding of prioritized composition. Let $\succ_p$ be the prioritized composition of $\succ_{C_1}$ over $\succ_{C_2}$. Recall that we have:

$$t \succ_p t' \text{ iff } ((t \succ_{C_1} t') \vee (t \sim_{C_1} t' \wedge t \succ_{C_2} t')).$$

Lemma 3 *Let C_1 and C_2 be two preference formulas. Let C be a preference formula C defined by:*
$C(t, t') \equiv C_1(t, t') \vee (\neg C_1(t, t') \wedge \neg C_1(t', t) \wedge C_2(t, t'))$.
Let $\succ_p$ be the result of prioritized composition of $\succ_{C_1}$ over $\succ_{C_2}$. Then,

$$\forall t, t' \in r,\ t \succ_p t' \textit{ iff } C(t, t').$$

Once C is computed we can use the encoding given in Proposition 5 for a unique preference relation.

Example 12 *Let $\succ_{C_4}$ and $\succ_{C_5}$ be two preference relations defined by*
$t \succ_{C_4} t'$ iff $(t.wine = red \wedge t'.wine \neq red)$
$t \succ_{C_5} t'$ iff $((t.dish = fish \wedge t.wine = white) \wedge (t'.dish = fish \wedge t'.wine \neq white))$.
Let $\succ_p$ be the prioritized composition of $\succ_{C_5}$ over $\succ_{C_4}$.
The preference relations $\succ_{C_4}$, $\succ_{C_5}$ and $\succ_p$ are depicted in Figure 5. After simplification we have that $C(t, t')$ is equivalent to $(t.dish = meat \wedge t.wine = red \wedge t'.wine \neq red) \vee (t.wine = red \wedge t'.dish = meat \wedge t'.wine \neq red) \vee (t.wine = red \wedge t'.wine = rosé) \vee (t.dish = fish \wedge t.wine = white \wedge t'.dish = fish \wedge t'.wine \neq white)$.
The set of comparative preference statements associated to C is $\mathcal{P}_T = \{meat \wedge red >_T \neg red, red >_T meat \wedge \neg red, red >_T rosé, fish \wedge white >_T fish \wedge \neg white\}$. Applying Algorithm 4 returns $\succeq = (E_1, E_2, E_3, E_4)$ with $E_1 =$

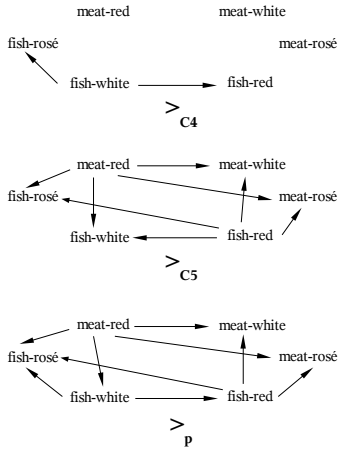


Figure 5: Preference relations $\succ_{C_4}, \succ_{C_5}, \succ_p$.

$\{meat-red\}$, $E_2 = \{fish-white\}$, $E_3 = \{fish-red\}$ *and* $E_4 = \{meat-white, meat-rosé, fish-rosé\}$. *From Figure 5 we can check that* $meat-red$ *is the most preferred tuple w.r.t.* $\succ_p$, $fish-white$ *is immediately less preferred,* $fish-red$ *is less preferred and lastly* $meat-white$, $meat-rosé$ *and* $fish-rosé$ *are the least preferred tuples.*

Conclusion

Managing users preferences is an important problem in database queries. After analyzing existing approaches for the compact representation of preferences in database queries, we proposed a unified framework where two semantics of users preferences are dealt with: strong semantics and ceteris paribus semantics. Our algorithm to handle both semantics is based on specificity principle which makes it very attractive from complexity point of view. More precisely, computing the best tuples is achieved in polynomial time complexity in the number of user's preferences and the number of tuples in the database.

The present work can be extended in different directions. First we expect to integrate other semantics of preferences (Wilson 2004). Then we intend to extend this work to handle preference queries expressed on different relational schemas. Lastly, we will compare the use of specificity principle in our framework and its use in (Lukasiewicz and Schellhase 2007).

References

Benferhat, S., and Kaci, S. 2001. A possibilistic logic handling of strong preferences. In *IFSA'01*, 962–967.

Benferhat, S.; Dubois, D.; and Prade, H. 2001. Towards a possibilistic logic handling of preferences. *Applied Intelligence* 14(3):303–317.

Börzsönyi, S.; Kossmann, D.; and Stocker, K. 2001. The skyline operator. In *ICDE*, 421–430.

Boutilier, C.; Brafman, R.; Hoos, H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *UAI*, 71–80.

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR* 21:135–191.

Brafman, R., and Domshlak, C. 2004. Database preference queries revisited. In *Technical report*.

Chang, C. 1976. Deduce – a deductive query language for relational data base. In *Pattern Recognition and AI*, 108–112.

Chomicki, J. 2002. Querying with intrinsic preferences. In *ICED*, 34–51.

Chomicki, J. 2003. Preference formulas in relational queries. *ACM transactions on Database Systems* 28(4):1–40.

Fishburn, P. 1999. Preference structures and their numerical representations. *Theoretical Computer Science* 217(2):359–383.

Kaci, S., and van der Torre, L. 2008. Reasoning with various kinds of preferences: Logic, non-monotonicity, and algorithms. *Annals of Operation Research* in press.

Kiessling, W. 2002. Foundations of preferences in database systems. In *VLDB'02*, 311–322.

Lichtenstein, S., and Slovic, P. 1971. Reversals of preference between bids and choices in gambling decisions. *Journal of Experimental Psychology* 89(1):46–55.

Lindman, H. 1971. Inconsistent preferences among gambles. *Journal of Experimental Psychology* 89(2):390–397.

Lukasiewicz, T., and Schellhase. 2007. Variable-strength conditional preferences for ranking objects in ontologies. *Journal of Web Semantic* 5(3):180–194.

Motro, A. 1986. Supporting goal queries in relational database. In *ICEDS'86*, 85–96.

Sjoberg, L. 1975. Uncertainty of comparative judgments and multidimensional structure. *Multivariate Behavioral Research* 10(2):207–218.

Torlone, R., and Ciaccia, P. 2002. Finding the best when it's a matter of preference. In *SEBD*, 347–360.

Tversky, A. 1969. Intransitivity of preference. *Psychological Review* 76:31–48.

Wilson, N. 2004. Extending cp-nets with stronger conditional preference statements. In *AAAI*, 735–741.

Yager, R. 1983. Entropy and specificity in a mathematical theory of evidence. *International Journal of General Systems* 9:249–260.

A Characterization of an Optimality Criterion for Decision Making under Complete Ignorance

Ramzi Ben Larbi Sébastien Konieczny Pierre Marquis
CRIL - CNRS
Université d'Artois, Lens, France
{benlarbi, konieczny, marquis}@cril.fr

Abstract

In this paper we present a model for decision making under complete ignorance, where each action is associated directly with its set of possible consequences (especially, there is no set of states that can be enumerated in order to compare the actions). We present two axioms designed to encapsulate the minimal requirements about decision making under complete ignorance. We show that in this setting, the family of optimality criteria meeting the requirements of these axioms have a suitable form and only take account for the extremal consequences.

Introduction

The traditional framework for decision making under uncertainty considers a set S of possible states of the world, a set A on actions and a set C of potential consequences of actions. In this setting an action a is defined as a mapping matching a state of the world to a consequence. Every consequence is associated to a utility that describes the agent satisfaction of reaching it. The agent has to choose an action without knowing the right state of the world in which it is performed. This decision problem is often treated by assuming that the agent is guided by a probability distribution p over the states of the world. This distribution can be either objective (statistically obtained) or subjective (derived from the agent beliefs). The agent then uses the expected utility theory (von Neumann & Morgenstern 1947; Savage 1954) to evaluate her actions. The most interesting actions are those with the highest expected utility.

The effectiveness of such an approach lies on the hypotheses made on the actions representation (as mappings between states and consequences) and on the availability of a probability distribution. Given a utility function u on the consequences and a probability distribution p over the states of the world, the expected utility EU of an action a is defined as

$$EU_{p,u}(a) = \sum_{s \in S} p(s) * u(a(s))$$

Alternative approaches referred to as qualitative approaches to decision making under uncertainty have been developed, see (Bouyssou *et al.* 2006) which surveys many other works including (Dubois *et al.* 2002). They can be seen as a way of overcoming some drawbacks of the expected utility approach. The data needed for adopting a probabilistic approach may indeed be too sophisticated for every day decisions and better suited for critical ones. Besides, the use of this approach may require the handling of a huge number of possible states of the world as it is the case for example in the Partially Observable Markovian Process framework (Lovejoy 1991).

In the qualitative approaches, an action is still considered as a mapping associating each state of the world to a consequence. Even if they start from the basic assumptions underlying Savage axiomatic framework (Savage 1954), i.e., a preorder over the states of the world expressing their plausibility, and make the necessary changes to express ordinality, the obtained criteria use only this preorder and do not derive a probability distribution to represent it. An example of such a qualitative model is (Boutilier 1994) where decisions are made based on the most plausible states, neglecting the others. The qualitative approaches to decision making can lead to different settings depending on assuming commensurabiltiy or not, i.e., whether the plausibility and utility scales can be compared. However, as stated in (Bouyssou *et al.* 2006), "a fully qualitative theory of decision is a theory which must not make use of the expressive power of numbers neither to model uncertainty nor to represent utility".

Some works studied the concept of complete ignorance (sometimes referred to as *strict uncertainty*) in such a setting. One of the main contributions is by Arrow and Hurwicz (Arrow & Hurwicz 1977). In decision under ignorance individual decision makers have to choose one of a finite number of alternatives with complete information about their consequences but in the absence of any information about the probabilities/plausibility of the various states of the world.

In (Arrow & Hurwicz 1977), the authors still make the assumption that an action is a mapping which maps a state of the world to a consequence and that the agent perfectly knows this mapping. They adopt an axiomatic approach, that is suggesting a set of properties which might characterize a criterion for decision-making under ignorance. They propose a characterization of a rational optimality criterion in this context, i.e., a defined procedure which gives for every problem a set of best actions to be chosen.

"Ignorance" prevents from the use of the expected utility

theory. As in the work of Arrow and Hurwicz (Arrow & Hurwicz 1977), the principle of insufficient reason (**PIR**) - also called Laplace principle - (see e.g., (Jaynes 2003)) may seem to be an intuitive solution for dealing with ignorance. **PIR** states that when facing n states which are indistinguishable except for their names, each state should be assigned a probability equal to 1/n. Yet, as Arrow and Hurwicz remark it, this principle may not be adequate for expressing complete ignorance. It can lead to contradictory beliefs for the same situation only by changing the description of the consequences associated to an action. Let us consider the following scenario. An agent has to bet on a horse race involving the horse H. If the player bets on H she will receive 0 dollars if H loses and 5 dollars otherwise. Then the **PIR** will lead the agent to assign a probability of 0.5 to the consequense 0 dollars. Now suppose that the agent knows she gets 0 dollars in the case of loss and 5 dollars in coins or a bill of 5 dollars otherwise. Then the use of **PIR** will lead the agent to assign a probability 0.33 to the consequence 0 dollars. This is not satisfactory since we changed nothing but the description of consequences (and not the consequences themselves).

In (Maskin 1979), the author extends the result of (Arrow & Hurwicz 1977). By adding additional axioms, he shows the unique characterisation of many decision criteria, some of which being widely used. He further illustrates how to obtain qualitative criteria by dropping the assumption that preferences over consequences satisfy the von Neumann-Morgenstern axioms.

Let us note here that all the previously cited works invariably consider the same definition of an action, i.e., a maping between states of the world and consequences and that the uncertainty is on the right state of the world.

We address here the problem of decision making under complete ignorance but consider a different structure for the available information. We suppose that an action is associated directly to a set of possible consequences, i.e., the consequences that can be reached when the action is performed; thus the right correspondance between states of the world and consequences realised by an action does not need to be known. So this is a framework which supposes that the decision maker has less information than in the ones presented above.It offers a more realistic representation of uncertainty when this uncertainty is about the action execution and does not depend solely on the initial state.

In the following, we adopt a similar methodology to the one in (Arrow & Hurwicz 1977) in order to characterize the structure of admissible optimality criteria in this setting via a set of axioms. Still our representation of actions is different from their one. That induces differences in the properties and the proofs.

We will start by introducing the framework. Then we will present the desired properties of an optimality criterion. We will prove some preliminary results leading to a theorem delimiting the family of admissible criteria. A second theorem refining the first one will also be proven before presenting some examples of admissible criteria.

Framework

A decision problem A is a set of actions. The problem of decision making under ignorance consists in assigning a subset $\hat{A}$ to a decision problem A, called the optimal set for A. We consider a fixed class P^* of decision problems, each of which is supposed to have a non-empty optimal set.

A consequence of an action $a \in A$ will not be associated to a real number. The only hypothesis made on the consequence set C is that the agent can compare any two alternatives $c, c^{'} \in C$ by means of a total preorder $\geqslant$, that is a reflexive, transitive, and total (complete) relation. An action a is associated to a non-empty subset of possible consequences by the means of a result function $R : A \rightarrow 2^C \setminus \{\emptyset\}$.

This structure of information is different from the one assumed in the previously presented approaches to uncertainty. The information about possible consequences of an action is the minimal information an agent needs about her environment to make a decision. In that sense, when trying to represent complete ignorance, the structure previously used of a mapping associating every state of the world to a consequence is more informative.

Complete ignorance is expressed by the fact that given an action a, the subset of possible results $R(a)$ is the only available information. Once the action executed, the realised consequence will be an element of $R(a)$. No probability distribution on the image of any action by R is available, which can be interpreted as "all the probability distributions on consequences are possible". In the same way, no plausibility relation on the states of the world is available to the agent. Thus complete ignorance also requires that the adopted approach must depart from the previously discussed qualitative approaches.

If $a_1, a_2 \in A$ and $a_1 \in \hat{A}$, we say that a_1 is preferred or indifferent to a_2 with respect to A and symbolise it by $a_1(\geqq)a_2[A]$.

If $a_1, a_2 \in A$, and either $a_1, a_2 \in \hat{A}$ or $a_1, a_2 \in A \setminus \hat{A}$, we say that a_1 and a_2 are optimally equivalent with respect to A, symbolised by $a_1 \doteq a_2[A]$.

Similarly, if $a_1, a_2 \in A$, $a_1 \in \hat{A}$ and $a_2 \in A \setminus \hat{A}$, a_1 is said to be preferred to a_2 with respect to A; the relation is symbolised by $a_1(>)a_2[A]$.

In particular, for decision problems containing just the actions a_1, a_2 and denoted by $[a_1, a_2]$, we simplify the notations as follows:
$a_1 \doteq a_2$ if $a_1 \doteq a_2[a_1, a_2]$,
$a_1(\geqq)a_2$ if $a_1(\geqq)a_2[a_1, a_2]$,
$a_1(>)a_2$ if $a_1(>)a_2[a_1, a_2]$.

We denote $minR(a)$ the set of minima of $R(a)$ and $maxR(a)$ the set of maxima of $R(a)$ with respect to the preference relation on consequences, i.e., $c \in minR(a)$ iff $\nexists c^{'} \in R(a)$ s.t. $c > c^{'}$, and $c \in maxR(a)$ iff $\nexists c^{'} \in R(a)$ s.t. $c^{'} > c$.

Let $[c_1, \cdots, c_n]$ be an action such that $R([c_1, \cdots, c_n]) = \{c_1, \cdots, c_n\}$.

Expected Properties

The class P^* of decision problems which have non-empty optimal sets is assumed to satisfy the following assumptions.

Assumption (A) *For every considered problem $A \in P^*$, $\hat{A} \neq \emptyset$.*

We consider only problems containing a non-empty optimal set.

Assumption (B) $\forall A \in P^*$, $\forall a \in A$, $minR(a)$ *and* $maxR(a)$ *exist.*

We assume that for every action, the associated set of consequences always contains consequences that are minimal and consequences that are maximal with respect to the agent preference relation over consequences. This assumption prevents us from treating cases with infinite descending/ascending chains of comparison.

These two assumptions delimitate the studied problems. They move aside degenerate cases where an optimal set cannot be derived.

Assumption (C) $\forall C' \subseteq C$, $\exists A \in P^*$, $\exists a \in A$ *s.t.* $R(a) = C'$.

This assumption expresses a density property of the set of possible consequences. It states that for every set of consequences we can envision an action whose possible consequences are exactly the ones in the set. For every subset $C' = \{c_1, c_2, \ldots\}$ of C, we denote by $[c_1, c_2, \ldots]$ an action a of A s.t. $R(a) = C'$ (the existence of such an action is ensured by the assumption).

Finally, an optimality criterion is a rule defining $\hat{A}$ for every $A \in P^*$.

We now start by presenting the desirable properties the admissible criteria should have and then deduce the corresponding form of these criteria.

Property ($\boldsymbol{\alpha}$) *If $A_1 \subseteq A_2$ and $A_1 \cap \hat{A}_2$ is non-empty, then $\hat{A}_1 = A_1 \cap \hat{A}_2$.*

That is, if an action is deemed optimal in a given set of alternatives and if subsequently the range of alternative actions available is contracted but the optimal action is still available, then the optimal action for the larger problem is still optimal (and no new optimal actions are added). This is a typical property of choice functions used for example in social choice theory, belief revision, belief merging, etc. This property requires intuitively that the agent considers optimality as issued from a comparison relation.

Property ($\boldsymbol{\beta}$) *Let A be a decision problem, $a, a', a_1, a'_1 \in A$ such that $\forall c \in R(a), \forall c' \in R(a'), c \geqslant c'$ then $\forall c'' \in C$, if $R(a_1) = R(a) \cup \{c''\}$ and $R(a'_1) = R(a') \cup \{c''\}$ then $a_1(\geqq)a'_1$.*

If two actions are such that they share one common consequence and if all the other consequences of the first are preferred or indifferent to all other consequences of the second then the first action is preferred or indifferent to the second one.

A desirable feature of this property is that it expresses the idea of ignorance. Assuming complete ignorance, it is natural to compare actions based on the only available information: their respective sets of consequences. This implies that the actions leading to the same sets of consequences have to be considered as indifferent. This can be deduced from the properties as shown in the forthcoming Proposition 1.

Some Preliminary Results

In this section, we prove some results that are useful for characterizing optimality criteria.

Lemma 1

1. *if $a_1 \doteq a_2$, then $a_1 \doteq a_2[A]$ for all A s.t. $a_1, a_2 \in A$.*
2. *if $a_1(\geqq)a_2[A]$ for some A, then $a_1(\geqq)a_2$.*
3. *if $a_1(>)a_2[A]$ for some A, then $a_1(>)a_2$.*

Proof: Let us first prove point 1. If a_1, a_2 are not optimally equivalent with respect to A, suppose w.l.o.g. $a_1 \in \hat{A}$, $a_2 \in A \setminus \hat{A}$. Since $\{a_1, a_2\}$ intersects $\hat{A}$, the optimal set for $\{a_1, a_2\}$ consists of the element a_1 by Property ($\boldsymbol{\alpha}$). That contradicts the hypothesis. Points 2 and 3 follow by similar applications of Property ($\boldsymbol{\alpha}$). □

This lemma mainly states that the comparison between two actions depends only of these two actions (and their consequences), and not of any other available action.

Lemma 1 thus enables us, when examining a decision problem, to add intermediary (and possibly imaginary) actions or to move aside existing ones in order to make easier the determination of the optimal set.

The next lemma specifies in which way Property ($\boldsymbol{\beta}$) encapsulates the idea of complete ignorance. It states that, when considering an action and its corresponding set of consequences, the only consequences that have to be taken into account to state about the action optimality are the extremal ones. That is the couple formed by the worst consequence and the best consequence for the agent.

Lemma 2 *Let $a \in A$. $[minR(a), maxR(a)] \doteq a$.*

Proof: Define a_1, a_n, a'_1, a'_n by $R(a_1) = \{minR(a)\}$, $R(a_n) = \{maxR(a)\}$, $R(a'_1) = R(a) \setminus \{minR(a)\}$, $R(a'_n) = R(a) \setminus \{maxR(a)\}$. Then we have $\forall c \in R(a_1), \forall c' \in R(a'_n), c' \geqslant c$. Since $R([minR(a), maxR(a)]) = R(a_1) \cup \{maxR(a)\}$ and $R(a) = R(a'_n) \cup \{maxR(a)\}$, Property ($\boldsymbol{\beta}$) together with point 2 of Lemma 1 gives $a(\geqq)[minR(a), maxR(a)]$. On the other hand, we have $\forall c \in R(a_n), \forall c' \in R(a'_1), c \geqslant c'$. Since $R([minR(a), maxR(a)]) = R(a_n) \cup \{minR(a)\}$ and $R(a) = R(a'_1) \cup \{minR(a)\}$, Property ($\boldsymbol{\beta}$) together with point 2 of Lemma 1 gives $[minR(a), maxR(a)](\geqq)a$. Finally, we can conclude $a \doteq [minR(a), maxR(a)]$. □

Now as stated in the previous section we can easily prove the following corollary:

Corollary 1 *Let a, a' be in a decision problem A. If $R(a) = R(a')$ then $a \doteq a'$.*

Proof: Let $[minR(a), maxR(a)]$ be an action such that $R([minR(a), maxR(a)]) = \{minR(a), maxR(a)\}$. By the previous lemma, it comes directly that :

$$a \doteq [minR(a), maxR(a)] \doteq a'.$$

□

This result mainly states that the only relevant information for comparing two actions are their set of consequences. So actions with the same set of consequences are undistinguishable. This means that actions can be identified to sets of consequences.

Characterizing Optimality Criteria

In this section we first state and prove a theorem which makes explicit the form of an optimality criterion meeting the requirements of Properties $(\boldsymbol{\alpha})$ and $(\boldsymbol{\beta})$.

Theorem 1 *Under assumptions (A), (B) and (C), an optimality criterion satisfies properties $(\boldsymbol{\alpha})$ and $(\boldsymbol{\beta})$ if and only if there exists a total preorder $\succcurlyeq$ on the set of consequence pairs (m, M) with $M \geqslant m$ such that the conditions (1) and (2) are fulfilled:*

(1) If $m_1 \geqslant m_2$ and $M_1 \geqslant M_2$ then $(m_1, M_1) \succcurlyeq (m_2, M_2)$.

(2) $\forall A \in P^*$, $\hat{A} = \{a | (minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')), \forall a' \in A\}$.

Proof: To prove necessity, we assume the existence of an optimality criterion satisfying Properties $(\boldsymbol{\alpha})$ and $(\boldsymbol{\beta})$. We first define the ordering over the half space of ordered pairs of consequences (m, M). Let $[m, M]$ be an action such that $R([m, M]) = \{m, M\}$. We define:

$$(m_1, M_1) \succcurlyeq (m_2, M_2) \text{ iff } [m_1, M_1](\geqq)[m_2, M_2][A].$$

It must first be shown that the relation $\succcurlyeq$ is a total preorder. Reflexivity comes immediately from Lemma 1 and Corollary 1. As to transitivity, suppose $(m_1, M_1) \succcurlyeq (m_2, M_2)$ and $(m_2, M_2) \succcurlyeq (m_3, M_3)$. Let A be the decision problem with elements $[m_1, M_1], [m_2, M_2], [m_3, M_3]$. There is at least one optimal action. Suppose that $[m_3, M_3]$ is the only optimal action, then $[m_3, M_3](>)[m_2, M_2][A]$. By Lemma 1, $[m_3, M_3](>)[m_2, M_2]$, which contradicts the hypothesis that $(m_2, M_2) \succcurlyeq (m_3, M_3)$. Suppose now that $[m_2, M_2] \in \hat{A}$ but $[m_1, M_1] \notin \hat{A}$. This also leads to a contradiction since $(m_1, M_1) \succcurlyeq (m_2, M_2)$. Hence, we must conclude that $[m_1, M_1] \in \hat{A}$, so that $[m_1, M_1](\geqq)[m_3, M_3][A]$, and therefore $[m_1, M_1](\geqq)[m_3, M_3]$, or, by definition, $(m_1, M_1) \succcurlyeq (m_3, M_3)$. The relation $\succcurlyeq$ is therefore transitive.

Since any decision problem $\{[m_1, M_1], [m_2, M_2]\}$ has at least one optimal element, either $(m_1, M_1) \succcurlyeq (m_2, M_2)$ or $(m_2, M_2) \succcurlyeq (m_1, M_1)$, so that the relation $\succcurlyeq$ is a total preorder.

Suppose now that $m_1 \geqslant m_2, M_1 \geqslant M_2$. Then

$$[m_1, M_2](\geqq)[m_2, M_2][A]$$

(since $m_1 \geqslant m_2$ and Property $(\boldsymbol{\beta})$ enables us to conclude by adding M_2). Besides

$$[m_1, M_1](\geqq)[m_1, M_2][A]$$

(since $M_1 \geqslant M_2$ and Property $(\boldsymbol{\beta})$ enables us to conclude by adding m_1). Thus we finally have $[m_1, M_1](\geqq)[m_2, M_2][A]$ and by definition $(m_1, M_1) \succcurlyeq (m_2, M_2)$. We can then conclude the first point of the necessity part of the theorem.

Suppose that $a \in \hat{A}$. Then $a(\geqq)a'[A]$ for all $a' \in A$, and therefore $a(\geqq)a'$ for all $a' \in A$. By Lemma 2 we have $[minR(a), maxR(a)] \doteq a$ and $[minR(a'), maxR(a')] \doteq a'$.
By Lemma 1 we conclude

$$[minR(a), maxR(a)](\geqq)[minR(a'), maxR(a')][A]$$

and by definition

$$(minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')) \text{ for all } a' \in A.$$

Suppose now that $a \in A \setminus \hat{A}$. Let a' be any element of $\hat{A}$, we have $a'(>)a$. By Lemma 2 we have $[minR(a), maxR(a)] \doteq a$ and $[minR(a'), maxR(a')] \doteq a'$.
So we can conclude:

$$[minR(a'), maxR(a')](>)[minR(a), maxR(a)][A]$$

and by definition

$$(minR(a'), maxR(a')) \succ (minR(a), maxR(a)) \text{ for some } a' \in A.$$

We thus proved the second point in the necessity part of the theorem.

Let us now prove the sufficiency part of the theorem. Suppose we have a total preorder on pairs of consequences fulfilling the conditions (1) and (2) of the theorem. Let us verify Property $(\boldsymbol{\alpha})$. Suppose that we have $A_1 \subseteq A_2$ and $A_1 \cap \hat{A}_2$ is non-empty.

Let $a \in A_1 \cap \hat{A}_2$. As $a \in \hat{A}_2$ by condition (2) of the theorem we have $(minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')), \forall a' \in A_2$. Since $A_1 \subseteq A_2$, $a \in \hat{A}_1$, we have $(minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')), \forall a' \in A_1$. We have thus $A_1 \cap \hat{A}_2 \subseteq \hat{A}_1$.

Let $a \in \hat{A}_1$. By condition (2) we have $(minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')), \forall a' \in A_1$ and especially for the elements of $A_1 \cap \hat{A}_2$ which implies $a \in \hat{A}_2$ because of the transitivity of $\succcurlyeq$ and then $a \in A_1 \cap \hat{A}_2$. We have thus $\hat{A}_1 \subseteq A_1 \cap \hat{A}_2$.

Finally we can conclude $\hat{A}_1 = A_1 \cap \hat{A}_2$.

Now let us verify Property $(\boldsymbol{\beta})$. Let a, a' be such that $\forall c \in R(a), c' \in R(a'), c \geqslant c'$. And let $c'' \in C$. Let a_1, a'_1 such that $R(a_1) = R(a) \cup \{c''\}$ and $R(a'_1) = R(a') \cup \{c''\}$.

- If $c'' \geqslant maxR(a)$, then $minR(a_1) \geqslant minR(a'_1)$ and $maxR(a_1) = maxR(a'_1)$.
- If $minR(a') \geqslant c''$, then $minR(a_1) = minR(a'_1)$ and $maxR(a_1) \geqslant maxR(a'_1)$.
- If $maxR(a) \geqslant c'' \geqslant minR(a')$, then $min(c'', minR(a_1)) \geqslant minR(a'_1)$ and $maxR(a_1) \geqslant max(maxR(a'_1), c'')$.

We can see that in every case, adding a common consequence we are again in the situation where the condition (1) of the theorem applies. Indeed, we have $minR(a_1) \geqslant minR(a'_1)$ and $maxR(a_1) \geqslant maxR(a'_1)$, which implies by condition (1) that $(minR(a_1), maxR(a_1)) \succcurlyeq (minR(a'_1), maxR(a'_1))$. □

Remark that the theorem characterizes in fact a family of possible criteria. This is due to condition (1) of the theorem that is only an implication enabling further distinctions to be made.

This theorem is interesting because the relatively simple requirements induce a family of criteria which depend only on the extremal consequences with respect to the agent's preference relation on consequences. This was already the case in the theorem proved by (Arrow & Hurwicz 1977). What we prove here is that their result could be extended to the poorer informational case (w.r.t. actions) we consider.

A Refined Version with Strict Preferences

We have previously defined two axioms expressing some basic requirements about an optimality criterion an agent should use given her complete ignorance about the consequences of an action. This theorem states that such an optimality criterion only depends on the pair formed by the less rewarding and the most rewarding consequence of each action.

Consider the following property:

Property ($\boldsymbol{\beta'}$) *Let A be a decision problem,* $a, a', a_1, a'_1 \in A$ *such that* $\forall c \in R(a), \forall c' \in R(a'), c > c'$ *then* $\forall c'' \in C$, *if* $R(a_1) = R(a) \cup \{c''\}$ *and* $R(a'_1) = R(a') \cup \{c''\}$ *then* $a_1(>)a'_1$.

We can obtain a theorem similar to Theorem 1 by adding the Property ($\boldsymbol{\beta'}$) which has as an effect to add a supplementary condition on the preorder defined in Theorem 1.

We then obtain the following theorem:

Theorem 2 *Under assumptions (A), (B) and (C), an optimality criterion satisfies Properties* ($\boldsymbol{\alpha}$), ($\boldsymbol{\beta'}$) *and* ($\boldsymbol{\beta}$) *if and only if there exists a total preorder* $\succcurlyeq$ *on the set of consequence pairs* (m, M) *with* $M \geqslant m$ *such that the conditions (1),(2) and (3) are fulfilled:*

(1) If $m_1 \geqslant m_2$ *and* $M_1 \geqslant M_2$ *then* $(m_1, M_1) \succcurlyeq (m_2, M_2)$.

(2) If $m_1 \geqslant m_2$ *and* $M_1 \geqslant M_2$ *and* $(m_1 > m_2$ *or* $M_1 > M_2)$ *then* $(m_1, M_1) \succ (m_2, M_2)$.

(3) $\forall A \in P^*$, $\hat{A} = \{a | (minR(a), maxR(a)) \succcurlyeq (minR(a'), maxR(a')), \forall a' \in A\}$.

Proof: Thanks to Theorem 1, we mainly have to prove that adding condition (2) in the theorem is "equivalent" to adding Property ($\boldsymbol{\beta'}$).

Let $[minR(a), maxR(a)]$ be an action such that $R([minR(a), maxR(a)]) = \{minR(a), maxR(a)\}$.

As in Theorem 1, let us define:

$(m_1, M_1) \succcurlyeq (m_2, M_2)$ if and only if $[m_1, M_1](\geqq)[m_2, M_2]$

We now define in the natural way the associated strict relation:

$(m_1, M_1) \succ (m_2, M_2)$ if and only if $[m_1, M_1](>)[m_2, M_2]$

Let us first prove that adding Property ($\boldsymbol{\beta'}$) implies condition (2).

- Let m_1, m_2, M_1, M_2 be such that $m_1 > m_2$ and $M_1 \geqslant M_2$. Then $[m_1, M_2](>)[m_2, M_2]$ (since $m_1 > m_2$ and Property ($\boldsymbol{\beta'}$) enables us to conclude by adding M_2). Besides $[m_1, M_1](\geqq)[m_1, M_2]$ (since $M_1 \geqslant M_2$ and Property ($\boldsymbol{\beta}$) enables us to conclude by adding m_1). Thus we finally have $[m_1, M_1](>)[m_2, M_2]$ and by definition $(m_1, M_1) \succ (m_2, M_2)$.
- Let m_1, m_2, M_1, M_2 be such that $m_1 \geqslant m_2$ and $M_1 > M_2$. Then $[m_1, M_2](\geqq)[m_2, M_2]$ (since $m_1 \geqslant m_2$ and Property ($\boldsymbol{\beta}$) enables us to conclude by adding M_2). Besides $[m_1, M_1](>)[m_1, M_2]$ (since $M_1 > M_2$ and Property ($\boldsymbol{\beta'}$) enables us to conclude by adding m_1). Thus we finally have $[m_1, M_1](>)[m_2, M_2]$ and by definition $(m_1, M_1) \succ (m_2, M_2)$.

This finishes the proof of the necessity part of the theorem.

Now let us verify Property ($\boldsymbol{\beta'}$). Let a, a' be such that $\forall c \in R(a), c' \in R(a'), c > c'$. And let $c'' \in C$. Let a_1, a'_1 such that $R(a_1) = R(a) \cup \{c''\}$ and $R(a'_1) = R(a') \cup \{c''\}$:

- If $c'' \geqslant maxR(a)$, then $minR(a_1) > minR(a'_1)$ and $maxR(a_1) = maxR(a'_1)$.
- If $minR(a') \geqslant c''$, then $minR(a_1) = minR(a'_1)$ and $maxR(a_1) > maxR(a'_1)$.
- If $maxR(a) > c'' > minR(a')$, then $min(c^*, minR(a_1)) > minR(a'_1)$ and $maxR(a_1) > max(maxR(a'_1), c^*)$.

We can see that in every case, adding a common outcome we are again in the situation where the condition (2) applies. We can then conclude $a_1(>)a'_1$. □

We can see that adding Property ($\boldsymbol{\beta'}$) increases the discriminating power of the admissible criteria. Theorem 1 involves only large comparisons over extremal consequences to design optimal actions. Theorem 2 introduces strict comparisons in condition (2) and then excludes more actions from the optimal set. An action a such that it exists another action a' with $minR(a') \geqslant minR(a)$ and $maxR(a') \geqslant maxR(a)$ can still be in the optimal set of a criterion admitted by Theorem 1, even if $minR(a') > minR(a)$ or $maxR(a') > maxR(a)$. This is no longer possible for a criterion admitted by Theorem 2.

Examples of Optimality Criteria

In this section we introduce some simple decision criteria and investigate whether they meet the requirements of the theorems given in the previous sections and then satisfy the desired properties.

A first criterion is a worst-case one, which is usual in strict uncertainty cases.

Definition 1 (Min criterion) *Consider two actions* a_1, a_2.

$$a_1(\geqq_{min})a_2 \text{ iff } minR(a_1) \geqslant minR(a_2).$$

The Min criterion is in a sense the safest one in case of complete ignorance. It is usually known as Wald criterion (Wald 1950).

Definition 2 (Max criterion) *Consider two actions* a_1, a_2.

$$a_1(\geqq_{min})a_2 \text{ iff } maxR(a_1) \geqslant maxR(a_2).$$

The Max criterion is the dual of the min criterion expressing some kind of optimistic state of mind.

It is easy to check that these two criteria define two total preorders verifying the condition (1) of Theorem 1 and that they satisfy properties $(\boldsymbol{\alpha})$ and $(\boldsymbol{\beta})$.

Proposition 1 *The Min and Max criteria satisfy properties* $(\boldsymbol{\alpha})$ *and* $(\boldsymbol{\beta})$.

Nevertheless, we can see that none of them satisfy Property $(\boldsymbol{\beta'})$ in general. Indeed, let us consider two actions a_1, a_2 such that $minR(a_1) \geqslant minR(a_2)$ and $maxR(a_1) > maxR(a_2)$. It is clear that $a_1(\doteq_{min})a_2$. This excludes the Min criterion from the family drawn by Theorem 2 since condition (2) in this theorem would imply in this case that $a_1(>_{min})a_2$. A similar argument excludes the Max criterion.

Thus, the above two criteria do not satisfy Theorem 2 which seems to have a greater discriminating power than Theorem 1. Besides, although they satisfy Theorem 1, they do it in a trivial way because they take account for only one extremal consequence whereas Theorem 1 suggests that the agent should use a criterion that takes into account both of them. We present a criterion that makes use of this possibility.

The Min criterion can be refined into a lexicographic Minmax, i.e., refining the Min thanks to the maximal consequences.

Given n relations $(\geq_i), i \in [1, n]$, on A, the associated *lexicographic relation* (lex($\geq_1, \cdots, \geq_n$)) is defined by a_1 $(\geq_{lex(\geq_1,\cdots,\geq_n)})$ a_2 iff a_1 $(>_{lex(\geq_1,\cdots,\geq_n)})$ a_2 or a_1 $(\sim_{lex(\geq_1,\cdots,\geq_n)})$ a_2 with:

- a_1 $(>_{lex(\geq_1,\cdots,\geq_n)})$ a_2 iff $\exists j \in [1, n]$ s.t. $\forall i < j$ ($a_1 \geq_i a_2$ and $a_2 \geq_i a_1$) and $a_1 >_j a_2$.
- a_1 $(\sim_{lex(\geq_1,\cdots,\geq_n)})$ a_2 iff $\forall i \in [1, n]$ ($a_1 \geq_i a_2$ and $a_2 \geq_i a_1$).

Definition 3 (Minmax criterion) *Consider two actions* a_1, a_2.

$$a_1(\geqq_{minmax})a_2 \text{ iff } a_1(\geqq_{lex(\geqq_{min},\geqq_{max})})a_2.$$

Proposition 2 *The Minmax criterion satisfies Properties* $(\boldsymbol{\alpha})$, $(\boldsymbol{\beta})$ *and* $(\boldsymbol{\beta'})$.

Proof: We must prove that the conditions (1), (2) and (3) of Theorem 2 are satisfied.

Let us define a total preorder $\succcurlyeq$ on the pairs of consequences based on the Minmax criterion using condition (3) of Theorem 2.

Consider two actions a_1, a_2 such that $minR(a_1) \geqslant minR(a_2)$ and $maxR(a_1) \geqslant maxR(a_2)$. The definition of the lexicographic relation implies then that $a_1(\geqq_{lex(\geqq_{min},\geqq_{max})})a_2$ which means $a_1(\geqq_{minmax})a_2$ and by definition $(minR(a_1), maxR(a_1)) \succcurlyeq (minR(a_2), maxR(a_2))$. This proves condition (1).

To prove condition (2), consider two actions a_1, a_2 such that $(minR(a_1) \geqslant minR(a_2)$ and $maxR(a_1) > maxR(a_2))$ or $(minR(a_1) > minR(a_2)$ and $maxR(a_1) \geqslant maxR(a_2))$. The definition of the lexicographic relation implies then that $a_1(>_{lex(\geqq_{min},\geqq_{max})})a_2$ which means $a_1(>)a_2$ and by definition $(minR(a_1), maxR(a_1)) \succ (minR(a_2), maxR(a_2))$. This proves condition (2). □

Conclusion

When trying to describe the minimal information an agent needs to make a sensible decision, the set consequences of an action seems to be an appropriate candidate. Knowing only this information seems to best encapsulate the idea of ignorance. The representation of an action needs to match directly an action to its consequences to be coherent with this structure of information. One can imagine that it is a structure too poor for decision making. We have shown here that even when the traditional mapping state-consequence to represent an action is given up, we can still characterize a family of criteria that have a suitable form.

We have stated some properties that such an optimality criterion should have. These properties express the minimal requirements about decision making under complete ignorance. We have shown that the family of criteria delimited by these properties take account for only the extremal consequences of actions. We have presented some of them and shown how they satisfy the desired properties.

The next step of our study will be to follow the steps of (Maskin 1979) by first listing some further properties an agent may require a decision criterion to meet and then combining them with the already presented properties to characterize other decision criteria.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. They have been partly supported by the ANR project PHAC (ANR-05-BLAN-0384).

References

Arrow, K. J., and Hurwicz, L. 1977. An optimality criterion for decision-making under ignorance. In *Studies in Resource Allocation Processes*, 463–471. Cambridge University Press.

Boutilier, C. 1994. Toward a logic for qualitative decision theory. In *KR*, 75–86.

Bouyssou, D.; Dubois, D.; Pirlot, M.; and Prade, H., eds. 2006. *Concepts et méthodes pour l'aide à la décision - risque et incertain*, volume 2 of *Traité IC2*. http://www.editions-hermes.fr/: Lavoisier.

Dubois, D.; Fargier, H.; Prade, H.; and Perny, P. 2002. Qualitative Decision Theory: From Savage's Axioms to Nonmonotonic Reasoning . *Journal of the ACM* 49(4):455–495.

Jaynes, E. T. 2003. Probability theory: The logic of science. In *Studies in Resource Allocation Processes*. Cambridge University Press.

Lovejoy, W. 1991. A survey of algorithmic methods for partially observed markov decision processes. *Ann. Oper. Res.* 28(1-4):47–66.

Maskin, E. 1979. Decision making under ignorance with implications for social choice. *Theory and Decision* 319–337.

Savage, L. 1954. *The Foundations of Statistics*. New York: J. Wiley. second revised edition, 1972.

von Neumann, J., and Morgenstern, O. 1947. *Theory of games and economic behaviour*. Princeton: Princeton University Press. 2nd edition.

Wald, A. 1950. *Statistical Decision Functions*. John Wiley.