Tomi Janhunen
Ilkka Niemelä (Eds.)

LNAI 6341

Logics in Artificial Intelligence

12th European Conference, JELIA 2010
Helsinki, Finland, September 2010
Proceedings

Lecture Notes in Artificial Intelligence 6341

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Tomi Janhunen Ilkka Niemelä (Eds.)

Logics in Artificial Intelligence

12th European Conference, JELIA 2010
Helsinki, Finland, September 13-15, 2010
Proceedings

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Tomi Janhunen
Aalto University
Department of Information and Computer Science
PO Box 15400
FI-00076 AALTO, Finland
E-mail: Tomi.Janhunen@tkk.fi

Ilkka Niemelä
Aalto University
Department of Information and Computer Science
PO Box 15400
FI-00076 AALTO, Finland
E-mail: Ilkka.Niemela@tkk.fi

Library of Congress Control Number: 2010933610

CR Subject Classification (1998): I.2, F.4.1, H.3, H.4, F.3, D.1.6

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743

ISBN-10 3-642-15674-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15674-8 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2010
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains the papers selected for presentation at the 12th European Conference on Logics in Artificial Intelligence, JELIA 2010, which was held in Helsinki, Finland, during September 13–15, 2010.

Logics provide a formal basis and key descriptive notation for the study and development of applications and systems in artificial intelligence (AI). With the depth and maturity of formalisms, methodologies, and systems today, such logics are increasingly important. The European Conference on Logics in Artificial Intelligence (or Journées Européennes sur la Logique en Intelligence Artificielle — JELIA) began back in 1988, as a workshop, in response to the need for a European forum for the discussion of emerging work in this field. Since then, JELIA has been organized biennially, with English as the official language, and with proceedings published in Springer's *Lecture Notes in Artificial Intelligence* series. In 2010 the conference was organized for the first time in Scandinavia, following previous meetings mainly taking place in Central and Southern Europe. The increasing interest in this forum, its international level with growing participation by researchers worldwide, and the overall technical quality has turned JELIA into a major biennial forum for the discussion of logic-based AI.

The JELIA 2010 Program Committee received a total of 91 submissions comprising 78 regular papers and 13 system descriptions with authors from 30 countries. Each submission was evaluated by at least three expert reviewers followed by a Program Committee discussion on the merits of the paper. The review process was very selective and as a result the Program Committee selected 26 regular papers (33% of submissions) and five system descriptions (38% of submissions) for presentation and inclusion in the proceedings. In addition to the selected papers in the proceedings, the highlights of the JELIA 2010 program included invited talks by Gerhard Brewka titled "Nonmonotonic Tools for Argumentation," by Adnan Darwiche titled "Relax, Compensate and then Recover: A Theory of Anytime, Approximate Inference," and by Stéphane Demri titled "Counter Systems for Data Logics." Moreover, the 5th European Workshop on Probabilistic Graphical Models (PGM 2010) was co-located with JELIA 2010 and the invited talk of Adnan Darwiche was given jointly to the JELIA and PGM audience. We were inspired by the breadth of topics covered in the conference and we are confident that this volume will provide a noteworthy reference to current research issues in logics in AI.

The JELIA 2010 conference was sponsored by the Finnish Cultural Foundation, the Federation of Finnish Learned Societies, University of Helsinki, and Aalto University. We greatly appreciate their generous support and, in particular, the possibility of having the main building of the University of Helsinki in the historical city center as the conference venue. Many individuals also contributed to the success of the conference, to whom we hereby extend our gratitude and

thanks. We are indebted to the members of the JELIA Steering Committee for selecting Helsinki for the JELIA 2010 event. Program Committee members and several other external referees provided timely and in-depth reviews of the submitted papers, and worked hard to select the best papers for presentation at the conference. The Local Chair, Matti Järvisalo, Publicity Chair, Emilia Oikarinen, Volunteers Chair, Mikko Koivisto, and members of the Organizing Committee contributed their invaluable assistance in arranging and hosting the conference. However, it is evident that the broad logics in AI community contributed the most to the success of JELIA 2010 by submitting excellent papers to the conference. Last but not least, we thank the developers of the EasyChair conference management system, which made our job definitely easier.

September 2010

Tomi Janhunen
Ilkka Niemelä

Organization

JELIA 2010 was organized in collaboration with the Department of Information and Computer Science of Aalto University and Department of Computer Science of the University of Helsinki.

Organizing Committee

Satu Eloranta	University of Helsinki, Finland
Raul Hakli	University of Helsinki, Finland
Tomi Janhunen	Aalto University, Finland
Matti Järvisalo	*Local Chair*, University of Helsinki, Finland
Tommi Junttila	Aalto University, Finland
Mikko Koivisto	*Volunteers Chair*, University of Helsinki, Finland
Ilkka Niemelä	*Chair*, Aalto University, Finland
Emilia Oikarinen	*Publicity Chair*, Aalto University, Finland

Program Committee

José Júlio Alferes	Universidade Nova de Lisboa, Portugal
Franz Baader	Technische Universität Dresden, Germany
Peter Baumgartner	NICTA, Australia
Salem Benferhat	Université d'Artois, France
Philippe Besnard	Université Paul Sabatier, France
Piero Bonatti	Università di Napoli Federico II, Italy
Gerhard Brewka	University of Leipzig, Germany
Pedro Cabalar	University of Coruña, Spain
Mehdi Dastani	Utrecht University, The Netherlands
James Delgrande	Simon Fraser University, Canada
Marc Denecker	Katholieke Universiteit Leuven, Belgium
Ulle Endriss	University of Amsterdam, The Netherlands
Esra Erdem	Sabanci University, Turkey
Wolfgang Faber	University of Calabria, Italy
Michael Fisher	University of Liverpool, UK
Lluís Godo	CSIC, Spain
Rajeev Goré	Australian National University, Australia
Andreas Herzig	Université Paul Sabatier, France
Tomi Janhunen	*Co-chair*, Aalto University, Finland
Tommi Junttila	Aalto University, Finland
Joohyung Lee	Arizona State University, USA
Nicola Leone	University of Calabria, Italy

Thomas Lukasiewicz	Technische Universität Wien, Austria
Carsten Lutz	Universität Bremen, Germany
Luís Moniz Pereira	Universidade Nova de Lisboa, Portugal
Angelo Montanari	University of Udine, Italy
Ilkka Niemelä	*Co-chair*, Aalto University, Finland
David Pearce	Universidad Politécnica de Madrid, Spain
Axel Polleres	National University of Ireland, Ireland
Henri Prade	Université Paul Sabatier, France
Jussi Rintanen	NICTA, Australia
Francesca Rossi	University of Padova, Italy
Chiaki Sakama	Wakayama University, Japan
Renate Schmidt	University of Manchester, UK
Ján Šefránek	Comenius University, Slovakia
Terry Swift	SUNY Stony Brook, USA
Michael Thielscher	University of New South Wales, Australia
Hans Tompits	Technische Universität Wien, Austria
Francesca Toni	Imperial College, UK
Wiebe van der Hoek	University of Liverpool, UK
Peter Vojtáš	Charles University, Czech Republic
Toby Walsh	University of New South Wales, Australia
Frank Wolter	University of Liverpool, UK

Additional Referees

Salvador Abreu	Antti Hyvärinen	Emilia Oikarinen
Stéphane Airiau	Giovambattista Ianni	Ravi Palla
Marco Alberti	Katsumi Inoue	Dirk Pattinson
Mario Alviano	Matti Järvisalo	Rafael Peñaloza
Martin Baláž	Reinhard Kahle	Daniele Porello
Annamaria Bria	Matthias Knorr	Jörg Pührer
Domenico Corapi	Boris Konev	Hilverd Reker
Broes De Cat	Roman Kontchakov	Francesco Ricca
Stef De Pooter	Thomas Krennwallner	Fabrizio Riguzzi
Pierangelo Dell'Acqua	João Leite	Riccardo Rosati
Dario Della Monica	Nuno Lopes	Sebastian Rudolph
Agostino Dovier	Michael Maher	Pietro Sala
Jori Dubrovin	Marco Maratea	Luigi Sauro
Halit Erdogan	Enrico Marchioni	Stephan Scheele
Marco Faella	Yunsong Meng	Thomas Schneider
Michael Fink	Thomas Meyer	Stefan Schulz
Alfredo Gabaldon	Sanjay Modgil	Jozef Šiška
Ana Sofia Gomes	Marco Montali	Christoph Sticksel
Ricardo Gonçalves	Peter Novák	Giorgio Terracina
Gianluigi Greco	Philipp Obermeier	Dmitry Tishkovsky
Davide Grossi	Johannes Oetsch	Paolo Turrini

Levan Uridia
Petko Valtchev
Agustín Valverde
Joost Vennekens
Hanne Vlaeminck
Dirk Walther
Heinrich Wansing
Gregory Wheeler
Johan Wittocx
Stefan Woltran
Michael Zakharyaschev
Antoine Zimmermann

Table of Contents

I Invited Talks

II Regular Papers

III System Descriptions

Nonmonotonic Tools for Argumentation

Gerhard Brewka

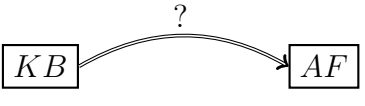
Universität Leipzig, Augustusplatz 10-11, 04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de

Abstract. Dung's argumentation frameworks (AFs) have become very popular as semantical tools in argumentation. We discuss a generalization of AFs called abstract dialectical frameworks (ADFs). These frameworks are more flexible in that they allow arbitrary boolean functions to be used for the specification of acceptance conditions for nodes. We present the basic underlying definitions and give an example illustrating why they are useful. More precisely, we show how they can be used to provide a semantical foundation for Gordon, Prakken and Walton's Carneades model of argumentation, lifting the limitation of this model to acyclic argument graphs.

1 Introduction

Dung's abstract argumentation frameworks (AFs) [4] are without doubt the most influential tools currently used in argumentation. They provide basic conflict handling mechanisms for argumentation: whenever all relevant arguments and the conflict relations among them have been established, the semantics defined for AFs specify different ways of identifying reasonable subsets of the arguments which are jointly acceptable.

AFs have become extremely popular in argumentation. They are commonly used in the following way: given a knowledge base, say consisting of defeasible rules, preferences, proof standards, etc., the available information is first compiled into adequate arguments and attacks. The resulting AF then provides the system with a choice of different semantics. The following picture illustrates this:



AFs are among the simplest nonmonotonic systems one can think of - and this is certainly part of why they are so popular. Still, we believe - and will demonstrate here - that adding further functionality to AFs may be worthwhile as this can bring the target systems of the compilation described above closer to what one typically finds in the original knowledge bases.

This was one of the reasons for Brewka and Woltran to introduce abstract dialectical frameworks (ADFs) [3]. ADFs are a powerful generalization of Dung-style argumentation frameworks. Dung argumentation frameworks have an implicit, fixed criterion for the acceptance of a node in the argument graph: a

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 1–6, 2010.
© Springer-Verlag Berlin Heidelberg 2010

node is accepted iff all its parents are defeated. This acceptance criterion can be viewed as an implicit boolean function assigning a status to an argument based on the status of its parents. The basic idea underlying ADFs is to make this boolean function explicit, and then to allow arbitrary acceptance conditions for nodes to be specified. In a nutshell, this turns the "calculus of opposition" provided by AFs into a "calculus of support an opposition".

It turns out that the standard semantics for Dung frameworks - grounded, preferred and stable - can be generalized to ADFs, the latter two to a slightly restricted class of ADFs called bipolar, where each link in the graph either supports or attacks its target node. Since all ADFs we are dealing with here are bipolar, we will simply speak of ADFs and omit the adjective "bipolar" whenever there is no risk of confusion.

In the next section we will briefly introduce the main definitions underlying ADFs. We will then illustrate why they are useful, showing how Carneades argument evaluation structures [6,7] can be reconstructed as ADFs.

2 Abstract Dialectical Frameworks

An ADF [3] is a directed graph whose nodes represent arguments or statements which can be accepted or not. The links represent dependencies: the status of a node s only depends on the status of its parents (denoted $par(s)$), that is, the nodes with a direct link to s. In addition, each node s has an associated acceptance condition C_s specifying the conditions under which s is accepted. This is where ADFs go beyond Dung argumentation frameworks. C_s is a boolean function yielding for each assignment of values to $par(s)$ one of the values *in*, *out* for s. As usual, we will identify value assignments with the sets of nodes which are *in*. Thus, if for some $R \subseteq par(s)$ we have $C_s(R) = in$, then s will be accepted provided the nodes in R are accepted and those in $par(s) \setminus R$ are not accepted.

Definition 1. *An* abstract dialectical framework *is a tuple* $D = (S, L, C)$ *where*

- S *is a set of statements,*
- $L \subseteq S \times S$ *is a set of links,*
- $C = \{C_s\}_{s \in S}$ *is a set of total functions* $C_s : 2^{par(s)} \rightarrow \{in, out\}$*, one for each statement* s*.* C_s *is called acceptance condition of* s*.*

S and L obviously form a graph, and we sometimes refer to elements of S as nodes. For the purposes of this paper we will only deal with a subset of ADFs, called bipolar in [3]. In such ADFs each link is either attacking or supporting:

Definition 2. *Let* $D = (S, L, C)$ *be an ADF. A link* $(r, s) \in L$ *is*

1. supporting *iff, for no* $R \subseteq par(s)$, $C_s(R) = in$ *and* $C_s(R \cup \{r\}) = out$,
2. attacking *iff, for no* $R \subseteq par(s)$, $C_s(R) = out$ *and* $C_s(R \cup \{r\}) = in$.

For simplicity we will only speak of ADFs here, keeping in mind that all ADFs in this paper are indeed bipolar.

It turns out that Dung's standard semantics - grounded, stable, preferred - can be generalized adequately to ADFs. We first introduce the notion of a model. Intuitively, in a model all acceptance conditions are satisfied.

Definition 3. *Let $D = (S, L, C)$ be an ADF. $M \subseteq S$ is a* model *of D if for all $s \in S$ we have $s \in M$ iff $C_s(M \cap par(s)) = in$.*

We first define the generalization of grounded semantics:

Definition 4. *Let $D = (S, L, C)$ be an ADF. Consider the operator*

$$\Gamma_D(A, R) = (acc(A, R), reb(A, R))$$

where

$$acc(A, R) = \{r \in S \mid A \subseteq S' \subseteq (S \setminus R) \Rightarrow C_r(S' \cap par(r)) = in\}, \text{ and}$$
$$reb(A, R) = \{r \in S \mid A \subseteq S' \subseteq (S \setminus R) \Rightarrow C_r(S' \cap par(r)) = out\}.$$

Γ_D is monotonic in both arguments and thus has a least fixpoint. E is the well-founded model *of D iff for some $E' \subseteq S$, (E, E') is the least fixpoint of Γ_D.*

For stable models we apply a construction similar to the Gelfond/Lifschitz reduct for logic programs. The purpose of the reduction is to eliminate models in which nodes are *in* just because of self supporting cycles:

Definition 5. *Let $D = (S, L, C)$ be an ADF. A model M of D is a* stable model *if M is the least model of the reduced ADF D^M obtained from D by*

1. *eliminating all nodes not contained in M together with all links in which any of these nodes appear,*
2. *eliminating all attacking links,*
3. *restricting the acceptance conditions C_s for each remaining node s to the remaining parents of s.*

Preferred extensions in Dung's approach are maximal admissible sets, where an admissible set is conflict-free and defends itself against attackers. This can be rephrased as follows: E is *admissible* in a Dung argumentation framework $A = (AR, att)$ iff for some $R \subseteq AR$

- R does not attack E, and
- E is a stable extension of $(AR\text{-}R, att \cap (AR\text{-}R \times AR\text{-}R))$.

This leads to the following generalization:

Definition 6. *Let $D = (S, L, C)$, $R \subseteq S$. D-R is the ADF obtained from D by*

1. *deleting all nodes in R together with their acceptance conditions and links they are contained in.*

2. *restricting acceptance conditions of the remaining nodes to the remaining parents.*

Definition 7. *Let $D = (S, L, C)$ be an ADF. $M \subseteq S$ is* admissible *in D iff there is $R \subseteq S$ such that*

1. *no element in R attacks an element in M, and*
2. *M is a stable model of D-R.*

M is a preferred *model of D iff M is (subset) maximal among the sets admissible in D.*

Brewka and Woltran also introduced weighted ADFs where an additional weight function w assigns qualitative or numerical weights to the links in the graph. This allows acceptance conditions to be defined in a domain independent way, based on the weights of links rather than on the involved statements. They also showed how the proof standards proposed by Farley and Freeman [5] can be formalized based on this idea.

The reader is referred to [3] for further details.

3 Application: Reconstructing Carneades

The Carneades model of argumentation, introduced by Gordon, Prakken and Walton in [6] and developed further in a series of subsequent papers [7,1,8], is an advanced general framework for argumentation.[1] It captures both static aspects, related to the evaluation of arguments in a particular context based on proof standards for statements and on weights arguments are given by an audience, and dynamic aspects, covering for instance the shift of proof burdens in different stages of the argumentation process.

Unlike many other approaches, Carneades does not rely on Dung's argumentation frameworks (AFs) [4] for the definition of its semantics, more specifically its notion of acceptable statements. One goal of our reconstruction is to provide a link, albeit an indirect one, between Carneades and AFs. As we will see, both are instances of a more general framework. Moreover, in spite of this generality, Carneades suffers from a restriction: it is assumed that the graphs formed by arguments are acyclic. This is not as bad as it may first sound, as the use of pro and con arguments allows some conflicts to be represented which require cyclic representations in other frameworks. Still, cycles in argumentation appear so common that forbidding them right from the start is certainly somewhat problematic. And indeed, the authors in [6] write (page 882):

> *"We ... leave an extension to graphs that allow for cycles through exceptions for future work."*

[1] As of June 2010, [6] is among the 10 most cited papers which appeared in the Artificial Intelligence Journal over the last 5 years.

Indeed, by reconstructing Carneades argument evaluation structures as ADFs, the mentioned limitation can be overcome.

We cannot go into the technical details of the translation here and refer the reader to [2]. Nevertheless, we want to give a basic idea how the translation works. We start with the definition of arguments in Carneades [7]:

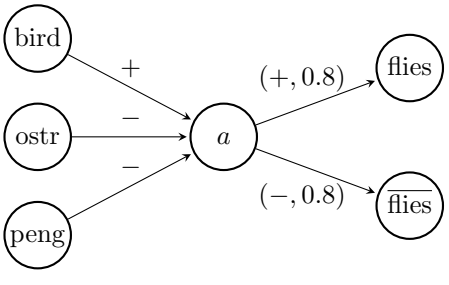
Definition 8 (argument). *Let $\mathcal{L}$ be a propositional language. An **argument** is a tuple $\langle P, E, c\rangle$ where $P \subset \mathcal{L}$ are its **premises**, $E \subset \mathcal{L}$ with $P \cap E = \emptyset$ are its **exceptions** and $c \in \mathcal{L}$ is its **conclusion**. For simplicity, c and all members of P and E must be literals, i.e. either an atomic proposition or a negated atomic proposition. Let p be a literal. If p is c, then the argument is an argument **pro** p. If p is the complement of c, then the argument is an argument **con** p.*

An argument evaluation structure was defined in [7] as a triple consisting of a stage, an audience, and a function assigning a proof standard to propositions. Since we are only interested in stage specific argument evaluation, the status part of the definition of stages (see [7]) can be skipped, keeping only the set of arguments, together with the audience (a pair consisting of a set of assumptions and a weight function) and the proof standards.

To illustrate our translation, consider the argument

$$a = \langle \{bird\}, \{peng, ostr\}, flies \rangle$$

and assume $weights(a) = 0.8$. The ADF graph generated by this argument is shown in the following figure (we mark links with their weights):



Proof standards and assumptions can be captured by adequate acceptance conditions for the nodes in the ADF. It was proven in [2] that this translation yields the desired results, that is, the acceptable statements in Carneades coincide with the statement nodes assigned *in* in the generated ADF.

The real advantage of our translation is that we can now lift the restriction of acyclicity. Nothing in the translation hinges on the fact that the set of Carneades arguments is acyclic. Indeed, cycles in the set of Carneades arguments will lead to cycles in the ADF, yet these cycles are handled - in different ways - by the available semantics of ADFs.

These results are of interest, both from the point of view of ADFs and from the point of view of Carneades:

1. They show that ADFs not only generalize Dung argumentation frameworks - which have been the starting point for their development. They also generalize Carneades argument evaluation structures.
2. They clarify the relationship between Carneades and Dung AFs, showing that both are instances of ADFs. They thus help to put Carneades on an equally solid formal foundation.
3. Finally, they allow us to lift the restriction of Carneades to acyclic argument structures.

As we believe, this provides sufficient evidence that the ADF framework is indeed a useful nonmonotonic tool in the theory of argumentation.

References

1. Ballnat, S., Gordon, T. : Goal selection in argumentation processes. In: Proc. Computational Models of Argumentation (2010)
2. Brewka, G., Gordon, T.F.: Carneades and abstract dialectical frameworks: A reconstruction. In: Proc. COMMA (2010)
3. Brewka, G., Woltran, S.: Abstract dialectical frameworks. In: Proc. Principles of Knowledge Representation and Reasoning, pp. 102–111 (2010)
4. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995)
5. Farley, A.M., Freeman, K.: Burden of proof in legal argumentation. In: Proc. ICAIL 1995, pp. 156–164 (1995)
6. Gordon, T.F., Prakken, H., Walton, D.: The Carneades model of argument and burden of proof. Artif. Intell. 171(10-15), 875–896 (2007)
7. Gordon, T.F., Walton, D.: Proof burdens and standards. In: Rahwan, I., Simari, G. (eds.) Argumentation in Artificial Intelligence, pp. 239–258 (2009)
8. Grabmair, M., Gordon, T., Walton, D.: Probabilistic semantics for the Carneades argument model using Bayesian belief networks. In: Proc. Computational Models of Argumentation (2010)

Relax, Compensate and Then Recover: A Theory of Anytime, Approximate Inference

Adnan Darwiche

Computer Science Department,
University of California, Los Angeles, USA
darwiche@cs.ucla.edu

This talk is based on two main ideas, one concerning exact probabilistic inference and the second concerning approximate probabilistic inference. Both ideas have their roots in symbolic inference and do complement each other.

The first idea shows how one can reduce exact probabilistic inference to a problem of knowledge compilation: transforming propositional knowledge bases so they attain certain syntactic properties [8]. In particular, I will discuss two syntactic properties of propositional knowledge bases, called decomposability and determinism, and show that an ability to enforce these two properties efficiently leads to an ability to efficiently do inference on probabilistic graphical models. This connection is not recent — see [7] for one of the first formulations. Yet, it is important to highlight as it helps in showing the relevance of work on symbolic knowledge compilation to probabilistic inference. I will in particular highlight some of the open problems and computational bottlenecks in this area, in addition to recent advances in this direction (e.g., [11]). My goal here is to motivate further work on knowledge compilation by the symbolic logic reasoning community.

In logical terms, decomposability is about expressing an event γ as a conjunction, say, $\alpha \wedge \beta$, where the conjuncts do not share variables. By decomposing γ in this fashion, we will able to decompose computations on γ into independent computations on α and on β. Sometimes, we cannot perform this decomposition, especially when the variables of α and β are already predetermined. The solution to this problem is to express γ as a disjunction $\alpha_1 \wedge \beta_1 \vee \ldots \vee \alpha_n \wedge \beta_n$, where each disjunct $\alpha_i \wedge \beta_i$ is a decomposition. This is always possible, but one would clearly want to minimize the size of such disjunctions. One may not be able to escape exponential growth in some cases, however, especially that each α_i and β_i would generally need to be decomposed recursively as well. Moreover, when the ultimate goal is to perform probabilistic reasoning, one would want the disjuncts to be mutually exclusive as well. That is, no pair of disjuncts can be satisfied by the same model. This is the property of determinism.

Another, but approximate, method to enforce decomposability is based on relaxations. Suppose for example that the conjuncts α and β share a single variable X. One can simply pretend that the occurrences of X in α are distinct from those in β. One way to realize this is by renaming every occurrence of X in β to X'. The conjunction $\alpha \wedge \beta$ is now decomposable, but clearly not equivalent to the original conjunction. Note, however, that if we add the equivalence constraint $X \equiv X'$ to this new conjunction, we would obtain a conjunction that is

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 7–9, 2010.
© Springer-Verlag Berlin Heidelberg 2010

equivalent to the original one. Hence, our approximate decomposition technique can be viewed as one of relaxing equivalence constraints. In its simplest form, this technique has been the basis for some MAXSAT solvers [12,13]. It is also the basis for some well known approximation methods in probabilistic reasoning, such as minibuckets [9], as shown in [1]. Interestingly enough, one can provide specific guarantees on the results obtained from such relaxations, typically in the form of upper/lower bounds. These bounds are used as final approximations, as in minibuckets, or form the basis of pruning in branch-and-bound search algorithms, as in some MAXSAT solvers.

A more refined version of the above technique is to compensate for the relaxed equivalence constraint. Such compensations, however, requires one to work with a more refined form of logical knowledge bases. An example of this is MAXSAT, where one is allowed to attach weights to logical constraints. Another example is probabilistic reasoning, where one can attach probabilities to such constraints. By utilizing these numerical tools, and adjusting them carefully, one can enforce some weaker notions of equivalence — for example, ensuring that variables X and its clone X' have the same probability. A number of such weaker notions have been proposed recently for both MAXSAT [6] and probabilistic reasoning [2,5]. In fact, the influential algorithm of loopy belief propagation [10,14] can be formulated in terms of relaxing equivalence constraints and a specific compensation method, as shown in [2,3].

One of the key questions with regards to this approximation scheme concerns the specific equivalence constraints to relax and then compensate for. The main problem here is that a careful method for making such decisions would have to perform inference, which is not feasible in the first place (otherwise, we would not have a need to relax equivalences). Another approach is to relax too many equivalence constraints, therefore, allowing one to perform approximate inference, and then use the resulting approximations to decide which equivalence constraints to recover. This is the approach advocated in [2] and the one we shall discuss in this talk. A number of "recovery heuristics" have been proposed for this purpose with varying computational overhead and effectiveness [2,4]. Ideally, one would recover equivalence constraints one at a time, until the problem becomes too difficult computationally. This would lead to a refined anytime behavior and allow one to recover each equivalence constraint based on more accurate approximations. It may also be inefficient, however, as it would require too much "thinking" during the recovery phase, therefore, calling for a coarser recovery scheme in certain situations. I will present some empirical results on different recovery strategies and discuss a particular one that was used successfully in the third UAI challenge on approximate inference, which took place in July 2010.

Acknowledgments. The results reported in this talk are mostly based on joint work with Arthur Choi, who also contributed to the preparations behind the talk itself.

References

1. Choi, A., Chavira, M., Darwiche, A.: Node splitting: A scheme for generating upper bounds in bayesian networks. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI), pp. 57–66 (2007)
2. Choi, A., Darwiche, A.: An edge deletion semantics for belief propagation and its practical impact on approximation quality. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), pp. 1107–1114 (2006)
3. Choi, A., Darwiche, A.: Approximating the partition function by deleting and then correcting for model edges. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 79–87 (2008)
4. Choi, A., Darwiche, A.: Focusing generalizations of belief propagation on targeted queries. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI), pp. 1024–1030 (2008)
5. Choi, A., Darwiche, A.: Relax then compensate: On max-product belief propagation and more. In: Proceedings of the Twenty-Third Annual Conference on Neural Information Processing Systems (NIPS), pp. 351–359 (2009)
6. Choi, A., Standley, T., Darwiche, A.: Approximating weighted max-sat problems by compensating for relaxations. In: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP), pp. 211–225 (2009)
7. Darwiche, A.: A logical approach to factoring belief networks. In: Proceedings of KR, pp. 409–420 (2002)
8. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research 17, 229–264 (2002)
9. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM 50(2), 107–153 (2003)
10. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers, Inc., San Mateo (1988)
11. Pipatsrisawat, K., Darwiche, A.: A lower bound on the size of decomposable negation normal form. In: Proceedings of the 25th National Conference on Artificial Intelligence, AAAI (2010)
12. Pipatsrisawat, K., Palyan, A., Chavira, M., Choi, A., Darwiche, A.: Solving weighted max-sat problems in a reduced search space: A performance analysis. Journal on Satisfiability Boolean Modeling and Computation (JSAT) 4, 191–217 (2008)
13. Ramírez, M., Geffner, H.: Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In: CP, pp. 605–619 (2007)
14. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Understanding belief propagation and its generalizations. In: Lakemeyer, G., Nebel, B. (eds.) Exploring Artificial Intelligence in the New Millennium, ch. 8, pp. 239–269. Morgan Kaufmann, San Francisco (2003)

Counter Systems for Data Logics

Stéphane Demri

LSV, CNRS, ENS de Cachan, INRIA Saclay IdF, France

Abstract. Data logics are logical formalisms that are used to specify properties on structures equipped with data (data words, data trees, runs from counter systems, timed words, etc.). In this survey talk, we shall see how satisfiability problems for such data logics are related to reachability problems for counter systems (including counter automata with errors, vector addition systems with states, etc.). This is the opportunity to provide an overview about the relationships between data logics and verification problems for counter systems.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, p. 10, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Similarity-Based Inconsistency-Tolerant Logics

Ofer Arieli[1] and Anna Zamansky[2]

[1] Department of Computer Science, The Academic College of Tel-Aviv, Israel
oarieli@mta.ac.il
[2] Department of Software Engineering, Jerusalem College of Engineering, Israel
annaza@jce.ac.il

Abstract. Many logics for AI applications that are defined by denotational semantics are trivialized in the presence of inconsistency. It is therefore often desirable, and practically useful, to refine such logics in a way that inconsistency does not cause the derivation of any formula, and, at the same time, inferences with respect to consistent premises are not affected. In this paper, we introduce a general method of doing so by incorporating preference relations defined in terms of similarities. We exemplify our method for three of the most common denotational semantics (standard many-valued matrices, their non-deterministic generalization, and possible worlds semantics), and demonstrate their usefulness for reasoning with inconsistency.

1 Introduction

Logics based on denotational semantics have many attractive properties for AI applications. However, most of the standard logics that are defined this way, including classical logic, intuitionistic logic, and some modal logics, are not inconsistency-tolerant, in the sense that they are trivialized for inconsistent theories: whenever the set of premises is not satisfiable, anything follows from it.[1] This renders such logics practically useless for reasoning with inconsistency.

In this paper, we introduce a general framework for adding inconsistency-maintenance capabilities to a wide range of logics that are defined by denotational semantics, without affecting their inferences with respect to consistent premises. More specifically, an *inconsistency-tolerant* variant of a logic L is a logic that is faithful to L with respect to consistent theories, but does not "explode" in the presence of inconsistency. For this, we incorporate the well-known preferential semantics of Shoham [13], in which for drawing conclusions from a set of premises, one takes into account its "most preferred" (or "plausible") valuations (rather than all of its models, none of which exists in case of contradictions).

Preferential semantics yields non-monotonic logics that often tolerate inconsistency in a proper, non-trivial way. However, in general this method does not guarantee faithfulness to the original logic (L) with respect to consistent theories. To achieve this, we consider a particular kind of preference criteria that are

[1] For languages with a negation $\neg$, this usually means that the underlying logic is not *paraconsistent* [6]: any formula ϕ follows from $\{\psi, \neg\psi\}$.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 11–23, 2010.
© Springer-Verlag Berlin Heidelberg 2010

based on the quantitative notion of *similarity*. Intuitively, similarities measure to what extent each valuation is "similar" to some model of a given theory, or how "close" each valuation is to satisfying the theory. This notion, which is more general than the notion of a distance, allows us to generalize many revision and merging operators considered in the literature for handling contradictory data.

We exemplify our similarity-based method on three of the most common types of denotational semantics, and demonstrate their usefulness by some concrete examples of reasoning with inconsistency.

2 Preliminaries

2.1 Denotational Semantics

In the sequel, $\mathcal{L}$ denotes a propositional language with a set Atoms of atomic formulas and a set $\mathcal{F}_\mathcal{L}$ of well-formed formulas. We denote the elements of Atoms by p, q, r, and the elements of $\mathcal{F}_\mathcal{L}$ by ψ, ϕ, σ. A theory Γ is a finite set of formulas in $\mathcal{F}_\mathcal{L}$. The atoms appearing in the formulas of Γ and the subformulas of Γ are denoted, respectively, $\mathsf{Atoms}(\Gamma)$ and $\mathsf{SF}(\Gamma)$. The set of all theories of $\mathcal{L}$ is $\mathcal{T}_\mathcal{L}$.

Definition 1. Given a language $\mathcal{L}$, a *propositional logic* for $\mathcal{L}$ is a pair $\langle \mathcal{L}, \vdash \rangle$, where $\vdash$ is a (Tarskian) consequence relation for $\mathcal{L}$, i.e., a binary relation satisfying the following conditions:

Reflexivity: if $\psi \in \Gamma$ then $\Gamma \vdash \psi$.
Monotonicity: if $\Gamma \vdash \psi$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \vdash \psi$.
Transitivity: if $\Gamma \vdash \psi$ and $\Gamma', \psi \vdash \varphi$ then $\Gamma, \Gamma' \vdash \varphi$.

A common (model-theoretical) way of defining consequence relations for $\mathcal{L}$ is based on *denotational semantics*:

Definition 2. A *denotational semantics* for a language $\mathcal{L}$ is a pair $\mathsf{S} = \langle S, \models_S \rangle$, where S is a nonempty set (of 'interpretations'), and $\models_S$ (the 'satisfiability relation' of S) is a binary relation on $S \times \mathcal{F}_\mathcal{L}$.

Let $\nu \in S$ and $\psi \in \mathcal{F}_\mathcal{L}$. If $\nu \models_S \psi$, we say that ν *satisfies* ψ and call ν an S-*model* of ψ. The set of the S-models of ψ is denoted by $mod_\mathsf{S}(\psi)$. If ν satisfies every formulas ψ in a theory Γ, it is called an S-model of Γ. The set of the S-models of Γ is denoted by $mod_\mathsf{S}(\Gamma)$. If $mod_\mathsf{S}(\Gamma) \neq \emptyset$ we say that Γ is S-*consistent*, otherwise Γ is S-inconsistent.

Below, we shall usually omit the prefix S of the above notions.

A denotational semantics S induces the following relation on $\mathcal{T}_\mathcal{L} \times \mathcal{F}_\mathcal{L}$:

Definition 3. We denote by $\Gamma \vdash_\mathsf{S} \psi$ that $mod_\mathsf{S}(\Gamma) \subseteq mod_\mathsf{S}(\psi)$.

Proposition 1. *Let* $\mathsf{S} = \langle S, \models_S \rangle$ *be a denotational semantics for* $\mathcal{L}$. *Then* $\langle \mathcal{L}, \vdash_\mathsf{S} \rangle$ *is a propositional logic for* $\mathcal{L}$.[2]

Next, we recall some common cases of denotational semantics and their corresponding logics.

[2] Proposition 1 is well-known and can be easily verified. Due to short of space, in what follows proofs are considerably reduced or omitted altogether.

2.2 Many-Valued Matrices

Definition 4. A *(multi-valued) matrix* for a language $\mathcal{L}$ is a triple $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{V}$ is a non-empty set of truth values, $\mathcal{D}$ is a non-empty proper subset of $\mathcal{V}$, and $\mathcal{O}$ contains an interpretation $\tilde{\diamond} : \mathcal{V}^n \to \mathcal{V}$ for every n-ary connective of $\mathcal{L}$.

Given a matrix $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, we shall assume that $\mathcal{V}$ includes at least the two classical values t and f, and that only the former belongs to the set $\mathcal{D}$ of the *designated elements* in $\mathcal{V}$ (those that represent 'true assertions'). The set $\mathcal{O}$ contains the interpretations (the 'truth tables') of each connective in $\mathcal{L}$. The associated semantical notions are now defined as usual.

Definition 5. Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$. An *$\mathcal{M}$-valuation* is a function $\nu : \mathcal{F}_{\mathcal{L}} \to \mathcal{V}$ so that, for every connective $\diamond$ in $\mathcal{L}$, $\nu(\diamond(\psi_1, \ldots, \psi_n)) = \tilde{\diamond}(\nu(\psi_1), \ldots, \nu(\psi_n))$. We shall sometimes denote by $\nu = \{p_1 : x_1, p_2 : x_2, \ldots\}$ the assignments $\nu(p_i) = x_i$, for $i = 1, 2 \ldots$. The set of all $\mathcal{M}$-valuations is denoted by $\Lambda_{\mathcal{M}}$. We say that $\nu \in \Lambda_{\mathcal{M}}$ is a *model* of ψ, denoted $\nu \models_{\mathcal{M}} \psi$, if $\nu(\psi) \in \mathcal{D}$.

Note that the pair $\langle \Lambda_{\mathcal{M}}, \models_{\mathcal{M}} \rangle$ is a denotational semantics in the sense of Definition 2. By Proposition 1 we have, then, that:

Proposition 2. *The relation $\vdash_{\mathcal{M}}$, induced from a matrix $\mathcal{M}$ by Definition 3, is a Tarskian consequence relation.*

Example 1. The most common matrix-based entailments are induced from two-valued matrices. Thus, for instance, when $\mathcal{L}$ is the standard propositional language, $\mathcal{V} = \{\mathsf{t}, \mathsf{f}\}$, $\mathcal{D} = \{\mathsf{t}\}$, and $\mathcal{O}$ consists of the standard interpretations of the connectives in $\mathcal{L}$, $\langle \mathcal{L}, \models_{\mathcal{M}} \rangle$ is the classical propositional logic.

Three-valued logics are obtained by adding to $\mathcal{V}$ a third element. For instance, Kleene's logic [9] and McCarthy's logic [11] are obtained, respectively, from the matrices $\mathcal{M}_K^{3\perp} = \langle \{\mathsf{t}, \mathsf{f}, \perp\}, \{\mathsf{t}\}, \mathcal{O}_K \rangle$ and $\mathcal{M}_M^{3\perp} = \langle \{\mathsf{t}, \mathsf{f}, \perp\}, \{\mathsf{t}\}, \mathcal{O}_M \rangle$, in which the disjunction and conjunction are interpreted differently:

(Kleene)

	$\tilde{\neg}$
f	t
$\perp$	$\perp$
t	f

$\tilde{\wedge}$	f	$\perp$	t
f	f	f	f
$\perp$	f	$\perp$	$\perp$
t	f	$\perp$	t

$\tilde{\vee}$	f	$\perp$	t
f	f	$\perp$	t
$\perp$	$\perp$	$\perp$	t
t	t	t	t

(McCarthy)

$\tilde{\wedge}$	f	$\perp$	t
f	f	f	f
$\perp$	$\perp$	$\perp$	$\perp$
t	f	$\perp$	t

$\tilde{\vee}$	f	$\perp$	t
f	f	$\perp$	t
$\perp$	$\perp$	$\perp$	$\perp$
t	t	t	t

Priest's logic LP [12] is similar to Kleene's logic, but the third element is designated, so we denote it by $\top$ rather than $\perp$. This logic is induced by $\mathcal{M}_P^{3\top} = \langle \{\mathsf{t}, \mathsf{f}, \top\}, \{\mathsf{t}, \top\}, \mathcal{O}_P \rangle$, where $\mathcal{O}_P$ is obtained from $\mathcal{O}_K$ by replacing $\perp$ by $\top$.

2.3 Non-deterministic Matrices

Matrix-based semantics is truth-functional in the sense that the truth-value of a complex formula is uniquely determined by the truth-values of its subformulas. Such a semantics cannot be useful in capturing non-deterministic phenomena. This leads to the idea of non-deterministic matrices, introduced in [4], which allows non-deterministic evaluation of formulas:

Definition 6. A *non-deterministic matrix* (*Nmatrix*) for $\mathcal{L}$ is a tuple $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{V}$ is a non-empty set of truth values, $\mathcal{D}$ is a non-empty proper subset of $\mathcal{V}$, and $\mathcal{O}$ contains an interpretation function $\tilde{\diamond} : \mathcal{V}^n \to 2^{\mathcal{V}} \setminus \{\emptyset\}$ for every n-ary connective of $\mathcal{L}$.

An $\mathcal{M}$-*valuation* is a function $\nu : \mathcal{F}_{\mathcal{L}} \to \mathcal{V}$ such that for every connective $\diamond$ in $\mathcal{L}$, $\nu(\diamond(\psi_1, \ldots, \psi_n)) \in \tilde{\diamond}(\nu(\psi_1), \ldots, \nu(\psi_n))$. The set of all $\mathcal{M}$-valuations is denoted by $\Lambda_{\mathcal{M}}$. Again, $\nu \in \Lambda_{\mathcal{M}}$ is a *model* of ψ in $\mathcal{M}$ ($\nu \models_{\mathcal{M}} \psi$), if $\nu(\psi) \in \mathcal{D}$.

Ordinary matrices can be thought of as Nmatrices, the interpretations of which return singletons of truth-values. Henceforth, we shall identify deterministic Nmatrices and the corresponding ordinary matrices. Again, for an Nmatrix $\mathcal{M}$, the pair $\langle \Lambda_{\mathcal{M}}, \models_{\mathcal{M}} \rangle$ is a denotational semantics and it induces a Tarskian consequence relation $\vdash_{\mathcal{M}}$.

Example 2. Consider an interaction with remote computers, where each computation may be either serial or parallel. This can be captured by non-deterministic interpretations, combining Kleene's and McCarthy's logics (Example 1):

$\tilde{\neg}$	
f	{t}
$\bot$	{$\bot$}
t	{f}

$\tilde{\wedge}$	f	$\bot$	t
f	{f}	{f}	{f}
$\bot$	{f, $\bot$}	{$\bot$}	{$\bot$}
t	{f}	{$\bot$}	{t}

$\tilde{\vee}$	f	$\bot$	t
f	{f}	{$\bot$}	{t}
$\bot$	{$\bot$}	{$\bot$}	{t, $\bot$}
t	{t}	{t}	{t}

Nmatrices have important applications in reasoning under uncertainty, proof theory, etc. We refer to [5] for a detailed discussion on Nmatrices.

2.4 Possible-Worlds Semantics

The last type of denotational semantics considered here is based on a many-valued extension of standard Kripke semantics (see [7]), where the logical connectives can be interpreted by a matrix $\mathcal{M}$,[3] and qualifications of the truth of a judgement is expressed by the necessitation operator "$\Box$". In case of the classical two-valued matrix we have the usual Kripke-style semantics.

Definition 7. Let $\mathcal{L}$ be a propositional language.

- A *frame* for $\mathcal{L}$ is a triple $\mathcal{F} = \langle W, R, \mathcal{M} \rangle$, where W is a non-empty set (of "worlds"), R (the "accessibility relation") is a binary relation on W, and $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ is a matrix for $\mathcal{L}$. We say that a frame is finite if so is W.
- Let $\mathcal{F} = \langle W, R, \mathcal{M} \rangle$ be a frame for $\mathcal{L}$. An $\mathcal{F}$-*valuation* is a function $\nu : W \times F_{\mathcal{L}} \to \mathcal{V}$ that assigns truth values to the $\mathcal{L}$-formulas at each world in W according to the following conditions: For every connective $\diamond$ in the language $\mathcal{L}$ (except for $\Box$),
  - $\nu(w, \diamond(\psi_1, \ldots, \psi_n)) = \tilde{\diamond}_{\mathcal{M}}(\nu(w, \psi_1), \ldots, \nu(w, \psi_n))$,
  - $\nu(w, \Box\psi) \in \mathcal{D}$ iff $\nu(w', \psi) \in \mathcal{D}$ for all w' such that $R(w, w')$.

[3] This framework can be extended to Nmatrices as well, but for simplicity we stick to deterministic matrices.

The set of $\mathcal{F}$-valuations is denoted by $\Lambda_{\mathcal{F}}$. The set of $\mathcal{F}$-valuations that satisfy a formula ψ in a world $w \in W$ is $mod^w_{\mathcal{F}}(\psi) = \{\nu \in \Lambda_{\mathcal{F}} \mid \nu(w, \psi) \in \mathcal{D}\}$.

– A *frame interpretation* is a pair $I = \langle \mathcal{F}, \nu \rangle$, in which $\mathcal{F} = \langle W, R, \mathcal{M} \rangle$ is a frame and ν is an $\mathcal{F}$-valuation. We say that I *satisfies* ψ (or that I is a model of ψ), if $\nu \in mod^w_{\mathcal{F}}(\psi)$ for every $w \in W$. We say that I satisfies Γ if it satisfies every $\psi \in \Gamma$.

Let $\mathcal{I}$ be a nonempty set of frame interpretations. Define a satisfaction relation $\models_{\mathcal{I}}$ on $\mathcal{I} \times F_{\mathcal{L}}$ by $I \models_{\mathcal{I}} \psi$ iff I satisfies ψ. Note that $\mathfrak{I} = \langle \mathcal{I}, \models_{\mathcal{I}} \rangle$ is a denotational semantics in the sense of Definition 2. By Proposition 1, then, the induced relation $\vdash_{\mathfrak{I}}$ is a Tarskian consequence relation for $\mathcal{L}$.

3 Inconsistency-Tolerant Logics

In the context of reasoning with uncertainty, a major drawback of a logic $\langle \mathcal{L}, \vdash_{\mathsf{S}} \rangle$, induced by a denotational semantics $\mathsf{S} = \langle S, \models_S \rangle$, is that it does not tolerate inconsistency properly. Indeed, if $mod_{\mathsf{S}}(\Gamma)$ is empty, then by Definition 3, $\Gamma \vdash_{\mathsf{S}} \psi$ for *every* formula $\psi \in \mathcal{F}_{\mathcal{L}}$. We therefore consider a 'refined' entailment relation, denoted $\mid\!\sim_{\mathsf{S}}$, that overcomes this explosive nature of $\vdash_{\mathsf{S}}$ but respects $\vdash_{\mathsf{S}}$ with respect to consistent theories. Formally, we require the following two properties:

I. Faithfulness: $\mid\!\sim_{\mathsf{S}}$ coincides with $\vdash_{\mathsf{S}}$ with respect to S-consistent theories, i.e., if $mod_{\mathsf{S}}(\Gamma) \neq \emptyset$ then for every $\psi \in \mathcal{F}_{\mathcal{L}}$, $\Gamma \mid\!\sim_{\mathsf{S}} \psi$ iff $\Gamma \vdash_{\mathsf{S}} \psi$.

II Non-Explosiveness: $\mid\!\sim_{\mathsf{S}}$ is not trivialized when the premises are not S-consistent, i.e., if $mod_{\mathsf{S}}(\Gamma) = \emptyset$ then there is $\psi \in \mathcal{F}_{\mathcal{L}}$ such that $\Gamma \not\mid\!\sim_{\mathsf{S}} \psi$.

We call $\mid\!\sim_{\mathsf{S}}$ an *inconsistency-tolerant* variant of $\vdash_{\mathsf{S}}$. When $\vdash_{\mathsf{S}}$ is clear from context, we shall just say that $\mid\!\sim_{\mathsf{S}}$ is inconsistency-tolerant.

Note 1. When $mod_{\mathsf{S}}(\Gamma) \neq \emptyset$ for every theory Γ (as in Priest's logic; see Example 1), $\vdash_{\mathsf{S}}$ itself is inconsistency-tolerant. In what follows we shall be interested in stronger logics (like classical logic) that do not tolerate inconsistency and so need to be refined. Moreover, being a consequence relation, Priest's logic is monotonic, but frequently commonsense reasoning is nonmonotonic, in particular in light of contradictions. Here, again, a refinement of the basic logic, adhering the two properties above, is called upon.

One way of achieving non-explosiveness is by incorporating Shoham's *preferential semantics* [13]: Given a denotational semantics $\mathsf{S} = \langle S, \models_S \rangle$ for $\mathcal{L}$, we define an S-*preferential operator* $\Delta_{\mathsf{S}} : \mathcal{F}_{\mathcal{L}} \to 2^S$ (where 2^S is the power-set of S), that relates a theory Γ to a set $\Delta_{\mathsf{S}}(\Gamma)$ of its 'most preferred' (or 'most plausible') elements in S. Then, the role of $mod_{\mathsf{S}}(\Gamma)$ in Definition 3 is taken now by $\Delta_{\mathsf{S}}(\Gamma)$:

Definition 8. Given a denotational semantics S and a S-preferential operator $\Delta_{\mathsf{S}} : \mathcal{F}_{\mathcal{L}} \to 2^S$, we denote by $\Gamma \mid\!\sim_{\Delta_{\mathsf{S}}} \psi$ that $\Delta_{\mathsf{S}}(\Gamma) \subseteq mod_{\mathsf{S}}(\psi)$.[4]

[4] In words: any conclusion should be satisfied by all the 'preferred' semantical objects (i.e., those elements in S describing the premises in the most plausible way).

Note 2. By faithfulness, every two S-consistent theories that are logically equivalent with respect to $\vdash_{\mathsf{S}}$ (that is, have the same S-models), must also share the same $\mid\!\sim_{\mathcal{S}}$-conclusions. On the other hand, while in any logic defined by denotational semantics (including classical logic) *all* inconsistent theories are logically equivalent, inconsistency-tolerant logics make a distinction between inconsistent theories, so they cannot preserve logical equivalence, and must employ other considerations. This is common to many methods for resolving inconsistencies, e.g., those that are based on information and inconsistency measures (see [8]).

Proposition 3. *Let $\mathsf{S} = \langle S, \models_S \rangle$ be a denotational semantics in which for every $\nu \in S$ there is some formula $\psi \in \mathcal{F}_{\mathcal{L}}$, such that $\nu \not\models_S \psi$.*[5] *Let Δ_{S} be a preferential operator for S. If (1) $\Delta_{\mathsf{S}}(\Gamma)$ is non-empty for every Γ, and (2) $\Delta_{\mathsf{S}}(\Gamma) = mod_{\mathsf{S}}(\Gamma)$ whenever $mod_{\mathsf{S}}(\Gamma)$ is not empty, then $\mid\!\sim_{\Delta_{\mathsf{S}}}$ is inconsistency-tolerant.*

Proof. Faithfulness follows from Condition (2); Non-explosiveness follows from the condition on S and from Condition (1). □

Proposition 3 shows that in many cases inconsistency-tolerant entailments can be obtained from a given denotational semantics S by a proper choice of a preferential operator Δ_{S}. Frequently, such an operator can be defined in terms of a preferential function P that maps every theory Γ to a strict partial order $<_\Gamma$ on S. In such cases,

$$\Delta^{\mathsf{P}}_{\mathsf{S}}(\Gamma) = \{\nu \in S \mid \neg\exists \mu \in S \text{ such that } \mu <_\Gamma \nu\}, \tag{1}$$

so, intuitively, $\Delta^{\mathsf{P}}_{\mathsf{S}}(\Gamma)$ consists of the 'best' elements in terms of $<_\Gamma$.

Proposition 4. *Let S be a denotational semantics as in Proposition 3 and let P be a preferential function, mapping every theory Γ to a strict partial order $<_\Gamma$ on S. If (1) for every theory Γ, $<_\Gamma$ is well-founded, and (2) for every S-consistent Γ, $\min_{<_\Gamma}(S)$ $[= \Delta^{\mathsf{P}}_{\mathsf{S}}(\Gamma)] = mod_{\mathsf{S}}(\Gamma)$, then $\mid\!\sim_{\Delta^{\mathsf{P}}_{\mathsf{S}}}$ is inconsistency-tolerant.*

Proof. Clearly, the two conditions of this proposition imply, respectively, the two conditions of Proposition 3, and so $\mid\!\sim_{\Delta^{\mathsf{P}}_{\mathsf{S}}}$ is inconsistency-tolerant. □

A preferential function P as in Proposition 4 represents *preference by satisfiability*, that is: the models of the underlying theory (if such elements exist) are preferred over the other elements in S.

Proposition 4 specifies natural conditions under which a strict pre-order $<_\Gamma$ induces an inconsistency-tolerant entailment. However, this proposition does not give a method for defining such an order. Next, we consider a simple and intuitive way of doing so by introducing the notion of *similarity*. In what follows, we demonstrate similarity-based reasoning for the three types of denotational semantics discussed previously.

[5] This holds, e.g., when there is a contradictory formula $\bot_{\mathsf{S}}$, for which $mod_{\mathsf{S}}(\bot_{\mathsf{S}}) = \emptyset$.

4 Similarity-Based Reasoning

4.1 Inconsistency Tolerance by Matrix Semantics

Given a matrix $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, we fix the corresponding denotational semantics $\mathsf{S} = \langle \Lambda_{\mathcal{M}}, \models_{\mathcal{M}} \rangle$. For simplicity, we shall identify S with $\mathcal{M}$. Now, the criterion of preference by satisfiability, considered previously for general denotational semantics, can be described in the case of (many-valued) valuations by the aspiration of being 'as similar as possible' to valuations that satisfy the set of premises, Γ. This is depicted in what follows by corresponding quantitative indications.

Definition 9. A *(numeric) aggregation function* is a total function f, such that: (1) for every multiset of real numbers, the value of f is a real number, (2) the value of f does not decrease when a number in its multiset increases, (3) $f(\{x_1, \ldots, x_n\}) = 0$ iff $x_1 = x_2 = \ldots x_n = 0$, and (4) $\forall x \in \mathbb{R}\ f(\{x\}) = x$.

Summation, average, and maximum, are all aggregation functions.

To keep the set of the "preferred valuations" computable, we restrict the comparison of valuations to relevant contexts:

Definition 10. A *context* is a finite set of formulas. A *context generator* is a function $\mathcal{G} : \mathcal{T}_{\mathcal{L}} \to \mathcal{T}_{\mathcal{L}}$, producing a context for every theory.

Simple examples for context generators are, e.g., the functions $\mathcal{G}^{\mathsf{At}}$, $\mathcal{G}^{\mathsf{SF}}$, $\mathcal{G}^{\mathsf{ID}}$, defined for every Γ by $\mathcal{G}^{\mathsf{At}}(\Gamma) = \mathsf{Atoms}(\Gamma)$, $\mathcal{G}^{\mathsf{SF}}(\Gamma) = \mathsf{SF}(\Gamma)$, and $\mathcal{G}^{\mathsf{ID}}(\Gamma) = \Gamma$.

Definition 11. Let $\mathcal{M}$ be a matrix, C a context, and $\mathcal{G}$ a context generator.

- An *$\mathcal{M}$-similarity with respect to* C is a symmetric function $\mathsf{s} : \Lambda_{\mathcal{M}} \times \Lambda_{\mathcal{M}} \to \mathbb{N}^{+}$, such that $\mathsf{s}(\nu, \mu) = 0$ iff $\nu(\phi) = \mu(\phi)$ for all $\phi \in \mathsf{C}$.
- Given an $\mathcal{M}$-similarity s with respect to C, we define:

$$\mathsf{m}^{\mathsf{s}}(\nu, \psi) = \begin{cases} min\{\mathsf{s}(\nu, \mu) \mid \mu \in mod_{\mathcal{M}}(\psi)\} & mod_{\mathcal{M}}(\psi) \neq \emptyset, \\ 1 + max\{\mathsf{s}(\nu, \mu) \mid \nu, \mu \in \Lambda_{\mathcal{M}}\} & \text{otherwise.} \end{cases}$$

- Given an aggregation function f, we define:

$$\mathsf{m}^{\mathsf{s}}_{f}(\nu, \Gamma) = f\big(\{\mathsf{m}^{\mathsf{s}}(\nu, \psi_1), \ldots, \mathsf{m}^{\mathsf{s}}(\nu, \psi_n)\}\big),$$

  where $\Gamma = \{\psi_1, \ldots, \psi_n\}$ and m^{s} is defined as in the previous item by a similarity s with respect to $\mathcal{G}(\Gamma)$.

Note that *lower* values of similarities indicate *higher* correspondence between valuations. However, this correspondence is limited to the relevant contexts: if s is a similarity with respect to C, then $\mathsf{s}(\nu, \mu) = 0$ indicates that ν and μ agree on the formulas of C, but this does not necessarily mean that $\nu = \mu$.

Intuitively, $\mathsf{m}^{\mathsf{s}}(\nu, \psi)$ indices how 'close' ν is to be a model of ψ. The function $\mathsf{m}^{\mathsf{s}}_{f}$ extends m^{s} to theories: $\mathsf{m}^{\mathsf{s}}_{f}(\nu, \Gamma)$ indicates how 'close' is the valuation ν to satisfy Γ. Note that if ψ is *not* $\mathcal{M}$-satisfiable, then, as expected, all the valuations $\nu \in \Lambda_{\mathcal{M}}$ are equally close to ψ: $\mathsf{m}^{\mathsf{s}}(\nu, \psi) = 1 + max\{\mathsf{s}(\nu, \mu) \mid \nu, \mu \in \Lambda_{\mathcal{M}}\}$. By Proposition 5 below, when ψ is $\mathcal{M}$-satisfiable, the valuations ν for which $\mathsf{m}^{\mathsf{s}}(\nu, \psi)$ is minimal, are the models of ψ.

Example 3. Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{V}$ is a finite Euclidean space, i.e., $\mathcal{V} = \mathbb{R}^m = \{\langle x_1, \ldots, x_m \rangle \mid \forall\, 1 \leq i \leq m\ x_i \in \mathbb{R}\}$.[6] The following are distances on $\mathbb{R}^m$:

- *the discrete distance*: $d_U(\overline{x}, \overline{x}) = 0$ and $d_U(\overline{x}, \overline{y}) = 1$ if $\overline{x} \neq \overline{y}$,
- *distance by average*: $d_{\frac{\Sigma}{m}}(\overline{x}, \overline{y}) = \frac{1}{m}\big(\sum_{i=1}^{m} |x_i - y_i|\big)$,
- *the k-norm distance* ($k \geq 1$): $||\overline{x}, \overline{y}||_k = \big(\sum_{i=1}^{m} |x_i - y_i|^k\big)^{\frac{1}{k}}$,
- *the infinity-norm distance*: $\lim_{k\to\infty} ||\overline{x}, \overline{y}||_k = \max(|x_1 - y_1|, \ldots, |x_n - y_n|)$.

Consider the context $\mathsf{C} = \mathsf{Atoms}(\Gamma)$ for some theory Γ. A function s_g, defined for every $\nu, \mu \in \Lambda_{\mathcal{M}}$ by

$$\mathsf{s}_g(\nu, \mu) = g(\{d(\nu(\psi), \mu(\psi)) \mid \psi \in \mathsf{C}\}), \tag{2}$$

where d is one of the distances above and g is an aggregation function, is a similarity function with respect to C. For instance, in the two-valued case, we get the uniform distance with respect to C when $g = max$, and the Hamming distance with respect to C when $g = \Sigma$.

Definition 12. A (semantical) *setting* for $\mathcal{L}$ is a quadruple $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f \rangle$, where $\mathcal{M}$ is a matrix, $\mathcal{G}$ is a context generator, f is an aggregation function, and $\mathcal{S}$ is a *similarity generator*, i.e., a function so that for all Γ $\mathcal{S}(\Gamma)$ is an $\mathcal{M}$-similarity with respect to $\mathcal{G}(\Gamma)$.

Preference by similarities is now defined as follows:

Definition 13. Given a setting $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f \rangle$, the *most plausible valuations* with respect to $\mathcal{K}$ of a (nonempty) theory Γ, are the elements of the set

$$\Delta_{\mathcal{K}}(\Gamma) = \big\{\nu \in \Lambda_{\mathcal{M}} \mid \forall \mu \in \Lambda_{\mathcal{M}}\ \mathsf{m}_f^{\mathcal{S}(\Gamma)}(\nu, \Gamma) \leq \mathsf{m}_f^{\mathcal{S}(\Gamma)}(\mu, \Gamma)\big\}.$$

In case that Γ is empty, we define $\Delta_{\mathcal{K}}(\emptyset) = \Lambda_{\mathcal{M}}$.

Note that $\Delta_{\mathcal{K}}$ can be represented in the form of (1), where $<_\Gamma$ is defined by $\nu <_\Gamma \mu$ iff $\nu \in \Delta_{\mathcal{K}}(\Gamma)$ and $\mu \notin \Delta_{\mathcal{K}}(\Gamma)$. Now, similarity-based entailments are defined as in Definition 8:

$$\Gamma \mid\!\sim_{\Delta_{\mathcal{K}}} \psi \text{ iff } \Delta_{\mathcal{K}}(\Gamma) \subseteq mod_{\mathcal{M}}(\psi). \tag{3}$$

Example 4. Let $\mathcal{K} = \langle \mathcal{M}_K^{3_\perp}, \mathcal{G}^{\mathsf{At}}, \mathcal{S}, \Sigma \rangle$, where $\mathcal{M}_K^{3_\perp}$ is Kleene's three-valued matrix (Example 1), $\mathcal{G}^{\mathsf{At}}$ is the atom-based context generator (see below Definition 10), Σ is a summation function, and $\mathcal{S}$ is a similarity generator that for

[6] This includes, among others, linearly ordered values (as in the three-valued logics considered above, or the elements of the unit interval), that are represented by a one-dimensional space; partial orders in which there are at most $i-1$ different x_j's such that $\mathsf{f} < x_1 < \ldots < x_{i-1} < \mathsf{t}$, that may be represented by pairs of numbers in $\{0, \ldots, i-1\}$ (see [2] for this kind of representation for Belnap's four-valued logic); the elements of an Nmatrix for mbC that can be represented by triples (see [5]), etc.

each Γ produces a similarity s_Σ in the form of (2), i.e., $\mathcal{S}(\Gamma)(\nu,\mu) = \mathsf{s}_\Sigma(\nu,\mu) = \Sigma\{d_\Sigma(\nu(p),\mu(p)) \mid p \in \mathsf{Atoms}(\Gamma)\}$. Here, d_Σ is a distance on $\{\mathsf{t},\mathsf{f},\bot\}$, in which $d_\Sigma(\mathsf{t},\mathsf{f}) = 1$ and $d_\Sigma(\mathsf{t},\bot) = d_\Sigma(\mathsf{f},\bot) = \frac{1}{2}$ (see Example 3).

Now, let $\Gamma = \{\neg p, \neg q, p \vee q\}$. Clearly, Γ is not $\mathcal{M}_K^{3\bot}$-satisfiable. We compute its most plausible models w.r.t. $\mathcal{K}$:

	p	q	$\neg p$	$\neg q$	$p \vee q$	1	2	3	$\mathsf{m}_\Sigma(\nu_i,\Gamma)$
ν_1	t	t	f	f	t	1	1	0	2
ν_2	t	f	f	t	t	1	0	0	**1**
ν_3	t	$\bot$	f	$\bot$	t	1	0.5	0	1.5
ν_4	f	t	t	f	t	0	1	0	**1**
ν_5	f	f	t	t	f	0	0	1	**1**
ν_6	f	$\bot$	t	$\bot$	$\bot$	0	0.5	0.5	**1**
ν_7	$\bot$	t	$\bot$	f	t	0.5	1	0	1.5
ν_8	$\bot$	f	$\bot$	t	$\bot$	0.5	0	0.5	**1**
ν_9	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	0.5	0.5	0.5	1.5

Legend. $1 = \mathsf{m}^{\mathsf{s}}(\nu_i,\neg p)$, $2 = \mathsf{m}^{\mathsf{s}}(\nu_i,\neg q)$, $3 = \mathsf{m}^{s}(\nu_i, p \vee q)$.

Hence, $\Delta_\mathcal{K}(\Gamma) = \{\nu_2,\nu_4,\nu_5,\nu_6,\nu_8\}$, and so, for instance, $\Gamma \mathrel{|\!\sim}_{\Delta_\mathcal{K}} \neg p \vee \neg q$ (even though $\Gamma \not\mathrel{|\!\sim}_{\Delta_\mathcal{K}} \neg p$ and $\Gamma \not\mathrel{|\!\sim}_{\Delta_\mathcal{K}} \neg q$).

In what follows, we shall abbreviate $\mathrel{|\!\sim}_{\Delta_\mathcal{K}}$ by $\mathrel{|\!\sim}_\mathcal{K}$. Next, we show that entailments of this type are inconsistency tolerant.

Definition 14. Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O}\rangle$ be a matrix for $\mathcal{L}$. A context C is *proper for* ψ (in $\mathcal{M}$), if for every $\nu,\mu \in \Lambda_\mathcal{M}$, if $\nu(\phi) = \mu(\phi)$ for all $\phi \in \mathsf{C}$, then $\nu(\psi) = \mu(\psi)$ as well. C is a proper context for a theory Γ, if it is proper for every $\psi \in \Gamma$.

In what follows we consider only *proper settings*, that is: settings $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f\rangle$ in which for every theory Γ, $\mathcal{G}(\Gamma)$ is a proper context for Γ (in $\mathcal{M}$). Note that for all the context generators considered above ($\mathcal{G}^{\mathsf{At}}$, $\mathcal{G}^{\mathsf{SF}}$, and $\mathcal{G}^{\mathsf{ID}}$), the corresponding setting is proper.

Proposition 5. *Let* s *be a similarity with respect to a context* C *that is proper for* ψ. *Then* $\mathsf{m}^{\mathsf{s}}(\nu,\psi) = 0$ *iff* $\nu \in mod_\mathcal{M}(\psi)$.

Corollary 1. *Let* s *be a similarity with respect to a context* C *that is proper for* Γ. *Then* $\mathsf{m}^{\mathsf{s}}_f(\nu,\Gamma) = 0$ *iff* $\nu \in mod_\mathcal{M}(\Gamma)$.

Proof. By Proposition 5 and since f is an aggregation function. □

Definition 15. Γ_1 and Γ_2 are *independent*, if $\mathsf{Atoms}(\Gamma_1) \cap \mathsf{Atoms}(\Gamma_2) = \emptyset$.

Proposition 6. *Let* $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f\rangle$ *be a semantic setting. If* $\mathcal{G}(\Gamma)$ *and* $\{\psi\}$ *are independent, then* $\Gamma \mathrel{|\!\sim}_\mathcal{K} \psi$ *iff* ψ *is an* $\mathcal{M}$*-tautology.*

Corollary 2. *Let* $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f\rangle$ *be a semantic setting. For every* Γ *there is a formula* ψ *such that* $\Gamma \not\mathrel{|\!\sim}_\mathcal{K} \psi$.

Proof. Given Γ, let $p \in \mathsf{Atoms} \setminus \mathcal{G}(\Gamma)$ (such a p exists, since Atoms is infinite and $\mathcal{G}(\Gamma)$ is not). As $\mathcal{G}(\Gamma)$ and $\{p\}$ are independent, by Proposition 6, $\Gamma \not\sim_{\mathcal{K}} p$. □

Proposition 7. *For every setting $\mathcal{K} = \langle \mathcal{M}, \mathcal{G}, \mathcal{S}, f \rangle$, $\vdash_{\mathcal{K}}$ is an inconsistency-tolerant variant of $\vdash_{\mathcal{M}}$.*

Proof. Faithfulness to $\vdash_{\mathcal{M}}$ follows from Corollary 1; Non-explosiveness follows from Corollary 2. □

4.2 Inconsistency Tolerance by Nmatrices

Similarity-based entailments can be defined in the non-deterministic case just as in the deterministic case. Given an Nmatrix $\mathcal{M}$, similarities and satisfiability measures are defined according to Definition 11. This induces the operator $\Delta_{\mathcal{K}}$ and the entailment $\sim_{\Delta_{\mathcal{K}}}$, as in Definition 13 and in (3), respectively.

The results in the previous section also carry on to non-deterministic semantics. It is important to note, though, that in the non-deterministic case the context generator $\mathcal{G}^{\mathsf{At}}$ does *not* produce proper contexts. This is explained by the fact that, unlike deterministic valuations, non-deterministic valuations are not truth functional, so they can agree on atomic formulas, but make different non-deterministic choices on complex formulas. Yet, as the next proposition shows, the other two context generators do provide proper contexts:

Proposition 8. *For every Nmatrix $\mathcal{M}$, similarity generator $\mathcal{S}$, and aggregation f, both $\langle \mathcal{M}, \mathcal{G}^{\mathsf{SF}}, \mathcal{S}, f \rangle$ and $\langle \mathcal{M}, \mathcal{G}^{\mathsf{ID}}, \mathcal{S}, f \rangle$ are proper.*

Proof. By the fact that if $\psi \in \mathsf{C}$ then C is proper in $\mathcal{M}$ for ψ. □

Example 5. Let $\mathcal{K} = \langle \mathcal{M}^{3_\perp}_{KM}, \mathcal{G}^{\mathsf{ID}}, \mathcal{S}, \Sigma \rangle$ be a setting in which $\mathcal{M}^{3_\perp}_{KM}$ is the Nmatrix of Example 2, combining Kleene's and McCarthy's three-valued logics, $\mathcal{G}^{\mathsf{ID}}$ is the context generator by identity, Σ is a summation function, and $\mathcal{S}$ a similarity generator defined for every Γ by $\mathcal{S}(\Gamma)\,(\nu, \mu) = \Sigma\,\{d_\Sigma(\nu(\psi), \mu(\psi)) \mid \psi \in \Gamma\}$. Again, here d_Σ is the distance on $\{\mathsf{t}, \mathsf{f}, \bot\}$ defined in Example 3.

As in Example 4, we let $\Gamma = \{\neg p, \neg q, p \vee q\}$. Clearly, Γ is not $\mathcal{M}^{3_\perp}_{KM}$-satisfiable. Note that in addition to the nine valuations in Example 4 we also have $\nu_{10} = \{p{:}\bot, q{:}\mathsf{t}, \neg p{:}\bot, \neg q{:}\mathsf{f}, p \vee q{:}\bot\}$. It can be verified that this time $\Gamma \not\sim_{\Delta_{\mathcal{K}}} \neg p \vee \neg q$ (cf. Example 4).

4.3 Inconsistency Tolerance by Possible Worlds

We now extend similarities to the context of finite frames.

Definition 16. Let $\mathcal{F} = \langle W, R, \mathcal{M} \rangle$ be a *finite* frame, C a context, and $\mathcal{G}$ a context generator.

- An *$\mathcal{F}$-similarity with respect to* C is a symmetric function $\mathsf{s} : \Lambda_{\mathcal{F}} \times \Lambda_{\mathcal{F}} \to \mathbb{N}^+$, such that $\mathsf{s}(\nu, \mu) = 0$ iff $\nu(w, \psi) = \mu(w, \psi)$ for every $w \in W$ and $\psi \in \mathsf{C}$.

– Given an $\mathcal{F}$-similarity s with respect to C, we define:

$$\mathsf{m}^{\mathsf{s}}(w,\nu,\psi) = \begin{cases} min\{\mathsf{s}(\nu,\mu) \mid \mu \in mod^{w}_{\mathcal{F}}(\psi)\} & mod^{w}_{\mathcal{F}}(\psi) \neq \emptyset, \\ 1 + max\{\mathsf{s}(\nu,\mu) \mid \nu,\mu \in \Lambda_{\mathcal{F}}\} & \text{otherwise.} \end{cases}$$

– For a frame interpretation $I = \langle \mathcal{F}, \nu \rangle$ and an aggregation function f, define: $\mathsf{m}^{\mathsf{s}}_{f}(I,\psi) = f\big(\{\mathsf{m}^{\mathsf{s}}(w,\nu,\psi) \mid w \in W\}\big)$, where m^{s} is defined as in the previous item by an $\mathcal{F}$-similarity s.

– For a frame interpretation $I = \langle \mathcal{F}, \nu \rangle$ and aggregation functions g, f, define: $\mathsf{m}^{\mathsf{s}}_{g,f}(I,\Gamma) = g\big(\{\mathsf{m}^{\mathsf{s}}_{f}(I,\psi_1),\ldots,\mathsf{m}^{\mathsf{s}}_{f}(I,\psi_n)\}\big)$, where $\Gamma = \{\psi_1,\ldots,\psi_n\}$ and $\mathsf{m}^{\mathsf{s}}_{f}$ is defined as in the item above by an $\mathcal{F}$-similarity s with respect to $\mathcal{G}(\Gamma)$.

Definition 17. A *setting* for $\mathcal{L}$ is a quintuple $\mathcal{K} = \langle \mathcal{I}, \mathcal{G}, \mathcal{S}, f, g \rangle$, where $\mathcal{I}$ is a set of finite frames, $\mathcal{G}$ is a context generator, f and g are aggregation functions, and $\mathcal{S}$ is a *similarity generator* for $\mathcal{G}$, i.e., for every $I = \langle \mathcal{F}, \nu \rangle \in \mathcal{I}$ and $\Gamma \in \mathcal{T}_{\mathcal{L}}$, $\mathcal{S}(I,\Gamma)$ is an $\mathcal{F}$-similarity with respect to $\mathcal{G}(\Gamma)$.

Example 6. Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix where $\mathcal{V} \subseteq \mathbb{R}^m$. A variety of similarity generators can be defined by letting d be one of the distances on $\mathbb{R}^m$ from Example 3: $\mathcal{S}(I,\Gamma)\,(\nu,\mu) = f_2\big(\{f_1\big(\{d(\nu(w,\psi),\mu(w,\psi)) \mid \psi \in \mathcal{G}(\Gamma)\}\big) \mid w \in W\}\big)$, where f_1, f_2 are some aggregation functions and $\mathcal{G}$ is a context generator.

Definition 18. Let $\mathcal{K} = \langle \mathcal{I}, \mathcal{G}, \mathcal{S}, f, g \rangle$ be a setting. The set of the *most plausible frame interpretations* of $\Gamma \neq \emptyset$ with respect to $\mathcal{K}$ is defined as follows:

$$\Delta_{\mathcal{K}}(\Gamma) = \big\{I \in \mathcal{I} \mid \forall J \in \mathcal{I} \;\; \mathsf{m}^{\mathcal{S}(I,\Gamma)}_{g,f}(I,\Gamma) \leq \mathsf{m}^{\mathcal{S}(J,\Gamma)}_{g,f}(J,\Gamma)\big\}.$$

If $\Gamma = \emptyset$, we define $\Delta_{\mathcal{K}}(\emptyset) = \mathcal{I}$.

Again, $\Delta_{\mathcal{K}}$ is a particular case of (1). Now, for a (multi-valued) possible world semantics $\mathfrak{I} = \langle \mathcal{I}, \models_{\mathcal{I}} \rangle$ and a corresponding semantic setting $\mathcal{K} = \langle \mathcal{I}, \mathcal{G}, \mathcal{S}, f, g \rangle$ we define, like before, $\Gamma \mathrel{|\!\sim_{\mathcal{K}}} \psi$ iff $\Delta_{\mathcal{K}}(\Gamma) \subseteq mod_{\mathfrak{I}}(\psi)$.

To show that $\mathrel{|\!\sim_{\mathcal{K}}}$ is an inconsistency-tolerant variant of $\vdash_{\mathfrak{I}}$, we consider a natural extension to possible-world semantics of the notion of properness:

Definition 19. Let $\mathcal{F} = \langle W, R, \mathcal{M} \rangle$ be a frame for $\mathcal{L}$. A context C is *proper for* ψ (in $\mathcal{F}$), if for every $\nu, \mu \in \Lambda_{\mathcal{F}}$ and every $w \in W$, if $\nu(w,\phi) = \mu(w,\phi)$ for all $\phi \in \mathsf{C}$, then $\nu(w,\psi) = \mu(w,\psi)$ as well. We say that C is proper for Γ if it is proper for every $\psi \in \Gamma$.

Note that, as before, for all the context generators considered above ($\mathcal{G}^{\mathsf{At}}$, $\mathcal{G}^{\mathsf{SF}}$, and $\mathcal{G}^{\mathsf{ID}}$), the corresponding setting is proper.

Proposition 9. *Let $\mathfrak{I} = \langle \mathcal{I}, \models_{\mathcal{I}} \rangle$ be a multi-valued possible world semantics, and let $\mathcal{K} = \langle \mathcal{I}, \mathcal{G}, \mathcal{S}, f, g \rangle$ be a corresponding proper setting. Then $\mathrel{|\!\sim_{\mathcal{K}}}$ is an inconsistency-tolerant variant of $\vdash_{\mathfrak{I}}$.*

Example 7. Consider two companies a and b and two investment houses, h_1 and h_2. An investment house h buys shares of a company if the latter is recommended by *all* the investment houses that h knows; otherwise h sells its shares. This can be modeled by a language $\mathcal{L} = \{\Box, \wedge, \neg\}$, and the classical two-valued matrix $\mathcal{M}_{cl}$ with the standard interpretations. We use two atoms in $\mathcal{L}$: $\mathsf{R_a}$ and $\mathsf{R_b}$ (where $\mathsf{R_x}$ intuitively means that 'company x is recommended') and denote by $\mathsf{Buy}(x)$ and by $\mathsf{Sell}(x)$ (for $x \in \{a, b\}$) the formulas $\Box\mathsf{R}_x$, and $\neg\Box\mathsf{R}_x$, respectively.

Suppose now that a third party, call it h_3, wants to detect the trading intentions of the two investment houses. However, h_3 faces two problems. One is that h_3 gets contradictory rumors about these intentions: One rumor says that both houses are going to buy shares of a and b: $\mathsf{Buy(a,b)} = \mathsf{Buy(a)} \wedge \mathsf{Buy(b)}$, and the other rumor claims that they will sell the shares of a. The third party has, then, an inconsistent theory describing the situation $\Gamma = \{\mathsf{Buy(a,b)}, \mathsf{Sell(a)}\}$.

The other problem of h_3 is that it does not know whether h_1 and h_2 have access to each other (but it does know that accessibility must be symmetric and reflexive). This can be represented by two frames $\mathcal{F}_i = \langle W, R_i, \mathcal{M}_{cl}\rangle$ (for $i = 1, 2$), in which $W = \{h_1, h_2\}$, $R_1 = \{\langle h_1, h_2\rangle, \langle h_2, h_1\rangle, \langle h_1, h_1\rangle, \langle h_2, h_2\rangle\}$, and $R_2 = \{\langle h_1, h_1\rangle, \langle h_2, h_2\rangle\}$. The corresponding possible world semantics is $\mathfrak{I} = \langle \mathcal{I}, \models_{\mathcal{I}}\rangle$ with $\mathcal{I} = \cup_{i=1,2}\{\langle \mathcal{F}_i, \nu\rangle \mid \nu \in \Lambda_{\mathcal{F}_i}\}$.

For making plausible decisions despite these uncertainties, h_3 uses $\mid\!\sim_{\mathcal{K}}$, the inconsistency-tolerant variant of $\vdash_{\mathfrak{I}}$, induced by the setting $\mathcal{K} = \langle \mathcal{I}, \mathcal{G}^{\mathsf{At}}, \mathcal{S}, \Sigma, \Sigma\rangle$, where $\mathcal{S}$ is defined by $\mathcal{S}(I, \Gamma)(\nu, \mu) = \Sigma_{w\in W}\, \Sigma_{\psi\in \mathsf{Atoms}(\Gamma)}\, d_U\big(\nu(w, \psi), \mu(w, \psi)\big)$. The relevant frame interpretations are represented in the table below:

I_i	1	2	3	4	5	6	7	8	$\mathsf{m^s}(I_i, \Gamma)$	I_i	1	2	3	4	5	6	7	8	$\mathsf{m^s}(I_i, \Gamma)$
I^1_1	f	f	f	f	0	0	4	4	8	I^2_1	f	f	f	f	0	0	2	2	4
I^1_2	f	f	f	t	0	0	3	3	6	I^2_2	f	f	f	t	0	0	2	1	3
I^1_3	f	f	t	f	0	0	3	3	6	I^2_3	f	f	t	f	0	1	2	1	4
I^1_4	f	f	t	t	0	0	2	2	4	I^2_4	f	f	t	t	0	1	2	0	3
I^1_5	f	t	f	f	0	0	3	3	6	I^2_5	f	t	f	f	0	0	1	2	3
I^1_6	f	t	f	t	0	0	2	2	4	I^2_6	f	t	f	t	0	0	1	1	**2**
I^1_7	f	t	t	f	0	0	2	2	4	I^2_7	f	t	t	f	0	1	1	1	3
I^1_8	f	t	t	t	0	0	1	1	**2**	I^2_8	f	t	t	t	0	1	1	0	**2**
I^1_9	t	f	f	f	0	0	3	3	6	I^2_9	t	f	f	f	1	0	1	2	4
I^1_{10}	t	f	f	t	0	0	2	2	4	I^2_{10}	t	f	f	t	1	0	1	1	3
I^1_{11}	t	f	t	f	1	1	2	2	6	I^2_{11}	t	f	t	f	1	1	1	1	4
I^1_{12}	t	f	t	t	1	1	1	1	4	I^2_{12}	t	f	t	t	1	1	1	0	3
I^1_{13}	t	t	f	f	0	0	2	2	4	I^2_{13}	t	t	f	f	1	0	0	2	3
I^1_{14}	t	t	f	t	0	0	1	1	**2**	I^2_{14}	t	t	f	t	1	0	0	1	**2**
I^1_{15}	t	t	t	f	1	1	1	1	4	I^2_{15}	t	t	t	f	1	1	0	1	3
I^1_{16}	t	t	t	t	1	1	0	0	**2**	I^2_{16}	t	t	t	t	1	1	0	0	**2**

Legend: $1 = \nu_i(h_1, \mathsf{R_a})$, $2 = \nu_i(h_1, \mathsf{R_b})$, $3 = \nu_i(h_2, \mathsf{R_a})$, $4 = \nu_i(h_2, \mathsf{R_b})$,
$5 = \mathsf{m^s}(h_1, \nu_i, \mathsf{Sell(a)})$, $6 = \mathsf{m^s}(h_2, \nu_i, \mathsf{Sell(a)})$, $7 = \mathsf{m^s}(h_1, \nu_i, \mathsf{Buy(a,b)})$,
$8 = \mathsf{m^s}(h_2, \nu_i, \mathsf{Buy(a,b)})$.

It follows that $\Delta_{\mathcal{K}}(\Gamma) = \{I_8^1, I_{14}^1, I_{16}^1, I_6^2, I_8^2, I_{14}^2, I_{16}^2\}$ and so $\Gamma \mid\sim_{\mathcal{K}} \mathsf{Buy(b)}$ while $\Gamma \not\mid\sim_{\mathcal{K}} \mathsf{Sell(a)}$. The third party anticipates, then, that the other houses will buy b, but it cannot infer that they will sell a.

5 Conclusion

We have introduced a general method of supplementing different logics, based on denotational semantics, with extra apparatus assuring a proper tolerance of inconsistency. This is also the main motivation of other works, such as [1] that introduced distance-based reasoning in deterministic matrices, and [3] that considers distance reasoning in two-valued non-deterministic matrices. This paper generalizes and extends those works in the following senses: First, the notion of similarities is a generalization of the notion of distances, allowing to incorporate a wider range of measures. This also admits the definition of some preferential logics that are not even cumulative (the weakest family of preferential logics considered in the well-known framework of Kraus-Lehmann-Magidor [10]), but which still have some merit for AI applications. Second, our framework captures some common properties shared by inconsistency-tolerant logics based on *any* kind of denotational semantics, whereas the other works handle only specific cases. In particular, new reasoning platforms are investigated, including applications within generalized Kripke-structures, and the extension of the similarity-based approach to many-valued matrices. The latter has not been investigated for Nmatrices, and yields some natural generalizations of well-studied distances.

References

1. Arieli, O.: Distance-based paraconsistent logics. International Journal of Approximate Reasoning 48(3), 766–783 (2008)
2. Arieli, O., Denecker, M.: Reducing preferential paraconsistent reasoning to classical entailment. Logic and Computation 13(4), 557–580 (2003)
3. Arieli, O., Zamansky, A.: Distance-based non-deterministic semantics for reasoning with uncertainty. Logic Journal of the IGPL 17(4), 325–350 (2009)
4. Avron, A., Lev, I.: Non-deterministic multi-valued structures. Logic and Computation 15, 241–261 (2005)
5. Avron, A., Zamansky, A.: Non-deterministic semantics for logical systems (A survey). In: Handbook of Philosophical Logic (Forthcoming)
6. da Costa, N.C.A.: On the theory of inconsistent formal systems. Notre Dame Journal of Formal Logic 15, 497–510 (1974)
7. Fitting, M.: Many-valued modal logics. Fundam. Inform. 15(3-4), 235–254 (1991)
8. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: Proc. KR 2008, pp. 358–366. AAAI Press, Menlo Park (2008)
9. Kleene, S.C.: Introduction to Metamathematics. Van Nostrand (1950)
10. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
11. McCarthy, J.: A basis for a mathematical theory of computation. In: Computer Programming and Formal Systems, pp. 33–70 (1963)
12. Priest, G.: Reasoning about truth. Artificial Intelligence 39, 231–244 (1989)
13. Shoham, Y.: Reasoning about Change. MIT Press, Cambridge (1988)

Decomposition of Distributed Nonmonotonic Multi-Context Systems*

Seif El-Din Bairakdar, Minh Dao-Tran, Thomas Eiter,
Michael Fink, and Thomas Krennwallner

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{bairakdar,dao,eiter,fink,tkren}@kr.tuwien.ac.at

Abstract. Multi-Context Systems (MCS) are formalisms that enable the interlinkage of single knowledge bases, called contexts, via bridge rules. Recently, a fully distributed algorithm for evaluating heterogeneous, nonmonotonic MCS was described in [7]. In this paper, we continue this line of work and present a decomposition technique for MCS which analyzes the topology of an MCS. It applies pruning techniques to get economically small representations of context dependencies. Orthogonal to this, we characterize minimal interfaces for information exchange between contexts, such that data transmissions can be minimized. We then present a novel evaluation algorithm that operates on a query plan which is compiled with topology pruning and interface minimization. The effectiveness of the optimization techniques is demonstrated by a prototype implementation, which uses an off-the-shelf SAT solver and shows encouraging experimental results.

1 Introduction

In the last years, there has been increasing interest in systems comprising multiple knowledge bases. The rise of distributed systems and the World Wide Web fostered this development, and to date, several formalisms are available that accommodate multiple, possibly distributed knowledge bases. One formalism are Multi-Context Systems (MCS) consisting of several theories (the contexts) that are interlinked with bridge rules which allow to add knowledge to a context depending on knowledge in other contexts. E.g., the bridge rule $a \leftarrow (2 : b)$ of a context C_1 means that C_1 should conclude a if context C_2 believes b. MCS have applications in various areas, such as argumentation, data integration, or multi-agent systems. There, contexts may model the beliefs of an agent while the bridge rules model an agent's perception of the environment, i.e., other contexts.

Among the various MCS proposals (e.g., [10,11,12]), the general MCS framework of [5] is of special interest, as it generalizes previous approaches in contextual reasoning and allows for *heterogeneous and nonmonotonic* MCS, i.e., with different, possibly nonmonotonic logics in its contexts (thus furthering heterogeneity), and bridge rules

* This research has been supported by the Austrian Science Fund (FWF) project P20841 and by the Vienna Science and Technology Fund (WWTF) project ICT 08-020.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 24–37, 2010.
© Springer-Verlag Berlin Heidelberg 2010

may use default negation (to deal, e.g., with incomplete information). Hence, nonmonotonic MCS interlinking monotonic context logics are possible. This MCS framework can conveniently capture the following scenario, which we use as a running example.

Example 1. A group of four scientists, Ms. 1, Mr. 2, Mr. 3, and Ms. 4, just finished their conference visit and are now arranging a trip back home. They can choose between going by train or by car (which is usually slower than the train); and if they use the train, they should bring along some food. Moreover, Mr. 3 and Ms. 4 have additional information from home that might affect their decision.

Mr. 3 has a daughter, Ms. 6. He is fine with either transportation option, but if Ms. 6 is sick then he wants to use the fastest vehicle to get home. Ms. 4 just got married, and her husband, Mr. 5, wants her to come back as soon as possible. He urges her to try to come home even sooner, while Ms. 4 tries to yield to her husband's plea.

If they go by train, Mr. 3 is responsible for buying provisions. He might choose either salad or peanuts. The options for beverages are coke or juice. Mr. 2 is a modest person as long as he gets home. He agrees to any choice that Mr. 3 and Ms. 4 select for vehicle but he dislikes coke. Ms. 1 is the leader of the group and prefers to go by car, but if Mr. 2 and 3 go by train then she would not object. A problem is that Ms. 1 is allergic to nuts.

Mr. 3 and Ms. 4 do not want to bother the group with their circumstances and communicate just their preferences, which is sufficient for reaching an agreement. Ms. 1 decides which option to take based on the information she gets from Mr. 2 and Mr. 3.

Similar scenarios have already been investigated in the realm of multi-agent systems (see, e.g., [6] on social answer set programming). We do not aim at introducing a new semantics for such scenarios; our example is meant to be a plain showcase application of MCS. We stress that MCS have potential as a host for KR formalisms, just like answer set programs have; however, in this paper we concentrate on efficient MCS evaluation.

The distributed algorithm introduced in [7], called DMCS, computes the semantics of an MCS, which is given in terms of equilibria. Roughly, an equilibrium is a collection of local models (belief sets) for the individual contexts that is compatible with the bridge rules. The principle of the algorithm is, starting from context C_k (the root), that models will be processed at each context. Bridge rules, which access beliefs in other contexts, implicitly span belief import dependencies between contexts. This relationship is used to navigate the system, and models returned from invoked neighbors are combined with the local beliefs and passed back to the invoking contexts. DMCS uses a parameter for projecting models to relevant variables to reduce data payload.

Experiments for an instantiation of DMCS with answer set programming contexts revealed some scalability issues which can be tracked down to the following problems:

(1) contexts are unaware of context dependencies in the system beyond their neighbors, and thus treat each neighbor in a generic way. Specifically, cyclic dependencies remain undetected until a context, seeing the invocation chain, requests models from a context in the chain. Furthermore, a context C_k does not know whether a neighbor C_i already requests models from another neighbor C_j which then would be passed to C_k; hence, C_k makes possibly a superfluous request to C_j.

(2) a context C_i returns the combination of its local models with the models received from all neighboring contexts. As contexts may have multiple models, the number of models can become huge as the size of the system respectively neighbors increases. In fact, this is one of the main performance obstacles.

In this work, we address the issue of optimization; there is an urgent need for this in order to increase the scalability of distributed MCS evaluation. Resorting to methods from graph theory, we aim at decomposing, pruning, and improved cycle breaking for dependencies in multi-context systems. Focusing on (1), we describe a decomposition method using biconnected components of inter-context dependencies. Based on this we can break cycles and prune acyclic parts before evaluating the system and create an acyclic query plan. To address (2), we foster a partial view of the system, which is often sufficient to reach a satisfactory answer. In Ex. 1, e.g., we could mask out the beliefs of Mr. 5 and Ms. 6 to compute a partial equilibrium within the scientist group. This way we can make a compromise between partial information and performance. We thus define a set of variables for each import dependency in the system to project the models in each context to the bare minimum such that they continue to be meaningful. In this manner, we can omit needless information and circumvent excessive model combinations.

Based on these ideas, we have designed a new evaluation algorithm DMCSOPT, which intertwines decomposition and pruning with variable projection. For evaluation, we adapted our DMCS prototype and ran some experiments. The results show a major improvement compared to DMCS; here we can handle systems with up to 600 contexts. This demonstrates that our optimization techniques are effective and bring MCS closer to applications.

2 Preliminaries

We recall some basic notions of heterogeneous nonmonotonic multi-context systems [5].

A *logic* is, viewed abstractly, a tuple $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, where

- $\mathbf{KB}_L$ is a set of well-formed knowledge bases, each being a set (of formulas),
- $\mathbf{BS}_L$ is a set of possible belief sets, each being a set (of formulas), and
- $\mathbf{ACC}_L \colon \mathbf{KB}_L \to 2^{\mathbf{BS}_L}$ assigns each $kb \in \mathbf{KB}_L$ a set of acceptable belief sets.

This covers many (non-)monotonic KR formalisms like description logics, default logic, answer set programs, etc. For example, a (propositional) *ASP logic* L may be such that $\mathbf{KB}_L$ is the set of answer set programs over a (propositional) alphabet $\mathcal{A}$, $\mathbf{BS}_L = 2^{\mathcal{A}}$ contains all subsets of atoms, and $\mathbf{ACC}_L$ assigns each $kb \in \mathbf{KB}_L$ the set of all its answer sets (see [9] for details).

Definition 1. *A* multi-context system (MCS) $M = (C_1, \ldots, C_n)$ *consists of* contexts $C_i = (L_i, kb_i, br_i)$, $1 \leq i \leq n$, *where* $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ *is a logic,* $kb_i \in \mathbf{KB}_i$ *is a knowledge base, and* br_i *is a set of* L_i-bridge rules *of the form*

$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathrm{not}\,(c_{j+1} : p_{j+1}), \ldots, \mathrm{not}\,(c_m : p_m) \tag{1}$$

where $1 \leq c_k \leq n$, p_k *is an element of some belief set of* L_{c_k}, $1 \leq k \leq m$, *and* $kb \cup \{s\} \in \mathbf{KB}_i$ *for each* $kb \in \mathbf{KB}_i$.

Informally, bridge rules allow to modify the knowledge base by adding s, depending on the beliefs in other contexts.

The semantics of an MCS M is defined in terms of particular *belief states*, which are sequences $S = (S_1, \ldots, S_n)$ of belief sets $S_i \in \mathbf{BS}_i$. Intuitively, S_i should be a belief set of the knowledge base kb_i; however, also the bridge rules br_i must be respected. To this end, kb_i is augmented with the conclusions of all $r \in br_i$ that are applicable.

Formally, r of form (1) is *applicable in* S, if $p_i \in S_{c_i}$, for $1 \leq i \leq j$, and $p_k \notin S_{c_k}$, for $j+1 \leq k \leq m$. Let $app(R, S)$ denote the set of all bridge rules $r \in R$ that are applicable in S. Furthermore, $head(r)$ denotes the part s, and $B(r) = \{(c_k : p_k) \mid 1 \leq k \leq m\}$, for any r of form (1).

Definition 2. *A belief state $S = (S_1, \ldots, S_n)$ of a multi-context system M is an* equilibrium *iff for all $1 \leq i \leq n$, $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$.*

In the rest of this paper, we assume that contexts C_i have finite belief sets S_i that are represented by truth assignments $v_{S_i} : \Sigma_i \to \{0, 1\}$ to a finite set Σ_i of propositional atoms such that $p \in S_i$ iff $v_{S_i}(p) = 1$ (as in [5], such S_i may serve as kernels that correspond 1-1 to infinite belief sets). Furthermore, we assume that the Σ_i are pairwise disjoint and that $\Sigma = \bigcup_i \Sigma_i$.

Example 2. The scenario in Ex. 1 can be encoded as an MCS $M = (C_1, \ldots, C_6)$, where all L_i are ASP logics (t_i and c_i represent train and car in C_i, resp.) and

- $kb_1 = \{c_1 \leftarrow \text{not } t_1; \quad \bot \leftarrow nuts_1\}$ and
  $br_1 = \{t_1 \leftarrow (2 : t_2), (3 : t_3); \quad nuts_1 \leftarrow (3 : peanuts_3)\}$;
- $kb_2 = \{\bot \leftarrow \text{not } c_2, \text{not } t_2\}$ and
  $br_2 = \{c_2 \leftarrow (3 : c_3), (4 : c_4); \quad t_2 \leftarrow (3 : t_3), (4 : t_4), \text{not } (3 : coke_3)\}$;
- $kb_3 = \{c_3 \vee t_3 \leftarrow; t_3 \leftarrow urgent_3; salad_3 \vee peanuts_3 \leftarrow t_3; coke_3 \vee juice_3 \leftarrow t_3\}$ and $br_3 = \{urgent_3 \leftarrow (6 : sick_6);\ t_3 \leftarrow (4 : t_4)\}$;
- $kb_4 = \{c_4 \vee t_4 \leftarrow\}$ and $br_4 = \{t_4 \leftarrow (5 : sooner_5)\}$;
- $kb_5 = \{sooner_5 \leftarrow soon_5\}$ and $br_5 = \{soon_5 \leftarrow (4 : t_4)\}$;
- $kb_6 = \{sick_6 \vee fit_6 \leftarrow\}$ and $br_6 = \emptyset$.

The context dependencies of M are shown in Fig. 1b. M has three equilibria:

- $S = (\{t_1\}, \{t_2\}, \{t_3, urgent_3, juice_3, salad_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{sick_6\})$;
- $T = (\{t_1\}, \{t_2\}, \{t_3, juice_3, salad_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{fit_6\})$; and
- $U = (\{c_1\}, \{c_2\}, \{c_3\}, \{c_4\}, \emptyset, \{fit_6\})$.

Partial Equilibria. We recall partial equilibria [7], which informally are equilibria of a sub-MCS generated by a context C_k.

Definition 3 (Import Closure). *Let $M = (C_1, \ldots, C_n)$ be an MCS. The* import neighborhood *of a context C_k is the set $In(k) = \{c_i \mid (c_i : p_i) \in B(r), r \in br_k\}$. Moreover, the* import closure *$IC(k)$ of C_k is the smallest set S such that (i) $k \in S$ and (ii) for all $j \in S$, $In(j) \subseteq S$.*

Based on the import closure, we then define:

Definition 4 (Partial Belief States and Equilibria). *Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \mathbf{BS}_i$. A* partial belief state *of M is a sequence $S = (S_1, \ldots, S_n)$, such that $S_i \in \mathbf{BS}_i \cup \{\epsilon\}$, for $1 \leq i \leq n$. A partial belief state $S = (S_1, \ldots, S_n)$ of M is a* partial equilibrium *of M w.r.t. a context C_k iff $i \in IC(k)$ implies $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$, and if $i \notin IC(k)$, then $S_i = \epsilon$, for all $1 \leq i \leq n$.*

For instance, in our running example $(\epsilon, \epsilon, \{c_3, coke_3, peanuts_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{fit_6\})$ is a partial equilibrium w.r.t. context C_3.

For combining partial belief states $S = (S_1, \ldots, S_n)$ and $T = (T_1, \ldots, T_n)$, we define their *join* $S \bowtie T$ as the partial belief state $(U_1, \ldots, U_n)$ with (i) $U_i = S_i$, if $T_i = \epsilon \vee S_i = T_i$, and (ii) $U_i = T_i$, if $T_i \neq \epsilon \wedge S_i = \epsilon$, for all $1 \leq i \leq n$. Note that $S \bowtie T$ is void, if some S_i, T_i are from $\mathbf{BS}_i$ but different. The *join* of two sets $\mathcal{S}$ and $\mathcal{T}$ of partial belief states is then naturally defined as $\mathcal{S} \bowtie \mathcal{T} = \{S \bowtie T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$.

Given a (partial) belief state S and set $V \subseteq \Sigma$ of variables, the *restriction* of S to V, denoted $S|_V$, is given by $S = (S_1|_V, \ldots, S_n|_V)$, where $S_i|_V = S_i \cap V$ if $S_i \neq \epsilon$, and $\epsilon|_V = \epsilon$; for a set of (partial) belief states $\mathcal{S}$, we let $\mathcal{S}|_V = \{S|_V \mid S \in \mathcal{S}\}$.

Definition 5. *The* import interface *of context C_k in M is $V(k) = \{p \mid (c:p) \in B(r), r \in br_k\}$, and its* recursive import interface *is $V^*(k) = V(k) \cup \{p \in V(j) \mid j \in IC(k)\}$.*

As an example, the recursive import interface of C_1 in M from Ex. 2 is $V^*(1) = \{c_3, c_4, peanuts_3, coke_3, sick_6, t_2, t_3, t_4, sooner_5\}$.

There are two extremal cases: 1. $V = V^*(k)$. Then, partial equilibria projected to V can be basically used for consistency-checking on the import closure of C_k. 2. $V = \Sigma$. Here, the projection to V yields partial equilibria w.r.t. C_k. By providing a fixed interface V such that $V^*(k) \subseteq V \subseteq \Sigma$, problem-specific knowledge (e.g. query variables) and infrastructure information can be exploited to focus computations to relevant projections.

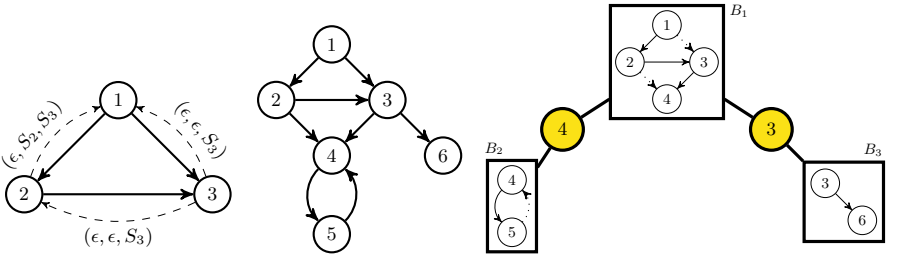
3 Decomposition of Nonmonotonic MCS

Reconsider our running example, with all contexts C_i and atoms referring to them removed, for $i > 3$. Then, C_1 has bridge rules with atoms of form $(2 : p_2)$ and $(3 : p_3)$ in the body, and C_2 with atoms $(3 : p_3)$. That is, C_1 depends on both C_2 and C_3, while C_2 depends on C_3 (see Fig. 1a). A straightforward approach to evaluate this modified MCS is to ask in C_1 for the belief sets of C_2 and C_3. But as C_2 also depends on C_3, we would need another query from C_2 to C_3 to evaluate C_2 w.r.t. the belief sets of C_3. This shows that there is some evident redundancy in this approach, as C_3 will need to compute its belief sets twice. Simple caching strategies could mellow out the second belief state building in C_3; nonetheless, when C_1 asks C_3, the context will transmit back its belief states, thus consuming network resources.

Moreover, when C_2 asks for the partial equilibria of C_3, it will receive a set of partial equilibria that covers the belief sets of C_3 and in addition all contexts in the import closure $IC(3)$. This is excessive from the view of C_1, as it only needs to know the truth of $(2 : p_2)$ and $(3 : p_3)$. However, C_1 needs the belief states of both C_2 and C_3 in reply of C_2: if C_2 only reports its own belief sets (which are consistent w.r.t. C_3), then C_1 has no chance to align the belief sets received from C_2 with those received from C_3. Realizing that C_2 also reports the belief sets of C_3, no call to C_3 must be made.

Based on this, we present an optimization strategy which pursues two orthogonal goals: (i) to prune dependencies in an MCS and cut superfluous transmissions, belief state building, and joining of belief states; and (ii) to minimize information in transmissions.

Graph-Theoretic Concepts. We start with defining the topology of an MCS.



(a) Triangle (b) Diamond-ring (c) Diamond-ring block tree

Fig. 1. Topologies and Decomposition of Scientist Group Example

Definition 6. *The* topology *of an MCS* $M = (C_1, \ldots, C_n)$ *is the digraph* $G_M = (V, E)$*, where* $V = \{1, \ldots, n\}$ *and* $(i, j) \in E$ *iff some rule in* br_i *has an atom* $(j{:}p)$ *in the body.*

The first optimization technique is made up of three graph operations. We get a coarse view of the topology by splitting it into *biconnected components*, which form a *tree representation* of the MCS. Then, edge removal techniques yield acyclic structures.

In the sequel, we will use standard terminology from graph theory (see [4]); graphs are directed by default. For any graph G and set $S \subseteq E(G)$ of edges, we denote by $G \backslash S$ the subgraph of G that has no edges from S. For a vertex $v \in V(G)$, we denote by $G \backslash v$ the subgraph of G induced by $V(G) \backslash \{v\}$. A graph is weakly connected if replacing every directed edge by an undirected edge yields a connected graph. A vertex c of a weakly connected graph G is a *cut vertex*, if $G \backslash c$ is disconnected. A *biconnected graph* is a weakly connected graph without cut vertices. A *block* in a graph G is a maximal biconnected subgraph of G. Let $T(G) = (\mathcal{B} \cup \mathcal{C}, \mathcal{E})$ denote the undirected bipartite graph, called *block tree of graph* G, where $\mathcal{B}$ is the set of blocks of G, $\mathcal{C}$ is the set of cut vertices of G, and $(B, c) \in \mathcal{E}$ with $B \in \mathcal{B}$ and $c \in \mathcal{C}$ iff $c \in V(B)$. Note that $T(G)$ is a rooted tree for any weakly connected graph G; for arbitrary graphs, it is a forest.

Example 3. The topology G_M of M in Ex. 2 is shown in Fig. 1b. It has two cut vertices, viz. 3 and 4; thus the block tree $T(G_M)$ (Fig. 1c) contains the blocks B_1, B_2, and B_3, which are subgraphs of G_M induced by $\{1, 2, 3, 4\}$, $\{4, 5\}$, and $\{3, 6\}$, respectively.

Pruning. In acyclic topologies, like the triangle presented in the previous section, we can exploit a minimal graph representation to avoid unnecessary calls between contexts. Namely, the *transitive reduction* of the graph G_M; recall that the transitive reduction of a digraph G is the graph G^- with the smallest set of edges whose transitive closure equals the one of G. Note that G^- is unique if G is acyclic.

Another essential part of our optimization strategy is to break cycles by removing edges from topologies. To this end, we use ear decompositions of cyclic graphs. A block may have multiple cycles which are not necessarily strongly connected, thus we first decompose cyclic blocks into their strongly connected components. The topological sort of these components yield a sequence of nodes $r_1, \ldots, r_s$ that are used as entry points to each component. The next step is to break cycles. An *ear decomposition* of a strongly connected graph G *rooted at a node* r is a sequence $P = \langle P_0, \ldots, P_m \rangle$ of

subgraphs of G such that (i) $G = P_0 \cup \cdots \cup P_m$, (ii) P_0 is a simple cycle (i.e., has no repeated edges or vertices) with $r \in V(P_0)$, and (iii) each P_i ($i > 0$) is a non-trivial path (without cycles) whose endpoints are in $P_0 \cup \cdots \cup P_{i-1}$, but the other nodes are not. Let $cb(G, P)$ be the set of edges containing (ℓ, r) from P_0 and the last edge (ℓ, t) from each P_i, $i > 0$.

Example 4. Block B_1 of $T(G_M)$ is acyclic, and the transitive reduction gives B_1^- with edges $\{(1,2),(2,3),(3,4)\}$. B_2 is cyclic, and $\langle B_2 \rangle$ is the only ear decomposition rooted at 4; removing $cb(B_2, \langle B_2 \rangle) = \{(5,4)\}$, we obtain B_2' with edges $\{(4,5)\}$. B_3 is acyclic and already reduced. Fig. 1c shows the final result (dotted edges are removed).

The graph-theoretic concepts introduced here, in particular the transitive reduction of acyclic blocks and the ear decomposition of cyclic blocks, are used to implement the first optimization of MCS evaluation outlined above. Intuitively, given the transitive reduction B^- of an acyclic block $B \in \mathcal{B}$, and a total order on $V(B^-)$ that extends B^-, one can evaluate the respective contexts in reverse order for computing partial equilibria at some context C_k: the first context simply computes its local belief sets which—represented as a set of partial belief states $\mathcal{S}_0$—constitutes an initial set of partial belief states $\mathcal{T}_0$. In any iterative Step i, $\mathcal{T}_{i-1}$ is updated by joining it with the local belief sets $\mathcal{S}_i$ of the context under consideration. Given $\mathcal{T}_k$ (after updating with $\mathcal{S}_k$) for context C_k, it holds that $\mathcal{T}_k|_{V^*(k)}$ is the set of partial equilibria at C_k (restricted to contexts in $V(B^-)$).

For cyclic blocks, one can in principle proceed as above; however, any context C_k accessing beliefs from a context C_i that precedes it in the given total order, has to temporarily consider all possible belief sets for C_i in $\mathcal{S}_k$. As a consequence, the above relation to partial equilibria can only be established after visiting all contexts that have been temporarily considered in previous steps.

Refined recursive import. Next, we define the second part of our optimization strategy which handles minimization of information needed for transmission between two neighboring contexts C_i and C_j. For this purpose, we refine the notion of recursive import interface in a context w.r.t. a particular neighbor, and a given (sub-)graph.

Definition 7. *Given an MCS $M = (C_1, \ldots, C_n)$ and a subgraph G of G_M, for an edge $(i,j) \in E(G)$, the* recursive import interface of C_i to C_j w.r.t. G *is* $V^*(i,j)_G = \{p \in V^*(i) \mid p \in \Sigma_\ell, j \text{ reaches } \ell \text{ in } G\}$.

Intuitively, if a context is a cut vertex c in G_M, one can drop all entries S_i ($i \neq c$) from the partial belief states computed at c, and pass this result to the parent block of c in $T(G_M)$, without compromising the computation of compatible (restricted) belief sets at the parent. Recursive import interfaces w.r.t. blocks in G_M reflect this property, which can be exploited for minimizing the information transmitted.

Algorithms. Alg. 1 and 2 combine the optimization techniques outlined above. Intuitively, OptimizeTree takes a block tree T as input together with parent cut vertex c_p and root cut vertex c_r. It traverses T in a DFS-way and calls OptimizeBlock on every block. The result of the latter calls are removed edges F; after all blocks have been processed, the final result of OptimizeTree is a pair of all edges removed from blocks in T, and a labelling v for the remaining edges. OptimizeBlock takes a graph G and calls subroutine CycleBreaker for cyclic G, which decomposes G into its strongly

Algorithm 1. OptimizeTree$(T = (\mathcal{B} \cup \mathcal{C}, \mathcal{E}), c_p, c_r)$

Input: T: block tree, c_p: identifies level in T, c_r: identifies level above c_p
Output: F: removed edges from $\bigcup \mathcal{B}$, v: labels for $(\bigcup \mathcal{B}) \backslash F$
$\mathcal{B}' := \emptyset,\ F := \emptyset,\ v := \emptyset$ // initialize siblings $\mathcal{B}'$ and return values
if $c_p = c_r$ **then** $\mathcal{B}' := \{B \in \mathcal{B} \mid c_r \in V(B)\}$ **else** $\mathcal{B}' := \{B \in \mathcal{B} \mid (B, c_p) \in \mathcal{E}\}$
foreach *sibling block* $B \in \mathcal{B}'$ **do** // sibling blocks B of parent c_p
  $E := \mathsf{OptimizeBlock}(B, c_p)$ // prune block
  $\mathcal{C}' := \{c \in \mathcal{C} \mid (B, c) \in \mathcal{E} \wedge c \neq c_p\}$ // children cut vertices of B
  $B' := B \backslash E,\ F := F \cup E$
  foreach *edge* (i, j) *of* B' **do** // setup interface of pruned B
    $v(i,j) := V^*(i,j)_{B'} \cup \bigcup_{c \in \mathcal{C}'} V^*(c_p)|_{\Sigma_c} \cup \bigcup_{(\ell,t) \in E} V^*(c_p)|_{\Sigma_t}$
  foreach *child cut vertex* $c \in \mathcal{C}'$ **do** // accumulate children
    $(F', v') := \mathsf{OptimizeTree}(T \backslash B, c, c_p)$
    $F := F \cup F',\ v := v \cup v'$
return (F, v)

Algorithm 2. OptimizeBlock$(G\colon \text{graph}, r\colon \text{context id})$

$F := \emptyset$
if G *is cyclic* **then** $F := \mathsf{CycleBreaker}(G, r)$ // ear decomp. of strong components
Let G^- be the transitive reduction of $G \backslash F$
return $E(G) \setminus E(G^-)$ // removed edges from G

connected components, creates an ear decomposition P for each component G_c, and breakes cycles by removing edges $cb(G_c, P)$. For the resulting acyclic subgraph of G (or if G was already acyclic), OptimizeBlock computes the transitive reduction G^-. All edges removed from G are returned. OptimizeTree continues computing the labelling v for the remaining edges, building on the recursive import interface, but keeping relevant interface variables of child cut vertices and removed edges. It can be shown that:

Proposition 1. *For any context C_k in an MCS M,* OptimizeTree$(T(G_M), k, k)$ *returns a pair (F, v) such that (i) the subgraph G of $G_M \backslash F$ induced by $IC(k)$ is acyclic, and (ii) for all $(i, j) \in E(G)$, $v(i, j) = V^*(i, j)_G$.*

Given G_M, the block tree graph $T(G_M)$ can be constructed in linear time; transitive reductions thereof can be computed in quadratic time. Since no other operation of the algorithm exceeds this bound, the following holds.

Proposition 2. *For any context C_k in an MCS M,* OptimizeTree$(T(G_M), k, k)$ *runs in time polynomial (quadratic) in the size of $T(G_M)$ resp. G_M.*

Given the topology of an MCS, we need to represent a stripped version of it which contains both the minimal dependencies between contexts and interface variables that need to be transferred between contexts. This representation will be a *query plan* that can be used for execution processing. Syntactically, query plans have the following form.

Definition 8 (Query Plan). *A* query plan of an MCS M w.r.t. context C_k *is any labeled subgraph* Π *of* G_M *induced by* $IC(k)$ *with* $E(\Pi) \subseteq E(G_M)$*, and edge labels* $v\colon E(G) \to 2^{\Sigma}$.

In particular, for any MCS M and context C_k of M, the labeled graph $\Pi_k = (V(G), E(G)\backslash F, v)$ is a query plan of M w.r.t. C_k, where G is the subgraph of G_M induced by $IC(k)$ and $(F, v) = \mathsf{OptimizeTree}(T(G_M), k, k)$. This query plan is in fact effective; we show how to use it for MCS evaluation.

4 Nonmonotonic MCS Evaluation with Query Plans

Given an MCS M and a starting context C_k, we aim at finding all projected partial equilibria of M w.r.t. C_k in a distributed way. To this end, we design an algorithm called DMCSOPT that is based on the algorithm DMCS in [7], but exploits properties of the optimization techniques described above.

As a by-product, we obtain a simplification, because explicit cycle breaking is not needed. At each context node, an instance of DMCSOPT runs independently and communicates with other instances for exchanging sets of partial belief states. This provides a method for distributed model building, such that DMCSOPT can be deployed to any MCS where appropriate solvers for the respective context logics are available. The main feature of DMCSOPT is that it computes projected partial equilibria based on a query plan. This can be exploited for specific tasks like, e.g., local query answering or consistency checking. When computing projected partial equilibria, the information communicated between contexts is minimized, keeping communication cost low.

In the sequel, we present a basic version of the algorithm, abstracting from low-level implementation issues. The idea is as follows: we start with context C_k and traverse a given query plan by expanding the outgoing edges of that plan at each context, like in a depth-first search, until a leaf context is reached. A leaf context C_i simply computes its local belief sets, transforms all belief sets into partial belief states, and returns this result to its parent. If the leaf C_i contains $(j : p)$ in bodies of bridge rules such that there is no context C_j to visit in the query plan—this means we broke a cycle by removing the last edge to C_j—, all possible truth assignments to the import interface to C_j are considered.

The result of any context C_i is a set of partial belief states, which amounts to the join, i.e., the consistent combination, of its local belief sets with the results of its neighbors; the final result is obtained from C_k. To keep re-computation and recombination of belief states with local belief sets at a minimum, partial belief states are cached in every context.

Alg. 3 shows our distributed algorithm, DMCSOPT, with its instance at a context C_k that runs in a background process (or daemon in Unix). On input of the id c of a predecessor context (which the process awaits), it proceeds based on an (acyclic) query plan Π_r w.r.t. context C_r, i.e., the starting context of the system. The algorithm maintains a cache $cache(k)$ at C_k, which is kept persistent by the background process. It uses the following helper functions:

- C_i.DMCSOPT(c): send id c to DMCSOPT at context C_i and wait for its result.
- guess(V): guess all possible truth assignments for the interface variables V.
- lsolve(S) (Alg. 4): given a partial belief state S, augment kb_k with all heads from bridge rules br_k applicable w.r.t. S (=: kb'_k), compute local belief sets by $\mathbf{ACC}(kb'_k)$, and merge them with S; return the resulting set of partial belief states.

Algorithm 3. DMCSOPT(c : context id of predecessor) at $C_k = (L_k, kb_k, br_k)$

Data: Π_r: query plan w.r.t. starting context C_r and label v, $cache(k)$: cache
Output: set of accumulated partial belief states

(a) **if** $cache(k)$ *is not empty* **then** $\mathcal{S} := cache(k)$ **else**
  $\mathcal{T} := \{(\epsilon, \ldots, \epsilon)\}$
(b) **foreach** $(k, i) \in E(\Pi_r)$ **do** $\mathcal{T} := \mathcal{T} \bowtie C_i$.DMCSOPT$(k)$ // neighbor beliefs
(c) **if** *there is* $i \in In(k)$ *s.t.* $(k, i) \notin E(\Pi_r)$ *and* $T_i = \epsilon$ *for* $T \in \mathcal{T}$ **then**
    $\mathcal{T} := \mathsf{guess}(v(c, k)) \bowtie \mathcal{T}$ // guess for removed dependencies in Π_r
(d) **foreach** $T \in \mathcal{T}$ **do** $\mathcal{S} := \mathcal{S} \cup \mathsf{lsolve}(T)$ // get local beliefs w.r.t. T
  $cache(k) := \mathcal{S}$
(e) **if** $(c, k) \in E(\Pi_r)$ (i.e., C_k is non-root) **then return** $\mathcal{S}|_{v(c,k)}$ **else return** $\mathcal{S}$

Algorithm 4. lsolve(S: partial belief state) at $C_k = (L_k, kb_k, br_k)$

Output: set of locally acceptable partial belief states
$\mathbf{T} := \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, S)\})$
return $\{(S_1, , \ldots, S_{k-1}, T_k, S_{k+1}, \ldots, S_n) \mid T_k \in \mathbf{T}\}$

The steps of Alg. 3 are explained as follows:

(a)+**(b)** check the cache, and if it is empty get neighbor contexts from the query plan, request partial belief states from all neighbors and join them;

(c) if there are $(i : p)$ in the bridge rules br_k such that $(k, i) \notin E(\Pi_r)$, and no neighbor delivered the belief sets for C_i in step (b) (i.e., $T_i = \epsilon$), we have to call guess on the interface $v(c, k)$ and join the result with $\mathcal{T}$: intuitively, this happens when edges had been removed from cycles;

(d) compute local belief states given the imported partial belief states collected from neighbors; and

(e) return the locally computed belief states and project to the variables in $v(c, k)$ for non-root contexts; this is the point were we mask out parts of the belief states that are not needed in contexts the lie in a different block of $T(G_M)$.

The following proposition shows that DMCSOPT is sound and complete.

Proposition 3. *Let C_k be a context of an MCS M, let Π_k be the query plan as defined above and let $V = \{p \in v(k, j) \mid (k, j) \in E(\Pi_k)\}$. Then, (i) for each $S' \in C_k$.DMCSOPT(k), there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$; and (ii) for each partial equilibrium S of M w.r.t. C_k, there exists an $S' \in C_k$.DMCSOPT(k) such that $S' = S|_V$.*

5 Implementation and Experimental Results

We present some results for a SAT-solver based prototype implementation of DMCSOPT under Ubuntu Linux 9.10, written in C++. Full details of the experiments and the implementation are available at `http://www.kr.tuwien.ac.at/research/systems/dmcs/`. The host system was using a Pentium Core2 Duo 2.53GHz processor with 4GB RAM. Based on generated benchmarks, we compare the average response time and the median of the number of results of DMCSOPT to our implementation

Table 1. Runtime for DMCSOPT (A_x) and DMCS (B_x), timeout 180 secs (—)

	n	A_ϕ	$A_{\bowtie}$	$A_{\leftrightarrow}$	A_Σ (σ)	# (σ)	B_ϕ	$B_{\bowtie}$	$B_{\leftrightarrow}$	B_Σ (σ)	# (σ)
D	13	0.9	0.0	0.0	1.0 (0.2)	28 (17.6)	0.8	8.4	0.0	9.4 (5.5)	3136 (3155.8)
	25	11.2	0.5	0.0	12.8 (1.3)	17 (18.9)	—				
	31	51.1	3.7	0.0	59.5 (8.9)	58 (49.7)	—				
R	10	0.1	0.0	0.0	0.1 (0.0)	3.5 (3.4)	0.1	0.0	0.0	0.2 (0.1)	300 (694.5)
	13	0.1	0.0	0.0	0.2 (0.1)	6 (1.2)	0.1	1.5	1.9	3.9 (5.3)	5064 (21523.8)
	301	4.1	0.1	2.1	10.2 (2.2)	8 (4.9)	—				
Z	13	0.6	0.1	0.0	0.7 (0.2)	34 (41.8)	5.5	4.2	0.0	11.5 (4.0)	3024 (1286.8)
	151	8.9	22.3	0.4	32.2 (7.3)	33 (28.5)	—				
	301	21.6	99.5	1.7	124.3 (20.6)	22 (41.4)	—				
H	9	0.2	0.0	0.0	0.2 (0.0)	28 (44.4)	1.1	0.9	0.0	2.0 (1.3)	684 (1308.0)
	101	1.8	0.3	0.3	3.8 (1.0)	48 (76.6)	—				
	301	7.8	2.0	2.4	25.1 (8.7)	38 (34.2)	—				

of DMCS. The processes encoding the contexts communicated over local TCP/IP connections. We used *clasp* 1.3.3 as a SAT solver, which accepts DIMACS CNF input [8]. Specifically, all generated instantiations of MCS have contexts with ASP logics. The translation defined in [7] is used to create SAT instances at all contexts and *clasp* builds all models.

For initial experimentation, we created random MCS instances with various fixed topologies that should resemble the context dependencies of realistic scenarios. We have generated instances with ordinary (D) and zig-zag (Z) diamond stack, house stack (H), ring (R), and binary tree topologies. A diamond stack combines multiple diamonds in a row (stacking m diamonds in a tower of $3m + 1$ contexts). Ordinary diamonds have, in contrast to zig-zag diamonds like block B_1 in Fig. 1c, no connection between the two middle contexts. A house consists of 5 nodes with 6 edges (the ridge context has directed edges to the two middle contexts, which form with the two base contexts a cycle with 4 edges); house stacks are subsequently built up by using the basement nodes as ridges for the next houses (thus, m houses have $4m+1$ contexts). Binary trees grow balanced, i.e., every level is complete except for the last level, which grows from the left-most context.

A parameter setting (n, s, b, r) specifies (i) the number n of contexts, (ii) the local alphabet size $|\Sigma_i| = s$ (each C_i has a random ASP program on s atoms with 2^k answer sets, $0 \leq k \leq s/2$), (iii) the maximum interface size b (number of atoms exported), and (iv) the maximum number r of bridge rules per context, each having ≤ 2 body literals. Table 1 shows some experimental results for parameter settings $(n, 10, 5, 5)$, where n varies between 10 and 301. Each subtable $X \in \{D, H, R, Z\}$ shows runs with growing number of contexts and correspond to a benchmark topology from above. Each row displays the average of total running time over 10 generated instances for DMCSOPT in A_Σ compared to DMCS in B_Σ. For each context in the instances, the columns A_x and B_x for $x \in \{\phi, \bowtie, \leftrightarrow\}$ show the average over the (i) total time spent in the SAT solver (ϕ), (ii) total time needed to join the belief states ($\bowtie$), and (iii) total time used

to transfer the belief states between the contexts ($\leftrightarrow$). The # columns show the median of numbers of projected partial equilibria computed at C_1 (initiated by sending the request 1 to C_1 for DMCSOPT with a fixed query plan Π_1, respectively $V^*(1)$ to C_1 for DMCS). Entries in parenthesis (σ) show the standard deviation of A_Σ and #.

The optimizations that can be applied in the topologies are quite diverse. In ordinary diamond stacks and in binary trees, we cannot remove edges, as the topologies are equal to their transitive reductions. But we can refine the import interface at each sub-diamond (every fourth context is a cut vertex), thus the partial belief states eventually computed just contain entries for the first four contexts. House stacks have a triangle element as roof (Fig. 1a) and a ring as walls, thus two connections are pruned which results in chains of contexts. As the two basement contexts are cut vertices, the final belief states contain only belief sets for the 5 contexts in the top-most house. The refinement of the import interface in binary trees is even more drastic, as every non-leaf context is a cut vertex, and we can restrict to import interfaces between two neighboring contexts. In the ring topology, we can remove the last edge closing the cycle to context C_1. As the resulting topology is a spanning tree, the refinement of the import interface is restricted to neighboring contexts including the import interface of the removed edge. In zig-zag diamond stacks, we remove in each block two edges to obtain the transitive reduction and update the recursive import interface accordingly.

Evaluating the MCS instances with DMCSOPT compared to DMCS yields a drastic improvement in response time. Stacking multiple diamonds in a tower models hard instances with many joins. This is reflected in the ratio of running time to result size in DMCS. Still, DMCSOPT could handle much larger instances. The ring topology shows a similar increase in scalability. Thanks to the refined interface, the system size can be increased dramatically; the runs for n=301 took only a few seconds. House instances use a complex topology with cycles in each block, and each cycle has two entry points. DMCS is already having a hard time evaluating instances with two houses, whereas DMCSOPT could evaluate ten houses in a reasonable time due to the localized computation of the belief states. Also for zig-zag diamond stacks, the optimizations effect that DMCSOPT can run substantially larger systems, with hundreds of contexts; DMCS has an early breakdown at n=16.

Comparing ordinary diamonds (D) to zig-zag diamonds (Z), one can notice a large gap in the size n of the MCS that can be handled. This is explained by the transitive reduction that can be applied to zig-zag diamonds, essentially resulting in a chain of contexts such that each context can take the partial belief states of its single neighbor and simply add its local beliefs. Diamonds cannot be further optimized and additionally need to join the results from their neighbors. We omit detailed outcomes for binary tree topology tests here. However, we noticeably could evaluate an instance with even n=600 contexts in 175.6 seconds (#=4) with our parameter setting; setting the timeout to 12 minutes, DMCS runs out of memory for instances with n=22 during belief state joining.

The A_ϕ/B_ϕ and $A_{\bowtie}/B_{\bowtie}$ columns show that most time is spent in the SAT solver in D, R, and H instances, whereas Z uses most time in combining the models. This is explained by the need to guess many interface variables in those instances, as we need to cater for the interface of the removed edges. $A_{\hookrightarrow}$ reveals that almost no time is used to transfer the belief states, even as n grows, thus showing the effectiveness of the

projection over blocks, as only a small amount of models have to be shipped. In $B_{\leftrightarrow}$ we can see that already small instances produce many models and the price is that the network is under heavy load.

6 Related Work and Conclusion

In [13], the authors described evaluation of monotone MCS with classical theories using SAT solvers for the contexts in parallel. They used a (co-inductive) fixpoint strategy to check MCS satisfiability, where a centralized process iteratively combines results of the SAT solvers. Apart from being not truly distributed, an extension to nonmonotonic MCS is non-obvious; also, no caching was used. Distributed tableaux algorithms for reasoning in distributed ontologies are defined in [14,15]. They can be used to decide consistency of distributed description logic knowledge bases, provided that the distributed TBox is acyclic. The DRAGO system is an implementation of this approach.

The authors of [1] presented a framework of peer-to-peer inference systems. Local theories of propositional clause sets share atoms, and a special algorithm can be used for consequence finding. As we pursue the dual problem of model building, application for our needs is not straightforward. Similarly, [3] developed a distributed algorithm for query evaluation in a MCS framework based on defeasible logic. Here, contexts are built using defeasible rules, and the algorithm can determine for a given literal l three values: whether l is (not) a logical conclusion of the MCS, or whether it cannot be proved that l is a logical conclusion. Again, applying this approach to model building is not easy.

Biconnected components are used in [2] to decompose constraint satisfaction problems. The decomposition is used to localize the computation of a single solution in the components of undirected constraint graphs. Likened to our approach, we are based on directed dependencies, which allows us to use a query plan for MCS evaluation.

We have presented techniques and algorithms for decomposing, pruning, and cycle breaking of dependencies in nonmonotonic multi-context systems. Based on this, we have devised an algorithm, which uses a query plan to compute all partial equilibria of such a system. A prototypical implementation of this approach shows promising experimental results. They are a substantial improvement and encourage to research further algorithms and methods for evaluation of distributed MCS, such that efficient platforms for distributed nonmonotonic reasoning applications will become available.

References

1. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Distributed reasoning in a peer-to-peer setting: Application to the semantic web. J. Artif. Intell. Res. 25, 269–314 (2006)
2. Baget, J.F., Tognetti, Y.: Backtracking through biconnected components of a constraint graph. In: IJCAI 2001, pp. 291–296. Morgan Kaufmann, San Francisco (2001)
3. Bikakis, A., Antoniou, G., Hassapis, P.: Strategies for contextual reasoning with conflicts in ambient intelligence. Knowl. Inf. Syst. (April 2010) (published online: April 9, 2010)
4. Bondy, A., Murty, U.S.R.: Graph Theory. Springer, Heidelberg (2008)

5. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI 2007, pp. 385–390. AAAI Press, Menlo Park (July 2007)
6. Buccafurri, F., Caminiti, G.: Logic programming with social features. Theory Pract. Log. Program. 8(5-6), 643–690 (2008)
7. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Distributed nonmonotonic multi-context systems. In: KR 2010, pp. 60–70. AAAI Press, Menlo Park (2010)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007, pp. 386–392. AAAI Press, Menlo Park (2007)
9. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. 9(3-4), 365–385 (1991)
10. Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning = locality + compatibility. Artif. Intell. 127(2), 221–259 (2001)
11. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. Artif. Intell. 65(1), 29–70 (1994)
12. McCarthy, J.: Notes on formalizing context. In: IJCAI 1993, pp. 555–562 (1993)
13. Roelofsen, F., Serafini, L., Cimatti, A.: Many Hands Make LightWork: Localized Satisfiability for Multi-Context Systems. In: ECAI 2004, pp. 58–62. IOS Press, Amsterdam (2004)
14. Serafini, L., Borgida, A., Tamilin, A.: Aspects of distributed and modular ontology reasoning. In: IJCAI 2005, pp. 570–575. AAAI Press, Menlo Park (2005)
15. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 361–376. Springer, Heidelberg (2005)

Bridging Possibilistic Conditional Knowledge Bases and Partially Ordered Bases

Salem Benferhat, Sylvain Lagrue, and Safa Yahi

Université Lille-Nord de France
Artois, F-62307 Lens, CRIL, F-62307 Lens
CNRS UMR 8188, F-62307 Lens
{benferhat,lagrue,yahi}@cril.fr

Abstract. Possibilistic logic offers a unified framework for revising prioritized pieces of information and for reasoning with conditional knowledge bases. A conditional assertion of the form "generally, if α is true then β is true" is interpreted as a constraint expressing that the possibility degree of having $\alpha \wedge \beta$ being true is greater than the possibility degree of having $\alpha \wedge \neg\beta$ being true. Recently, an important extension of possibilistic logic has been proposed to deal with partially pre-ordered bases in order to avoid comparing unrelated pieces of information.

This paper establishes relationships between reasoning from partially pre-ordered bases and reasoning from conditional knowledge bases. It contains two important contributions. The first contribution consists in identifying conditions under which a partially ordered belief base can be encoded as a set of conditional assertions, and conversely. In particular, we provide the correspondences between the concept of compatible possibility distributions used for conditional assertions and the one of compatible prioritized bases used for partially pre-ordered bases. The second important contribution of this paper consists in providing the computational complexity of reasoning with partially pre-ordered bases using the well-known possibilistic and inclusion-based policies.

1 Introduction

It is well known that one of the major purposes of nonmonotonic reasoning is to cope with the presence of exceptions in knowledge based systems. Some emphasis has been put on the application of nonmonotonic reasoning techniques to practical problems. For instance, in [15] several potential domains of applications, like medical reasoning, legal reasoning and reasoning in business organizations, have been identified.

A number of approaches for nonmonotonic reasoning can be found in the literature. Reiter's default logic [19] is among the best known and most widely studied logical frameworks for reasoning with rules having exceptions. Inferences are obtained by first defining a concept of extensions (composed of propositional formulas derived from default rules), and then applying propositional inference on these extensions. Moreover there are different efficient algorithms that have

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 38–50, 2010.
© Springer-Verlag Berlin Heidelberg 2010

been used to compute extensions and defaults inferences (e.g., [12,13,14]). There exist other approaches more normative like System P [10] and rational closure [11]. This paper considers this framework using possibility theory.

A conditional knowledge base T is a set of default rules having exceptions, simply called here conditional assertions. This paper focuses on the possibilistic handling of conditional bases proposed in [2] that consists in viewing each conditional assertion $\alpha \rightarrow \beta$ as a constraint expressing that the situation where α and β is true has a greater possibility than the one where α and $\neg\beta$ is true. This statement is expressed in a possibility theory framework [22,7] by $\alpha\wedge\beta <_\Pi \alpha\wedge\neg\beta$ where $<_\Pi$ is a plausibility ordering between formulas. Hence, a conditional base T can be viewed as a restricting a family $\prod(T)$ of possibility distributions, where a possibility distribution is a function from a set of universe of discourse to the unit interval [0,1]. Selecting a possibility distribution from $\prod(T)$ using the minimum specificity principle is equivalent to System Z [11].

Recently, several approaches [3,21] have been proposed to reason from partially pre-ordered belief bases using possibilistic logic or the well-known lexicographic based approaches. Partially pre-ordered belief bases offer much more flexibility in order (compared to totally pre-ordered bases) to efficiently represent incomplete knowledge and to avoid comparing unrelated pieces of information. Indeed, in many applications, the priority relation associated with available beliefs is only partially defined and forcing the user to introduce additional unwanted priorities may lead to infer undesirable conclusions.

Reasoning from partially pre-ordered belief bases also comes down to reason with a family of "compatible" totally pre-ordered knowledge base. A compatible totally pre-ordered (or prioritized) base represents a possible completion (i.e., by relating incomparable formulas) of a partially pre-ordered belief base.

This paper establishes on one side connections between reasoning from partially pre-ordered bases and reasoning from conditional knowledge bases, and on the other side provides complexity result of an important inference relation that deals with partially pre-ordered belief bases. More precisely, this paper contains two important contributions. The first contribution consists in identifying conditions under which a partially ordered belief bases can be encoded as a set of conditional assertions, and conversely. In particular, we provide the correspondences between the concept of compatible possibility distribution used for conditional assertions and the one of compatible prioritized bases used for partially ordered bases.

The second important contribution consists in providing the computational complexity of reasoning with partially pre-ordered bases using inclusion-based policies. In particular, we show that dealing with partially pre-ordered belief bases is equivalent to deal with partially ordered bases, obtained by replacing equally reliable formulas by one formula representing their conjunction.

The rest of the paper is structured as follows. Section 2 gives a brief refresher on inference from partially ordered belief bases. Section 3 recalls possibilistic handling of conditional bases. In section 4, we give the relation between conditional bases and partially pre-ordered belief bases interpreted using possibilistic

inference. We give in Section 5 a tight complexity results of compatible-based inclusion inference and conclude by Section 6.

2 Brief Refresher on Inference from Partially Ordered Belief Bases

2.1 Notations

We consider a finite set of propositional variables V where its elements are denoted by lower case Roman letters $a, b, \ldots$ The symbols $\top$ and $\bot$ denote tautology and contradiction respectively. Let PL_V be the propositional language built from V, $\{\top, \bot\}$ and the connectives $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ in the usual way. Formulas, i.e., elements of PL_V are denoted by Greek letters $\phi, \varphi, \psi, \ldots$ The set of formulas are denoted by the upper case Roman letters $A, B, \ldots$ The symbol $\vdash$ denotes classical inference relation. Let Σ be a finite set of formulas, $Cons(\Sigma)$ denotes the set of all consistent subbases of Σ while $MCons(\Sigma)$ denotes the set of all maximally consistent subbases of Σ. If Σ is consistent, then $MCons(\Sigma)$ contains one element which is Σ. The set of interpretations is denoted by Ω. Let ϕ be a propositional formula, $Mod(\phi)$ denotes the set of models of ϕ, namely $Mod(\phi) = \{\omega \in \Omega : \omega \models \phi\}$.

A partial pre-order $\preceq$ on a finite set A is a reflexive and transitive binary relation. In this paper, $a \preceq b$ expresses that a is at least as preferred as b. A strict order $\prec$ on A is an irreflexive and transitive binary relation. $a \prec b$ means that a is strictly preferred to b. A strict order is defined from a pre-order as $a \prec b$ if and only if $a \preceq b$ holds but $b \preceq a$ does not hold.

We assume that the reader is familiar with some basic notions about complexity theory, like the classes P, NP and co-NP. Now, we will sketch the classes of the polynomial hierarchy (PH) (see [17] for more details). Let X be a class of decision problems. Then P^X denotes the class of decision problems that can be solved using a polynomial algorithm that uses an oracle for X (informally, a subroutine for solving a problem in X at unit cost). Similarly, NP^X denotes the class of decision problems that can be solved using a nondeterministic polynomial algorithm that uses an oracle for X. Based on these notions, the classes Δ_k^p, Σ_k^p and Π_k^p $(k \geq 0)$ are defined as follows:

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = P$
- $\Delta_{k+1}^p = P^{\Sigma_k^p}$
- $\Sigma_{k+1}^p = NP^{\Sigma_k^p}$
- $\Pi_{k+1}^p = \text{co}\Sigma_{k+1}^p$.

In particular, Δ_1^P =P, Σ_1^p =NP and Π_1^p = coNP.

2.2 Qualitative Possibility Distribution

One of the basic object of possibility theory [22,8] is the *possibility distribution*, which is simply viewed here as a total pre-order, denoted by $\leq_\pi$, on the set of

interpretations . A possibility distribution $\preceq_\pi$ represents the available knowledge about what the real world is. In fact, $\omega \leq_\pi \omega'$ means that ω is at least as plausible (preferred) as ω'. In this case, Ω can be stratified or viewed as a well ordered partition (WOP) as follows $\Omega = E_1 \cup E_1 \cup \ldots \cup E_m \cup E_\perp$ where E_1 represents totally possible interpretations while $E_\perp$ represents fully impossible ones. Moreover, $\forall i, 1 \leq i \leq m$, $\forall j, 1 \leq j \leq m$, if $i < j$ then the interpretations of E_i are more plausible than those of E_j.

In possibility theory, $\leq_\pi$ induces two pre-orders grading respectively the possibility and the certainty of formulas:

- The possibility ordering between formulas of the language: $\varphi <_\Pi \psi$ iff $\exists \omega \in Mod(\varphi)$ such that $\omega \in E_i$ and $\forall \omega' \in Mod(\psi)$, if $\omega' \in E_j$ then $i < j$. Namely, φ is more plausible than ψ iff there exists a model of φ which is more plausible than any model of ψ.
- The certainty (or necessity) ordering between formulas of the language is defined from the possibility one as follows: $\varphi <_N \psi$ iff $\neg\psi <_\Pi \neg\varphi$

2.3 Inference from Totally Pre-ordered Inconsistent Belief Bases

A totally pre-ordered belief base $(\Sigma, \leq)$ is a set of propositional formulas Σ equipped with a total pre-order $\leq$. $(\Sigma, \leq)$ can be viewed as a stratified belief base $\Sigma = S_1 \cup \cdots \cup S_m$ such that the formulas in S_i have the same level of priority and have a higher priority than those in S_j with $j > i$. Note that only somewhat certain formulas are present in Σ (namely, all formulas in Σ are accepted to some extent).

Reasoning under inconsistency represents a fundamental problem that arises in many situations including exceptions tolerant reasoning, belief revision, integrating pieces of information coming from different possibly conflicting sources, reasoning with uncertainty or from incomplete information, etc. In this case, classical inference cannot be directly used since from an inconsistent base every formula can be inferred (ex falso quodlibet sequitur principle).

Many approaches have been proposed in order to reason under inconsistency without trivialization. While some of them consist in weakening the inference relation such as paraconsistent logics [6], others weaken the available beliefs like the so-called coherence-based approaches which are quite popular. Most of the coherence-based approaches [20] are defined with respect to totally pre-ordered belief bases like

- the possibilistic inference [7],
- the inclusion-based inference [9],
- the lexicographic inference [1].

These inference relations are defined respectively with respect to the following preference relations between consistent subbases of Σ which is denoted in the following by $Cons(\Sigma)$.

Besides, Let $A, B \in Cons(\Sigma)$.

- A is at least as preferred as B with respect to the **possibilistic** or the **best-out** preference, denoted by $A \leq_\pi B$, iff $m(B) \leq m(A)$ where $m(X) = min\{i$ such that $\exists \varphi \in S_i \setminus X\}$ for any consistent subbase X.
- A is **lexicographically** preferred to B, denoted by $A <_{lex} B$, iff $\exists i$, $1 \leq i \leq m$ such that $|S_i \cap A| > |S_i \cap B|$ [1] and $\forall j$, $j < i$, $|S_j \cap B| = |S_j \cap A|$.
- A is preferred to B with respect to the **inclusion preference**, denoted by $A <_{incl} B$, iff $\exists i$, $1 \leq i \leq m$ such that $(S_i \cap B) \subset (S_i \cap A)$ and $\forall j$, $j < i$, $(S_j \cap B) = (S_j \cap A)$.

Example 1. Let $(\Sigma, \leq) = (S_1, S_2, S_3)$ be a stratified belief base such that:

- $S_1 = \{r \wedge q \wedge e\}$
- $S_2 = \{\neg r \vee \neg p, \neg q \vee p, \neg e \vee p\}$
- $S_3 = \{\neg e \vee l\}$

Let us consider three consistent subbases A, B and C such that:

- $A = \{r \wedge q \wedge e, \neg q \vee p, \neg e \vee p\}$,
- $B = \{r \wedge q \wedge e, \neg q \vee p, \neg e \vee l\}$,
- $C = \{\neg q \vee p, \neg e \vee p, \neg e \vee l\}$.

Clearly, we have $m(A) = 2$, $m(B) = 2$ and $m(C) = 1$. So, $A =_\pi B <_\pi C$.

Moreover, we have

- $A \cap S_1 = B \cap S_1$,
- $B \cap S_2 \subset A \cap S_2$.

Then, we deduce that $A <_{incl} B$.

Besides, since we have

- $|A \cap S_1| = |B \cap S_1|$
- $|B \cap S_2| < |A \cap S_2|$

we conclude that $A <_{lex} B$.

Definition 1. *Let ψ be a formula and $(\Sigma, \leq)$ be a totally pre-ordered belief base. Then, ψ is said to be a* ***possibilistic*** *(resp. a* ***lexicographic****, an* ***inclusion-based****) consequence of $(\Sigma, \leq)$, denoted by $(\Sigma, \leq) \vdash_\pi \psi$ (resp. $(\Sigma, \leq) \vdash_{lex} \psi$, $(\Sigma, \leq) \vdash_{incl} \psi$) , if and only if $\forall B \in Pref(Cons(\Sigma), <_\pi) : B \models \psi$ (resp. $\forall B \in Pref(Cons(\Sigma), <_{lex}) : B \models \psi$, $\forall B \in Pref(Cons(\Sigma), <_{incl}) : B \models \psi$).*

2.4 Inference from Partially Pre-ordered Belief Bases

Now, given a partially pre-ordered belief base $(K, \prec)$, the three previous inference relations have been extended to the case of partially ordered belief bases. Some of these extensions rely on the notion of what we call here POB-compatible [3] where POB stands for Partially pre-Ordred Bases.

[1] $|A|$ denotes the number of formulas of A.

Definition 2. *Let $(K, \prec)$ be a partially ordered belief base. A totally pre-ordered belief base $(K, \leq)$ is said to be POB-compatible with $(K, \prec)$ if and only if:*

$$\forall \varphi, \phi \in K : \text{if } \varphi \prec \phi \text{ then } \varphi < \phi.$$

Then, the possibilistic (resp. the inclusion-based, lexicographic) extension amounts to apply the possibilistic inference (resp. inclusion, lexicographic inference) on all the POB-compatible bases with the partially ordered belief base at hand.

Example 2. Let us consider a partially pre-order belief base $(\Sigma, \preceq)$ such that:

$$\Sigma = \{R \wedge Q \wedge E, \neg E \vee S, \neg Q \vee P, \neg R \vee \neg P, \neg E \vee P, \neg S \vee L\}.$$

The pre-order $\preceq$ is given by Figure 1.

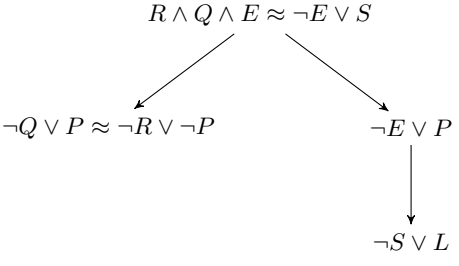


Fig. 1. Partial pre-order $\preceq$ over Σ

The base $(\Sigma, \preceq)$ has five compatible totally pre-ordered bases: $(\Sigma, \leq^1), \ldots, (\Sigma, \leq^5)$ such that:

- $(\Sigma, \leq^1) = (\{R \wedge Q \wedge E, \neg E \vee S\}, \{\neg Q \vee P, \neg R \vee \neg P\}, \{\neg E \vee P\}, \{\neg S \vee L\})$,
- $(\Sigma, \leq^2) = (\{R \wedge Q \wedge E, \neg E \vee S\}, \{\neg Q \vee P, \neg R \vee \neg P, \neg E \vee P\}, \{\neg S \vee L\})$,
- $(\Sigma, \leq^3) = (\{R \wedge Q \wedge E, \neg E \vee S\}, \{\neg E \vee P\}, \{\neg Q \vee P, \neg R \vee \neg P\}, \{\neg S \vee L\})$,
- $(\Sigma, \leq^4) = (\{R \wedge Q \wedge E, \neg E \vee S\}, \{\neg E \vee P\}, \{\neg S \vee L, \neg Q \vee P, \neg R \vee \neg P\})$,
- $(\Sigma, \leq^5) = (\{R \wedge Q \wedge E, \neg E \vee S\}, \{\neg E \vee P\}, \{\neg S \vee L\}, \{\neg Q \vee P, \neg R \vee \neg P\})$.

One can easily check that $\forall i, 1 \leq i \leq 5$, we have, for instance, $(\Sigma, \leq^i) \vdash_\pi S$. Thus, we deduce that S is a consequence of $(\Sigma, \preceq)$ with respect to the extension of possibilistic inference.

3 Possibilistic Handling of Conditional Assertions

By a conditional assertion we mean a generic rule of the form "generally, if α then β" having possibly some exceptions. These rules are denoted by "$\alpha \rightarrow \beta$". A default base is a set $T = \{\alpha_i \rightarrow \beta_i : 1 \leq i \leq n\}$ of default rules.

In [2], it has been proposed to view each conditional assertion $\alpha \rightarrow \beta$ as a constraint expressing that the situation where α and β is true has a greater possibility than the one where α and $\neg\beta$ is true which is expressed in possibility theory by the strict inequality $\alpha \wedge \beta <_\Pi \alpha \wedge \neg\beta$.

Hence, a default theory T can be viewed as a family of constraints restricting a family $\prod(T)$ of possibility distributions. Elements of $\prod(T)$ are said to be compatible with T and are defined as:

Definition 3. *A possibility distribution $<_\pi$ is said to be compatible with a conditional base T, $\pi \in \prod(T)$, iff for each default rule $\alpha_i \rightarrow \beta_i$ of Δ: $\alpha_i \wedge \beta_i <_\Pi \alpha_i \wedge \neg\beta_i$, where $<_\Pi$ is the possibility ordering induced by $<_\pi$.*

A first way of defining a nonmonotonic consequence relation consists in considering all the possibility distributions of $\prod(\Upsilon)$, namely a conditional assertion $\alpha \rightarrow \beta$ is said to be a universal possibilistic consequence of Υ, denoted by $\Upsilon \models_{\forall\Pi} \alpha \rightarrow \beta$, iff $\alpha \wedge \beta <_\Pi \alpha \wedge \neg\beta$ for each possibility distribution $<_\pi$ from $\prod(\Upsilon)$.

It has been shown in [2] that the universal possibilistic inference relation $\models_{\forall\pi}$ is equivalent to System P (P as Preferential) which is a set of postulates encoded by a reflexivity axiom and five inference rules namely Left Logical Equivalence, Right Weakening, Or, Cautious Monotony and Cut [10]. In [4], the authors establishes the retionships between three formats of compact representations of possibility distributions : conditional-based representation, possibilistic-logic based representation and graphical-based representation.

However, the universal possibilistic consequence relation, even if it produces acceptable and safe conclusions, is very cautious. One way of coping with the cautiousness of $\models_{\forall\pi}$ is to pick only one possibility distribution among those in $\prod(\Upsilon)$. It is what it is done when the so-called rational monotony property [11] is added to System P.

To this effect, the minimum specificity principle can be used [2]. Let us consider two possibility distributions $<_\pi$ and $<'_\pi$ such that:

- $<_\pi = E_1 \cup \ldots \cup E_m \cup E_\perp$,
- $<'_\pi = F_1 \cup \ldots \cup F_n \cup F_\perp$.

Then, $<_\pi$ is said to be less specific than $<'_\pi$ iff:

- $F_\perp \subseteq E_\perp$,
- $\forall i, 1 \leq i \leq n$, $\forall \omega \in F_i$, $\exists E_j$ such that $\omega \in E_j$ and $j < i$.

Now, the MSP-entailment (MSP as minimum specificity principle) which is equivalent to System Z is defined as follows:

Definition 4. *Let π be the possibility distribution selected from $\prod(\Upsilon)$ using the minimum specificity principle. Then, a conditional assertion $\alpha \rightarrow \beta$ is said to be a MSP-consequence of Υ, denoted by $\Upsilon \models_{MSP} \alpha \rightarrow \beta$, iff $\alpha \wedge \beta <_\Pi \alpha \wedge \neg\beta$.*

4 On the Relation between Conditional Bases and Partially Ordered Belief Base

Clearly, one common feature of reasoning from partially ordered belief bases and conditional knowledge bases is that both of them use the concept of compatible bases or distributions. The main difference is that the concept of compatible is defined on interpretations for conditional knowledge bases while it is defined on formulas for partially pre-ordered belief bases (by means of totally pre-ordered bases).

4.1 Preliminary Results

Recall that reasoning from partially pre-ordered knowledge bases is based on the concept of POB-compatible totally pre-ordered bases. A natural question is how to provide something similar to POB-compatibles but at semantical level (namely, defined on interpretations) by means of a class of possibility called π-compatible.

Then, we introduce the following definition:

Definition 5. *A qualitative possibility distribution $<_\pi$ on Ω is said to be π-compatible with $(K, \prec)$ if and only if:*

- $\forall \varphi, \phi \in K$: *if* $\varphi \prec \phi$ *then* $\varphi <_N \phi$ *where* $<_N$ *is the necessity ordering associated with* $<_\pi$,
- $E_1 = Mod(K)$, *namely models of* K *are the most preferred ones.*

So, a π-compatible distribution preserves the ordering between formulas of a partially pre-ordered knowledge base. In order to compare POB-compatible bases and π-compatible distributions, we need to define possibility distributions associated with totally pre-ordered bases. Namely, given a totally ordered belief base $(K, <)$, so the associated qualitative possibility distribution $<_\pi$ is defined by : $\forall \omega, \omega' \in \Omega$, $\omega <_\pi \omega'$ iff $\exists \varphi \in K$ such that $\omega' \not\models \varphi$ and $\forall \psi$ such that $\omega \not\models \psi$, we have $\varphi < \psi$.

Let $(K, \prec)$ be a partially ordered belief base. In the following, $F_\pi(K, \prec)$ denotes the set of all π-compatible of $(\Sigma, \prec)$, namely

$$F_\pi(K, \prec) = \{<_\pi \text{ such that } <_\pi \text{ is } \pi\text{-compatible with } (K, \prec)\}.$$

Besides, $POB(K, \prec)$ denotes the set of all qualitative possibility distributions associated with totally ordered belief bases which are compatible with $(K, \prec)$, that is

$$POB(K, \prec) = \{<_\pi \text{ such that } <_\pi \text{ is associated with the POB-compatible of } (K, \prec)\}.$$

Hence, we have (see also Figure 2):

Lemma 1. *Let $(K, \prec)$ be a partially ordered belief base. Then, $POB(K, \prec) \subseteq F_\pi(K, \prec)$.*

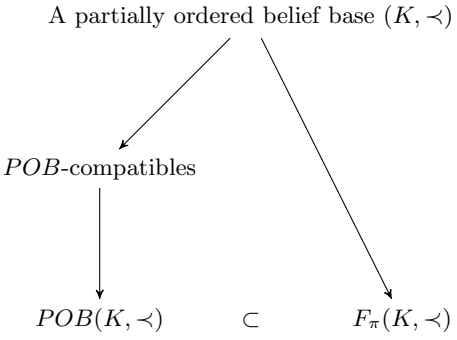


Fig. 2.

The converse does not hold. Indeed, let us consider the following example.

Example 3. Let us consider the belief base $(K, \prec) = \{p, q\}$ with $p \prec q$. In this case, K has a single POB-compatible base $(K, <) = \{p, q\}$ such that $p < q$. However, $<_\pi$ defined by:

$$pq <_\Pi p\neg q <_\Pi \neg pq <_\Pi \neg p\neg q$$

is π-compatible, but does not belong to $POB(K, \prec)$.

4.2 From Conditional Bases to Partially Ordered Bases

In this section, we first point out that a conditional base can also be viewed as partially ordered belief base. In fact, a conditional base $T = \{p_i \rightarrow q_i, 1 \leq i \leq n\}$ can be viewed as a set of constraints under the form :

$$\{p_i \wedge q_i >_\Pi p_i \wedge \neg q_i, 1 \leq i \leq n\}$$

or equivalently

$$\{\neg p_i \vee q_i <_N \neg p_i \vee \neg q_i, 1 \leq i \leq n\}.$$

Subsequently, a conditional knowledge base can be represented by a partially ordered belief base of the form : $\{\neg p_i \vee q_i \prec \neg p_i \vee \neg q_i, 1 \leq i \leq n\}$.

Namely:

Proposition 1. *Let $T = \{\alpha_i \rightarrow \beta_i, 1 \leq i \leq n\}$ be a conditional base. Let $(K_T, <_T)$ be a partially ordered belief base where $K_T = \{\neg\alpha_i \vee \beta_i : \alpha_i \rightarrow \beta_i \in T\} \cup \{\neg\alpha_i \vee \neg\beta_i : \alpha_i \rightarrow \beta_i \in T\}$ and $<_T$ is defined by $\{\neg\alpha \vee \beta <_T \neg\alpha \vee \beta : \alpha_i \rightarrow \beta_i \in T\}$. Then, a possibility distribution is π-compatible with $(K_T, <_T)$ iff $<_\pi \in \prod(T)$ where $\prod(T)$ is defined in Section 3.*

4.3 From Partially Ordered Bases to Conditional Bases

Let us consider a partially ordered belief base $K = \{\varphi_i \prec \psi_i\}$ such that $\varphi_i \prec \psi_i$ is interpreted by $\varphi_i <_N \psi_i$.

Note that $\varphi_i <_N \psi_i$ if and only if $\neg\psi_i <_\Pi \neg\varphi_i$. Then, $\neg\psi_i <_\Pi \neg\varphi_i$ is equivalent to say that $\neg\psi_i \wedge \phi_i >_\Pi \neg\varphi_i$ which encodes in a possibilistic way the conditional rule $\neg\psi_i \vee \neg\varphi_i \rightarrow \varphi_i$.

In fact, one can easily see that :

- $(\neg\psi_i \vee \neg\varphi_i) \wedge \varphi_i \equiv \varphi_i \wedge \neg\psi_i$,
- $(\neg\psi_i \vee \neg\varphi_i) \wedge \neg\varphi_i \equiv \neg\varphi_i$.

Consequently, a partially ordered belief base $(K, \prec)$ can be represented via a conditional base $T(K, \prec)$.

4.4 Compatible Bases vs. Compatible Distributions

In Section 4.1, we investigated relationships between the set of possibilistic distributions used from POB-compatible bases and the set of π-compatible bases. In Section 4.2 and Section 4.3, we showed the relation between the set of conditional assertions and the set of partially pre-ordered bases.

A first corollary of the above subsections (4.1, 4.2,4.3) is a proposition of a new inference relation which is stronger than System P and weaker than System Z [18].

Definition 6. *Let T be a conditional knowledge base. Let $(K_T, \prec_T)$ be a partially pre-ordered base obtained from T (see Section 4.2). Then, $\alpha \rightarrow \beta$ is a $F\Pi$-consequence of T iff $\forall <_\pi \in POB(K_T, _p rec_T), \alpha \wedge \beta <_\pi \alpha \wedge \neg\beta$.*

Then, we show that :

Proposition 2. *Let T be a conditional knowledge base. Then,*

- *if $\alpha \rightarrow \beta$ is a consequence of System P then $\alpha \rightarrow \beta$ is an POB consequence,*
- *if $\alpha \rightarrow \beta$ is an POB consequence then $\alpha \rightarrow \beta$ is a consequence of System Z.*

Hence, the relation between conditional knowledge bases and partially ordered bases allows to provide a new consequence relation between P and Z.

5 Complexity of Compatible-Based Inclusion Inference

In [5], complexity results regarding different inference relations from partially pre-ordered bases have been proposed. It is in particular shown that the decision problem associated with lexicographic inference jumps from Δ_2^P-complete when dealing with totally pre-ordered bases to Π_2^P-complete when dealing with partially pre-ordered bases. Regarding inclusion-based inference, only a weaker result has been established, namely the fact that the associated decision problem lies between Π_2^P and Π_3^P.

We show in this section that the decision problem associated with the compatible-based inclusion inference, denoted in the following by CmpIncl, is in fact Π_2^P-complete. Hence, no extra computational complexity is generated when we deal with partially pre-ordered information since it is well-known that inclusion-based inference from totally pre-ordered bases is Π_2^P-complete [16].

The key idea is to show that this problem is reductible to the problem CmpLex which is the decision problem corresponding to the compatible-based lexicographic inference.

Let us first give the following lemma which describes the passage from partially pre-ordered bases to partially ordered bases. Note that in the following, given a partially pre-ordered belief base $(\Sigma, \preceq)$, then $CmpIncl(\Sigma, \preceq)$ (resp. $CmpLex(\Sigma, \preceq)$) denotes the set of all consistent preferred subbases with respect to the inclusion-based (resp. lexicographic) preference over all the totally pre-ordered POB-compatible bases.

Lemma 2. *Let $(\Sigma, \preceq)$ be a partially pre-ordered belief base and let $(\Sigma, \prec_I)$ be the partially strictly ordered belief base obtained from $(\Sigma, \preceq)$ by replacing all equalities by incomparabilities: $\forall \varphi, \psi \in \Sigma$, if $\phi = \psi$ then $\phi \sim_I \psi$. Then, we have:*

$$CmpIncl(\Sigma, \preceq) = CmpIncl(\Sigma, \prec_I).$$

Besides, we show that the compatible-based inclusion inference and the compatible-based lexicographic inference are equivalent with respect to a partially strictly ordered belief base. More formally :

Lemma 3. *Let $(\Sigma, \prec)$ be a partially strictly ordered belief base. Then, we have:*

$$CmpLex(\Sigma, \prec) = CmpIncl(\Sigma, \prec).$$

Now, we are in position to give the following proposition.

Proposition 3. CmpIncl *is Π_2^P-complete.*

Let us sketch the corresponding proof.

- **Membership to Π_2^P :**
  Given a partially pre-ordered belief base $(\Sigma, \preceq)$, then according to lemma 2, we have $CmpIncl(\Sigma, \preceq) = CmpIncl(\Sigma, \prec_I)$ where $(\Sigma, \prec_I)$ is be the partially strictly ordered belief base obtained from $(\Sigma, \preceq)$ by replacing all equalities by incomparabilities.
  Furthermore, we know that according to lemma 3 $CmpIncl(\Sigma, \prec_I) = CmpLex(\Sigma, \prec_I)$ which implies that $CmpIncl(\Sigma, \preceq) = CmpLex(\Sigma, \prec_I)$.
  Subsequently, the problem CmpIncl is reducible to CmpLex, and since this latter is Π_2^P-complete and the class Π_2^P is closed by polynomial reduction, we deduce that CmpIncl belongs to Π_2^P.

- **Completeness for Π_2^P :**
  First of all, the problem CmpIncl generalizes the problem Incl which means that Incl is reducible to CmplNcl. In addition, the problem Incl is Π_2^P-complete. Therefore, CmpIncl is Π_2^P-complete.

6 Conclusion

This paper first pointed out relations between reasoning from conditional knowledge bases and reasoning from partially pre-ordered belief bases. Then, we established relationships between the concept of compatible bases and the concept of compatible distributions. We show that reasoning from compatible distributions is more cautious than reasoning from compatible bases (corollary of Figure 2). The second important contribution concerns the computational complexity of an important inconsistency-tolerant inference, which is inclusion-based is Π_2^P-complete. Hence, in the worst case, with inclusion based inference, reasoning from partially pre-ordered (inconsistent) belief bases has the same complexity as reasoning from totally pre-ordered (inconsistent) belief bases. Then, there is no extra cost when information are partially pre-ordered.

References

1. Benferhat, S., Dubois, D., Cayrol, C., Lang, J., Prade, H.: Inconsistency management and prioritized syntaxbased entailment. In: 13th International Joint Conference on Artificial Intelligence (IJCAI 1993), pp. 640–645 (1993)
2. Benferhat, S., Dubois, D., Prade, H.: Practical handling of exception-tainted rules and independence information in possibilistic logic. Journal of Applied Intelligence 9(2), 101–127 (1998)
3. Benferhat, S., Lagrue, S., Papini, O.: Reasoning with partially ordered information in a possibilistic framework. Fuzzy Sets and Systems 144, 25–41 (2004)
4. Benferhat, S., Dubois, D., Kaci, S., Prade, H.: Bridging logical, comparative and graphical possibilistic representation frameworks. In: Benferhat, S., Besnard, P. (eds.) ECSQARU 2001. LNCS (LNAI), vol. 2143, pp. 422–431. Springer, Heidelberg (2001)
5. Benferhat, S., Yahi, S.: Complexity and cautiousness results for reasoning from partially preordered belief bases. In: Sossai, C., Chemello, G. (eds.) ECSQARU 2009. LNCS, vol. 5590, pp. 817–828. Springer, Heidelberg (2009)
6. da Costa, N.C.A.: Theory of inconsistent formal systems. Notre Dame Journal of Formal Logic 15, 497–510 (1974)
7. Dubois, D., Lang, J., Prade, H.: Possibilistic logic. In: Handbook of Logic in Articial Intelligence and Logic Programming, vol. 3, pp. 439–513 (1994)
8. Dubois, D., Prade, H.: Possibility Theory: An Approach to Computerized Processing of Uncertainty. Plenum Press, New York (1988)
9. Junker, U., Brewka, G.: Handling partially ordered defaults in TMS. In: IJCAI 1989, pp. 1043–1048 (1989)
10. Kraus, S., Lehmann, D.J., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligience 44(1-2), 167–207 (1990)
11. Lehmann, D.J., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)
12. Linke, T., Schaub, T.: An approach to query-answering in reiter's default logic and the underlying existence of extensions problem. In: Dix, J., Fariñas del Cerro, L., Furbach, U. (eds.) JELIA 1998. LNCS (LNAI), vol. 1489, pp. 233–247. Springer, Heidelberg (1998)

13. Linke, T., Schaub, T.: Alternative foundations for reiter's default logic. Artificial Intelligence 124(1), 31–86 (2000)
14. Mengin, J.: A theorem prover for default logic based on prioritized conflict resolution and an extended resolution principle. In: Froidevaux, C., Kohlas, J. (eds.) ECSQARU 1995. LNCS, vol. 946, pp. 301–310. Springer, Heidelberg (1995)
15. Morgenstern, L.: Inheritance comes of age: applying nonmonotonic techniques to problems in industry. In: 15th International Joint Conference on Artificial Intelligence (IJCAI 1997), pp. 1613–1613 (1997)
16. Nebel, B.: Belief Revision and Default Reasoning: Syntax-based Appraoch. In: KR 1991, pp. 417–427 (1991)
17. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Publishing Company, Reading (1994)
18. Pearl, J.: System z: A natural ordering of defaults with tractable applications to default reasoning. In: 6th Theoretical Aspects of Rationality and Knowledge (TARK 1990), pp. 121–135 (1990)
19. Reiterp, R.: A logic for default reasoning. Artificial Intelligence 13(1-2), 81–132 (1980)
20. Rescher, N., Manor, R.: On inference from inconsistent premises. Theory and Decision 1, 179–219 (1970)
21. Yahi, S., Benferhat, S., Lagrue, S., Sérayet, M., Papini, O.: A lexicographic inference for partially preordered belief bases
22. Zadeh, L.A.: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems 1, 3–28 (1978)

A Decidable Constructive Description Logic

Loris Bozzato[1], Mauro Ferrari[1], Camillo Fiorentini[2], and Guido Fiorino[3]

[1] DICOM, Univ. degli Studi dell'Insubria, Via Mazzini 5, 21100, Varese, Italy
[2] DSI, Univ. degli Studi di Milano, Via Comelico, 39, 20135 Milano, Italy
[3] DIMEQUANT, Univ. degli Studi di Milano-Bicocca
P.zza dell'Ateneo Nuovo 1, 20126 Milano, Italy

Abstract. Recently, there has been a growing interest in constructive reinterpretations of description logics. This has been motivated by the need to model in the DLs setting problems that have a consolidate tradition in constructive logics. In this paper we introduce a constructive description logic for the language of $\mathcal{ALC}$ based on the Kripke semantics for Intuitionistic Logic. Moreover we give a tableau calculus and we show that it is sound, complete and terminating.

1 Introduction

Nowadays Description Logics (DLs) are the most prominent formalism for Knowledge Representation. Their success depends on the one side on their "natural" classical semantics and their expressivity and on the other side on the decidability and efficiency of the reasoning problems. However, in recent years there has been a growing interest in different interpretations of DLs allowing one to model knowledge domains and problems that can hardly be treated in the context of a classical semantics. Among these, we recall some proposals towards a constructive approach to DLs: paraconsistent versions of DLs [14], different interpretations of negation [12], the introduction of semantics supporting a computational reading of proofs [3,4,8] and the characterization of incomplete information and typing systems for data streams [13].

As pointed out in [13], in general a constructive reading of DLs is useful in domains with possibly dynamic and incomplete knowledge. This view is supported by the model theoretical features of constructive semantics which allows the representation of stages of information and truth evidence. Following this line, in this paper we introduce the constructive description logic $\mathcal{KALC}$. This logic is based on the same language of $\mathcal{ALC}$ and relies on a Kripke-style semantics which consists in a reformulation in the DLs setting of the Kripke semantics for first-order Intuitionistic Logic. A Kripke model can be considered as a set of worlds, representing states of knowledge, partially ordered by their information content. This permits to express partial and incomplete states of knowledge which can increase in time in the context of the Open World Assumption. We exemplify this in Section 2 by means of an example inspired by [13]. The main difference between our semantics and the one of [13] is that our refinement relation concerns a whole state of knowledge and not a single individual. This provides a

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 51–63, 2010.
© Springer-Verlag Berlin Heidelberg 2010

semantics which seems a "natural" generalization of the classical semantics for $\mathcal{ALC}$. Reasoning problems for DLs in our setting are formulated as in $\mathcal{ALC}$ but have a constructive meaning. Indeed, as we prove in Section 2, $\mathcal{KALC}$ meets the *disjunction property*: if the assertion $c : C \sqcup D$ (c belongs to concept $C \sqcup D$) holds in $\mathcal{KALC}$, then either $c : C$ or $c : D$ is true in $\mathcal{KALC}$. In particular, the classically valid assertion $c : C \sqcup \neg C$ is not true in $\mathcal{KALC}$. This is considered an essential feature of every constructive system. We show that reasoning problems for $\mathcal{KALC}$ are decidable by introducing a tableau calculus $\mathcal{T}_{\mathcal{K}}$ for $\mathcal{KALC}$. To conclude, we notice that our semantics can be viewed as a refinement of $i\mathcal{ALC}$ [6], which corresponds to a direct translation of semantics of Intuitionistic first order logics in the language of $\mathcal{ALC}$.

In the following section we first introduce the syntax and semantics of $\mathcal{KALC}$, discussing its relations with classical semantics and the disjunction property. In Section 3 we introduce the calculus $\mathcal{T}_{\mathcal{K}}$ and we prove its soundness with respect to $\mathcal{KALC}$ semantics: completeness and termination are presented in Section 4. We conclude in Section 5 by considering possible directions for future work.

2 Syntax and Semantics

We begin by introducing the language $\mathcal{L}$ of $\mathcal{KALC}$ which essentially coincides with the one of $\mathcal{ALC}$ [7]. It is based on the following denumerable sets: the set $\mathtt{NI}$ of *individual names*, the set $\mathtt{NC}$ of *atomic concept names*, the set $\mathtt{NR}$ of *role names*. *Concepts* C,D and *formulas* H are defined as follows:

$$C, D ::= \bot \mid A \mid C \sqcap D \mid C \sqcup D \mid C \to D \mid \exists R.C \mid \forall R.C$$
$$H ::= (c, d) : R \mid c : C \mid C \sqsubseteq D$$

where $c, d \in \mathtt{NI}$, $A \in \mathtt{NC}$ and $R \in \mathtt{NR}$. As usual in constructive logics, we write $\neg C$ as an abbreviation for $C \to \bot$. Given $\mathcal{N} \subseteq \mathtt{NI}$, we denote with $\mathcal{L}_{\mathcal{N}}$ the language only containing individual names from $\mathcal{N}$. A *(classical) model* $\mathcal{M}$ for $\mathcal{L}_{\mathcal{N}}$ is a pair $(\mathcal{D}^{\mathcal{M}}, \cdot^{\mathcal{M}})$, where $\mathcal{D}^{\mathcal{M}}$ is a finite non-empty set (the *domain* of $\mathcal{M}$) and $\cdot^{\mathcal{M}}$ is a *valuation* map such that: for every $c \in \mathcal{N}$, $c^{\mathcal{M}} \in \mathcal{D}^{\mathcal{M}}$; for every $A \in \mathtt{NC}$, $A^{\mathcal{M}} \subseteq \mathcal{D}^{\mathcal{M}}$; $\bot^{\mathcal{M}}$ is the empty set; for every $R \in \mathtt{NR}$, $R^{\mathcal{M}} \subseteq \mathcal{D}^{\mathcal{M}} \times \mathcal{D}^{\mathcal{M}}$. A *Kripke model* for $\mathcal{L}_{\mathcal{N}}$ is a quadruple $\underline{K} = \langle P, \leq, \rho, \iota \rangle$, where:

- $(P, \leq)$ is a finite poset with minimum element ρ; P is the set of *worlds* of $\underline{K}$, ρ the *root* of $\underline{K}$.
- ι is a function associating with every world α a model $(\mathcal{D}^{\alpha}, \cdot^{\alpha})$ for $\mathcal{L}_{\mathcal{N}}$ such that, for every $\alpha \leq \beta$, the following holds:

  (K1) $\mathcal{D}^{\alpha} \subseteq \mathcal{D}^{\beta}$;
  (K2) for every $c \in \mathcal{N}$, $c^{\beta} = c^{\alpha}$;
  (K3) for every $A \in \mathtt{NC}$, $A^{\alpha} \subseteq A^{\beta}$;
  (K4) for every $R \in \mathtt{NR}$, $R^{\alpha} \subseteq R^{\beta}$.

Worlds of $\underline{K}$ represent states of knowledge that can be updated or refined by the relation $\leq$; conditions (K1)–(K4) settle that the knowledge is monotonic. Given

$\underline{K} = \langle P, \leq, \rho, \iota \rangle$ and $\alpha \in P$, we denote with $\mathcal{L}_\alpha$ the language obtained by adding to the individual names of $\mathcal{L}_\mathcal{N}$ every element $d \in \mathcal{D}^\alpha$ and setting $d^\alpha = d$. Note that, by Condition (K1), $\alpha \leq \beta$ implies $\mathcal{L}_\alpha \subseteq \mathcal{L}_\beta$. Let α be a world of $\underline{K}$ and H a formula of $\mathcal{L}_\alpha$; we inductively define the *forcing relation* $\alpha \Vdash H$ as follows:

- $\alpha \Vdash c : A$, where $A \in \mathtt{NC}$ or $A = \bot$, iff $c^\alpha \in A^\alpha$;
- $\alpha \Vdash (c, d) : R$ where $R \in \mathtt{NR}$, iff $(c^\alpha, d^\alpha) \in R^\alpha$;
- $\alpha \Vdash c : C \sqcap D$ iff $\alpha \Vdash c : C$ and $\alpha \Vdash c : D$;
- $\alpha \Vdash c : C \sqcup D$ iff $\alpha \Vdash c : C$ or $\alpha \Vdash c : D$;
- $\alpha \Vdash c : C \to D$ iff, for every $\beta \geq \alpha$, $\beta \Vdash c : C$ implies $\beta \Vdash c : D$;
- $\alpha \Vdash c : \exists R.C$ iff there is $d \in \mathcal{D}^\alpha$ such that $\alpha \Vdash (c, d) : R$ and $\alpha \Vdash d : C$;
- $\alpha \Vdash c : \forall R.C$ iff, for every $\beta \geq \alpha$ and $d \in \mathcal{D}^\beta$, $\beta \Vdash (c, d) : R$ implies $\beta \Vdash d : C$;
- $\alpha \Vdash C \sqsubseteq D$ iff, for every $\beta \geq \alpha$ and $c \in \mathcal{D}^\beta$, $\beta \Vdash c : C$ implies $\beta \Vdash c : D$.

We remark that, differently from $\mathcal{ALC}$, the logical connectives are not interdefinable; e.g., $C \to D$ is not equivalent to $\neg C \sqcup D$. As for negation, being $\neg C$ an abbreviation for $C \to \bot$, we get $\alpha \Vdash c : \neg C$ iff for every $\beta \geq \alpha$, $\beta \nVdash c : C$.

By conditions (K1)–(K4) the forcing relation satisfies the *monotonicity property*: if $\alpha \Vdash H$ then $\beta \Vdash H$ for every $\beta \geq \alpha$. A *final* world ϕ of $\underline{K}$ is a maximal element of $(P, \leq)$. Note that $\phi \nVdash H$ implies $\phi \Vdash \neg H$. As a consequence, in ϕ any formula $c : C \sqcup \neg C$ is valid, as in classical models for $\mathcal{ALC}$, hence a final world represents a state of complete knowledge.

We now introduce the notion of $\mathcal{KALC}$*-logical consequence*, denoted by $\models$, which allows us to represent in $\mathcal{KALC}$ the usual inference problems for DLs. Let $\mathcal{F}$ be a set of formulas; by $\alpha \Vdash \mathcal{F}$ we mean that $\alpha \Vdash H$ for every $H \in \mathcal{F}$. Given a formula H, the relation $\mathcal{F} \models H$ holds iff:

- for every $\underline{K} = \langle P, \leq, \rho, \iota \rangle$ and $\alpha \in P$, if $\alpha \Vdash \mathcal{F}$ then $\alpha \Vdash H$.

By the above discussion, it follows that if $\mathcal{F} \models H$ then H is a logical consequence of $\mathcal{F}$ as understood in $\mathcal{ALC}$. Thus, $\mathcal{KALC}$-logical consequence refines the corresponding notion for $\mathcal{ALC}$.

In our setting, an ABox $\mathcal{A}$ is a set of assertions of the kind $(c, d) : R$ and $c : C$, a TBox $\mathcal{T}$ is a set of inclusions of the form $C \sqsubseteq D$. The inference problems for $\mathcal{KALC}$ are formulated as in $\mathcal{ALC}$, using the $\models$ relation. To exemplify:

- *Concept satisfiability.* C is satisfiable w.r.t. $(\mathcal{A}, \mathcal{T})$ iff $\mathcal{A} \cup \mathcal{T} \cup \{q : C\} \not\models c : \bot$, with q not occurring in $\mathcal{A}$.
- *Instance checking.* $c : C$ is entailed by $(\mathcal{A}, \mathcal{T})$ iff $\mathcal{A} \cup \mathcal{T} \models c : C$.
- *Subsumption.* D subsumes C w.r.t. $(\mathcal{A}, \mathcal{T})$ iff $\mathcal{A} \cup \mathcal{T} \models C \sqsubseteq D$.

In the next example we show how our semantics allows us to represent partial and incomplete information and supports constructive reasoning.

Example 1 (Auditing). We reconsider the auditing example of [13]. Let us suppose to have a knowledge base defined by the following ABox $\mathcal{A}$ (CW stands for "Credit Worthy"):

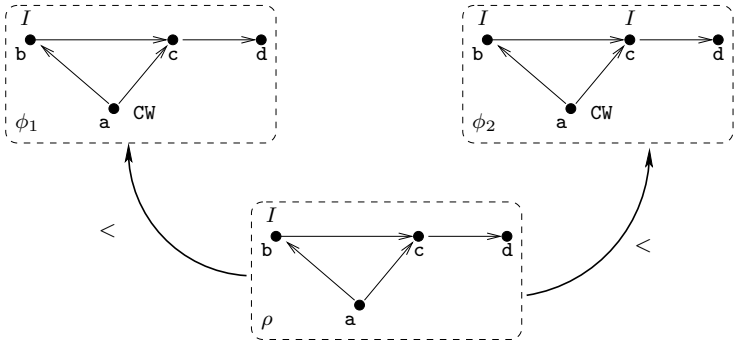


Fig. 1. The model $\underline{K}$

a:Company	d:Company	(a,b):hasCustomer	(c,d):hasCustomer
b:Company	b:Insolvent	(a,c):hasCustomer	a:$D \rightarrow$ CW
c:Company	d:$\neg$Insolvent	(b,c):hasCustomer	

where D is the concept $\exists$hasCustomer.(Insolvent $\sqcap \exists$hasCustomer.$\neg$Insolvent)

The formula a : $D \rightarrow$ CW states that if the company a has an insolvent customer *solv* which in turn can rely on at least one non-insolvent customer *unsolv*, then a can be trusted as CW (here and in Fig. 1 I abbreviates Insolvent). In $\mathcal{ALC}$ the assertion a:CW is entailed by $\mathcal{A}$. To prove this, let $\mathcal{M}$ be any model of $\mathcal{A}$. We first show that a : D holds in $\mathcal{M}$. Since *tertium non datur* classically holds, in $\mathcal{M}$ the customer c is either insolvent or not insolvent. Let us consider the worlds ϕ_2 and ϕ_1 in Fig. 1 representing the two possibilities; in both cases, we can find out the clients *solv* and *unsolv* required by D: $solv =$ c and $unsolv =$ d in the former case, $solv =$ b and $unsolv =$ c in the latter. Since a : $D \rightarrow$ CW holds in $\mathcal{M}$, it follows that a : CW holds in $\mathcal{M}$. Thus, in $\mathcal{ALC}$ the company a is trusted as CW though we have not any knowledge about the identity of the customers *solv* and *unsolv*.

On the contrary, in $\mathcal{KALC}$ the information in $\mathcal{A}$ does not enable to assert that a is CW. The point is that we have not enough knowledge on c, thus neither "c is insolvent" nor "c is not insolvent" can be asserted (indeed, in a future world c might become insolvent). This can be formalized in $\mathcal{KALC}$ semantics as follows. Let $\mathcal{N} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and let us consider the Kripke model $\underline{K} = \langle P, \leq, \rho, \iota \rangle$ for $\mathcal{L}_\mathcal{N}$ in Fig. 1, consisting of the root ρ, two final worlds ϕ_1 and ϕ_2 such that:

- for every $\alpha \in P$, $\mathcal{D}^\alpha = \mathcal{N}$ and, for every $z \in \mathcal{N}$, $z^\alpha = z$;
- atomic concepts and roles are interpreted as follows:

World	Company	Insolvent	CW	hasCustomer
ρ	$\mathcal{N}$	{b}	$\emptyset$	{(a,b), (a,c), (b,c), (c,d)}
ϕ_1	$\mathcal{N}$	{b}	{a}	{(a,b), (a,c), (b,c), (c,d)}
ϕ_2	$\mathcal{N}$	{b, c}	{a}	{(a,b), (a,c), (b,c), (c,d)}

Since $\phi_1 \not\Vdash$ c : Insolvent (actually, $\phi_1 \Vdash$ c : $\neg$Insolvent) and $\phi_2 \Vdash$ c : Insolvent, we have that $\rho \not\Vdash$ c : Insolvent and $\rho \not\Vdash$ c : $\neg$Insolvent, hence

$\rho \not\Vdash \mathtt{c} : \mathtt{Insolvent} \sqcup \neg\mathtt{Insolvent}$. Note that, $\mathtt{b}$ and $\mathtt{c}$ are the only individuals such that $(\mathtt{a},\mathtt{b}) \in \mathtt{hasCustomer}^{\rho}$, $\mathtt{b} \in \mathtt{Insolvent}^{\rho}$ and $(\mathtt{b},\mathtt{c}) \in \mathtt{hasCustomer}^{\rho}$, but $\rho \not\Vdash \mathtt{c} : \neg\mathtt{Insolvent}$. Thus $\rho \not\Vdash \mathtt{a} : D$. Since $\phi_1 \Vdash \mathtt{a} : \mathtt{CW}$ and $\phi_2 \Vdash \mathtt{a} : \mathtt{CW}$, we have $\rho \Vdash \mathtt{a} : D \rightarrow \mathtt{CW}$. To sum up, $\rho \Vdash \mathcal{A}$ and $\rho \not\Vdash \mathtt{a} : \mathtt{CW}$; we conclude that $\mathtt{a} : \mathtt{CW}$ is not a $\mathcal{KALC}$-logical consequence of $\mathcal{A}$. Observe that the final worlds correspond to the two possible ways of acquiring a complete knowledge about the insolvency of $\mathtt{c}$: clearly, in the final worlds $\mathtt{a}$ must be $\mathtt{CW}$. $\diamond$

We conclude the discussion on Kripke semantics by remarking that $\mathcal{KALC}$ satisfies the *Disjunction Property (DP)*:

– $\models c : C_1 \sqcup C_2$ implies $\models c : C_1$ or $\models c : C_2$.

As an immediate consequence, the classically valid assertion $c : C \sqcup \neg C$ is not valid in $\mathcal{KALC}$. The proof of (DP) exploits the standard technique of gluing models: given two Kripke models $\langle P_j, \leq_j, \rho_j, \iota_j\rangle$ $(j = 1, 2)$ such that $\rho_j \not\Vdash_j c : C_j$, one can build a model $\langle P, \leq, \rho, \iota\rangle$, with $\rho \notin P_1 \cup P_2$, such that the immediate successors of ρ are ρ_1 and ρ_2. It follows that $\rho \not\Vdash c : C_1 \sqcup C_2$. Handling with care the same technique, we can prove (DP) in a more general form:

– $\mathcal{F}_H \models c : C_1 \sqcup C_2$ implies $\mathcal{F}_H \models c : C_1$ or $\mathcal{F}_H \models c : C_2$

with $\mathcal{F}_H$ a set of *Harrop Formulas* (occurrences of $\sqcup$ and $\exists$ are only allowed in the left-hand scope of $\rightarrow$ or $\sqsubseteq$).

In this paper we only consider the reasoning problems over *acyclic* TBoxes $\mathcal{T}$ according to the standard definition:

1. $\mathcal{T}$ only contains inclusions $A \sqsubseteq C$, with A an atomic concept.
2. Let us say that the atomic concept A *directly uses* A' in $\mathcal{T}$ iff, for some $A \sqsubseteq C \in \mathcal{T}$, A' is a subformula of C and let *uses* be the transitive closure of the "directly uses" relation. Then, no concept occurring in $\mathcal{T}$ uses itself.

3 The Tableau Calculus $\mathcal{T}_{\mathcal{K}}$

The tableau calculus $\mathcal{T}_{\mathcal{K}}$ works on *signed formulas* $W = \mathcal{S}(H)$, with H a formula and $\mathcal{S}$ a sign in $\{\mathbf{T}, \mathbf{F}, \mathbf{T_s}\}$. Formally:

$$W \quad ::= \quad \mathbf{T}((c,d) : R) \mid \mathbf{F}(c : C) \mid \mathbf{T}(c : C) \mid \mathbf{T_s}(c : C) \mid \mathbf{T}(C \sqsubseteq D)$$

Given a Kripke model $\underline{K} = \langle P, \leq, \rho, \iota\rangle$, a world $\alpha \in P$ and a signed formula W, α *realizes* W *in* $\underline{K}$, and we write $\underline{K}, \alpha \rhd W$, iff:

– $W = \mathbf{T}(H)$ and $\alpha \Vdash H$.
– $W = \mathbf{F}(H)$ and $\alpha \not\Vdash H$.
– $W = \mathbf{T_s}(H)$ and, for every $\beta \in P$ such that $\alpha < \beta$, $\beta \Vdash H$.

The signs $\mathbf{T}$ and $\mathbf{F}$ have the usual meaning [9], whereas $\mathbf{T}_s$ refers to the successors of a world. Let Δ be a set of signed formulas and let

$$\Delta_s \;=\; \{\, \mathbf{T}(H) \mid \mathbf{T}(H) \in \Delta \,\} \;\cup\; \{\, \mathbf{T}(H) \mid \mathbf{T_s}(H) \in \Delta \,\}$$

$$\frac{\Delta, \mathbf{T}(c : C \sqcap D)}{\Delta, \mathbf{T}(c : C), \mathbf{T}(c : D)}\,\mathbf{T}\sqcap \qquad \frac{\Delta, \mathbf{F}(c : C \sqcap D)}{\Delta, \mathbf{F}(c : C) \mid \Delta, \mathbf{F}(c : D)}\,\mathbf{F}\sqcap$$

$$\frac{\Delta, \mathbf{T}(c : C \sqcup D)}{\Delta, \mathbf{T}(c : C) \mid \Delta, \mathbf{T}(c : D)}\,\mathbf{T}\sqcup \qquad \frac{\Delta, \mathbf{F}(c : C \sqcup D)}{\Delta, \mathbf{F}(c : C), \mathbf{F}(c : D)}\,\mathbf{F}\sqcup$$

$$\frac{\Delta, \mathbf{F}(c : C \to D)}{\Delta, \mathbf{T}(c : C), \mathbf{F}(c : D) \mid \Delta_s, \mathbf{T}(c : C), \mathbf{F}(c : D)}\,\mathbf{F}\to$$

$$\frac{\Delta, \mathbf{T}(c : C \to D)}{\Delta, \mathbf{T}(c : D) \mid \Delta, \mathbf{F}(c : C), \mathbf{T_s}(c : D) \mid \Delta_s, \mathbf{F}(c : C), \mathbf{T_s}(c : D)}\,\mathbf{T}\to$$

$$\frac{\Delta, \mathbf{T}(c : A), \mathbf{T}(A \sqsubseteq C)}{\Delta, \mathbf{T}(c : A), \mathbf{T}(A \sqsubseteq C), \mathbf{T}(c : C)}\,\mathbf{T}\sqsubseteq$$

$$\frac{\Delta, \mathbf{T}(c : \exists R.C)}{\Delta, \mathbf{T}((c, q) : R), \mathbf{T}(q : C)}\,\mathbf{T}\exists^* \qquad \frac{\Delta, \mathbf{T}((c, d) : R), \mathbf{F}(c : \exists R.C)}{\Delta, \mathbf{T}((c, d) : R), \mathbf{F}(c : \exists R.C), \mathbf{F}(d : C)}\,\mathbf{F}\exists$$

$$\frac{\Delta, \mathbf{T}((c, d) : R), \mathbf{T}(c : \forall R.C)}{\Delta, \mathbf{T}((c, d) : R), \mathbf{T}(c : \forall R.C), \mathbf{T}(d : C)}\,\mathbf{T}\forall$$

$$\frac{\Delta, \mathbf{F}(c : \forall R.C)}{\Delta, \mathbf{T}((c, q) : R), \mathbf{F}(q : C) \mid \Delta_s, \mathbf{T}((c, q) : R), \mathbf{F}(q : C)}\,\mathbf{F}\forall^*$$

$^*q$ does not occur in the premise

Fig. 2. Rules of $\mathcal{T}_\mathcal{K}$

Then, $\underline{K}, \alpha \rhd \Delta$ implies $\underline{K}, \beta \rhd \Delta_s$ for every $\beta > \alpha$ ($\underline{K}, \alpha \rhd \Delta$ means $\underline{K}, \alpha \rhd W$ for every $W \in \Delta$). We also note that $\underline{K}, \alpha \rhd \mathbf{T_s}(c : \bot)$ iff α is final. We say that Δ *is realizable* if $\underline{K}, \alpha \rhd \Delta$ for some $\underline{K}$ and α. Given a set of formulas $\mathcal{F}$ and a sign $\mathcal{S}$, $\mathcal{S}(\mathcal{F})$ denotes the set of signed formulas $\mathcal{S}(H)$ such that $H \in \mathcal{F}$. The relations among realizability, $\mathcal{KALC}$-logical consequence and $\mathcal{ALC}$-logical consequence are stated by the following theorem, which can be easily proved:

Theorem 1. *Let $\mathcal{F}$ be a set of formulas and q an individual name not in $\mathcal{F}$.*

(i) $\mathcal{F} \models c : C$ iff the set $\mathbf{T}(\mathcal{F}) \cup \{\mathbf{F}(c : C)\}$ is not realizable.
(ii) $\mathcal{F} \models C \sqsubseteq D$ iff the set $\mathbf{T}(\mathcal{F}) \cup \{\mathbf{F}(q : C \to D)\}$ is not realizable.
(iii) H is an $\mathcal{ALC}$-logical consequence of $\mathcal{F}$ iff $\mathbf{T}(\mathcal{F}) \cup \{\mathbf{T_s}(c : \bot), \mathbf{F}(H)\}$ is not realizable.
(iv) $c : C$ is an $\mathcal{ALC}$-logical consequence of $\mathcal{F}$ iff $\mathcal{F} \models c : \neg\neg C$. □

The rules of the tableau calculus $\mathcal{T}_\mathcal{K}$ are shown in Fig. 2. In the rules we write Δ, W as a shorthand for $\Delta \cup \{W\}$; moreover, if Δ, W is the premise of a rule, we assume $W \notin \Delta$. Every rule applies to a set of signed formulas, but only acts on the signed formula W explicitly indicated in the premise. The consequence of a rule consists of one or more sets of signed formulas separated by the symbol '|'.

In the rules $\mathbf{T}\exists$ and $\mathbf{F}\forall$, q is a fresh individual name. Formulas of the kind $\mathbf{F}(c : \exists R.C)$, $\mathbf{T}(c : \forall R.C)$ and $\mathbf{T}(A \sqsubseteq C)$ must be duplicated in rule application to guarantee the completeness; we call them *dup-formulas*. Note that in the

intuitionistic case the treatment of $\mathbf{T}\rightarrow$-rule is problematic and requires duplications [1]; in $\mathcal{T}_{\mathcal{K}}$ duplications are avoided by the introduction of the sign $\mathbf{T_s}$. A set Δ *clashes* iff $\{\mathbf{F}(c:C),\mathbf{T}(c:C)\}\subseteq\Delta$ or $\mathbf{T}(c:\bot)\in\Delta$. Clearly, a clashing set is not realizable. A *proof table* for Δ is a finite tree τ with Δ as root and such that all the children of a node Δ' of τ are the sets in the consequence of a rule applied to Δ'. If all the leaves of τ clash, τ is a *closed proof table* for Δ and we say that Δ is *provable (in* $\mathcal{T}_{\mathcal{K}}$*)*; Δ is *consistent* iff Δ is not provable.

Before proving soundness and completeness we give an example of a proof.

Example 2. Let $H = C\sqcup\neg C$. Since $c:H$ is valid in $\mathcal{ALC}$, by Theorem 1 $c:\neg\neg H$ is valid in $\mathcal{KALC}$. We show a proof of $c:\neg\neg H$ (recall that $\neg D = D\rightarrow\bot$). The proof is displayed according to the standard notation [9]. In the proof we underline the clashing formulas, we denote with X a clashing set and we label with an integer the formulas treated by the rules when needed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{F}(c:\neg\neg H)}{\mathbf{T}(c:\neg H),\mathbf{F}(c:\bot)}\,\mathbf{F}\rightarrow}{\underline{\mathbf{T}(c:\bot)},\mathbf{F}(c:\bot)\mid\mathbf{F}(c:H)^1,\mathbf{T_s}(c:\bot),\mathbf{F}(c:\bot)\mid\mathbf{F}(c:H)^2,\mathbf{T_s}(c:\bot)}\,\mathbf{T}\rightarrow}{\mathrm{X}\mid\mathbf{F}(c:C),\mathbf{F}(c:\neg C)^3,\mathbf{T_s}(c:\bot),\mathbf{F}(c:\bot)\mid\Delta=\mathbf{F}(c:C),\mathbf{F}(c:\neg C)^4,\mathbf{T_s}(c:\bot)}\,\mathbf{F}\sqcup^1,\mathbf{F}\sqcup^2}{\mathrm{X}\mid\underline{\mathbf{F}(c:C)},\underline{\mathbf{T}(c:C)},\mathbf{T_s}(c:\bot),\mathbf{F}(c:\bot)\mid\mathbf{T}(c:C),\mathbf{F}(c:\bot),\underline{\mathbf{T}(c:\bot)}\mid\Delta}\,\mathbf{F}\rightarrow^3}{\mathrm{X}\mid\mathrm{X}\mid\mathrm{X}\mid\underline{\mathbf{F}(c:C)},\underline{\mathbf{T}(c:C)},\mathbf{F}(c:\bot),\mathbf{T_s}(c:\bot)\mid\mathbf{T}(c:C),\mathbf{F}(c:\bot),\underline{\mathbf{T}(c:\bot)}}\,\mathbf{F}\rightarrow^4$$

Note that, if $\mathbf{T_s}(c:\bot)\in\Delta$, then Δ_s clashes. Thus, in applying one of the rules $\mathbf{F}\rightarrow$, $\mathbf{T}\rightarrow$ and $\mathbf{F}\forall$ to Δ, we can drop out the rightmost set in the conclusion and a proof table for $\Delta,\mathbf{T_s}(c:\bot)$ resembles an $\mathcal{ALC}$ proof table. ◇

Soundness. The following is the main lemma to prove the soundness of $\mathcal{T}_{\mathcal{K}}$.

Lemma 1. *Let Δ be a set of signed formulas, $\underline{K}=\langle P,\leq,\rho,\iota\rangle$ a Kripke model such that $\underline{K},\alpha\rhd\Delta$, with $\alpha\in P$, and r a rule of $\mathcal{T}_{\mathcal{K}}$ applicable to Δ. Then, there is a set Δ' in the consequence of r and $\beta\in P$ such that $\underline{K},\beta\rhd\Delta'$.*

Proof. We only discuss the case of rule $\mathbf{T}\rightarrow$. Let $W=\mathbf{T}(c:C\rightarrow D)$ and let us assume $\underline{K},\alpha\rhd\Delta,W$. If $\underline{K},\alpha\rhd\mathbf{T}(c:D)$, the assertion holds. Otherwise, $\underline{K},\alpha\rhd\mathbf{F}(c:C)$; being $\underline{K}$ finite, there exists $\beta\geq\alpha$ such that $\underline{K},\beta\rhd\mathbf{F}(c:C)$ and $\underline{K},\beta\rhd\mathbf{T_s}(c:C)$, which implies $\underline{K},\beta\rhd\mathbf{T_s}(c:D)$. If $\beta=\alpha$ then $\underline{K},\beta\rhd\Delta$, otherwise $\underline{K},\beta\rhd\Delta_s$, and the assertion is proved. □

By the previous lemma we get:

Theorem 2 (Soundness). *Let Δ be a set of signed formulas. If Δ is realizable, then Δ is consistent.* □

4 Completeness and Termination

In this section we prove the completeness of $\mathcal{T}_{\mathcal{K}}$ and we provide a decision procedure for $\mathcal{KALC}$ based on $\mathcal{T}_{\mathcal{K}}$. Let Δ be a set of signed formulas; we say that Δ

is *acyclic* iff the set of $A \sqsubseteq C$ such that $\mathbf{T}(A \sqsubseteq C) \in \Delta$ is an acyclic TBox. Note that, according to Theorem 1, to solve the inference problems w.r.t. an acyclic TBox is equivalent to decide the realizability of an acyclic set. We show that, given a finite acyclic consistent set Δ, we can build in finite time a countermodel for Δ, i.e. a Kripke model $\underline{K} = \langle P, \leq, \rho, \iota \rangle$ such that $\underline{K}, \rho \rhd \Delta$. Our construction is inspired to the standard technique used for $\mathcal{ALC}$ [2] based on graph expansion. A labelled graph $\mathcal{G}$ refers to a world α of the countermodel $\underline{K}$ under construction: the nodes of $\mathcal{G}$ form the domain $\mathcal{D}^\alpha$ of α, while the labelled arcs (c, d, R) of $\mathcal{G}$ define the interpretation of R in α. Each node c is associated with a finite set of signed formulas $\mathcal{S}(c : C)$, representing the formulas that must be realized in α. To get this, we repeatedly apply the following transformation rules on $\mathcal{G}$:

1. Firstly, we apply to $\mathcal{G}$ *expansion rules* as in the standard construction of a downward saturated set. We call *expanded graph* the graph $\mathrm{Exp}(\mathcal{G})$ obtained at the end of this step; $\mathrm{Exp}(\mathcal{G})$ completely describes a world α of $\underline{K}$.
2. Let $\mathcal{G}_e$ be an expanded graph describing a world α. We give rules to compute the *successor graphs* $\mathcal{G}'$ of $\mathcal{G}_e$ so that the graphs $\mathrm{Exp}(\mathcal{G}')$ will be all the immediate successors of α in $\underline{K}$.

We need some care to guarantee the termination. We partition the formulas associated with a node in *primary* and *secondary formulas*. Roughly speaking, primary formulas drive the graph construction. At every step a primary formula or a TBox axiom is selected and the graph is expanded according to the chosen formula. The formulas already considered are collected in the set of secondary formulas. Dup-formulas require an ad-hoc treatment to avoid infinite loops: for every dup-formula we store the individual names already considered in the expansion procedure. The TBox formulas can be seen as "global constraints" on $\mathcal{G}$ and are not affected by the transformation rules, thus we take them apart.

Formally, we consider *labelled graphs* $\mathcal{G} = \langle \mathcal{N}, \mathcal{E}, \textsc{Pf}, \textsc{Sf}, \textsc{Tb}, \textsc{Df} \rangle$ where:

- $\mathcal{N}$ is the set of *nodes*, with $\mathcal{N}$ a finite subset of $\mathtt{NI}$.
- $\mathcal{E}$ is the set of *labelled edges* (c, d, R), with $c, d \in \mathcal{N}$ and $R \in \mathtt{NR}$.
- $\textsc{Pf}$ and $\textsc{Sf}$ are functions associating with every node c a finite set of signed formulas $\mathcal{S}(c : C)$, called the *primary* and *secondary* formulas of c respectively.
- $\textsc{Tb}$ has the form $\mathbf{T}(\mathcal{T})$, with $\mathcal{T}$ a finite acyclic TBox.
- $\textsc{Df}$ is a function mapping a dup-formula to a finite set of nodes.

The sets $\textsc{Form}(\mathcal{G})$ and $\textsc{Form}^*(\mathcal{G})$ are defined as:

$$\textsc{Form}(\mathcal{G}) = \bigcup_{c \in \mathcal{N}} \textsc{Pf}(c) \cup \{ \mathbf{T}((c, d) : R) \mid (c, d, R) \in \mathcal{E} \} \cup \textsc{Tb}$$
$$\textsc{Form}^*(\mathcal{G}) = \textsc{Form}(\mathcal{G}) \cup \bigcup_{c \in \mathcal{N}} \textsc{Sf}(c)$$

Assumptions on $\mathcal{G}$. In the following we assume that at any step of the countermodel construction a graph $\mathcal{G}$ satisfies the following properties (G1) and (G2):

(G1) $\textsc{Form}(\mathcal{G})$ is consistent.
(G2) The following closure properties hold:

- If $\mathbf{T}(c : C \sqcap D) \in \textsc{Sf}(c)$, then $\{\mathbf{T}(c : C), \mathbf{T}(c : D)\} \subseteq \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{F}(c : C \sqcap D) \in \textsc{Sf}(c)$, then $\mathbf{F}(c : C) \in \textsc{Form}^*(\mathcal{G})$ or $\mathbf{F}(c : D) \in \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{T}(c : C \sqcup D) \in \textsc{Sf}(c)$, then $\mathbf{T}(c : C) \in \textsc{Form}^*(\mathcal{G})$ or $\mathbf{T}(c : D) \in \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{F}(c : C \sqcup D) \in \textsc{Sf}(c)$, then $\{\mathbf{F}(c : C), \mathbf{F}(c : D)\} \subseteq \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{T}(c : C \to D) \in \textsc{Sf}(c)$, then $\mathbf{T}(c : D) \in \textsc{Form}^*(\mathcal{G})$ or $\{\mathbf{F}(c : C), \mathbf{T_s}(c : D)\} \subseteq \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{F}(c : C \to D) \in \textsc{Sf}(c)$, then $\{\mathbf{T}(c : C), \mathbf{F}(c : D)\} \in \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{T}(c : \exists R.C) \in \textsc{Sf}(c)$, then there is $(c, q, R) \in \mathcal{E}$ s.t. $\mathbf{T}(q : C) \in \textsc{Form}^*(\mathcal{G})$.
- If $W = \mathbf{F}(c : \exists R.C) \in \textsc{Pf}(c)$ and $d \in \textsc{Df}(W)$, then $\mathbf{F}(d : C) \in \textsc{Form}^*(\mathcal{G})$.
- If $W = \mathbf{T}(c : \forall R.C) \in \textsc{Pf}(c)$ and $d \in \textsc{Df}(W)$, then $\mathbf{T}(d : C) \in \textsc{Form}^*(\mathcal{G})$.
- If $\mathbf{F}(c : \forall R.C) \in \textsc{Sf}(c)$, then there is $(c, q, R) \in \mathcal{E}$ s.t. $\mathbf{F}(q : C) \in \textsc{Form}^*(\mathcal{G})$.
- If $W = \mathbf{T}(A \sqsubseteq C) \in \textsc{Tb}$ and $c \in \textsc{Df}(W)$ and $\mathbf{T}(c : A) \in \textsc{Pf}(c)$, then $\mathbf{T}(c : C) \in \textsc{Form}^*(\mathcal{G})$.

The starting graph $\mathcal{G}_\Delta$. The countermodel construction for Δ starts with the graph $\mathcal{G}_\Delta = \langle \mathcal{N}_\Delta, \mathcal{E}_\Delta, \textsc{Pf}_\Delta, \textsc{Sf}_\Delta, \textsc{Tb}, \textsc{Df}_\Delta \rangle$, where $\mathcal{N}_\Delta$ is the set of individual names occurring in Δ, $\mathcal{E}_\Delta$ is the set of (c, d, R) such that $\mathbf{T}((c, d) : R) \in \Delta$, $\textsc{Pf}_\Delta(c)$ is the set of $\mathcal{S}(c : A) \in \Delta$, $\textsc{Sf}_\Delta$ and $\textsc{Df}_\Delta$ maps any element to the empty set, $\textsc{Tb}$ is the set of $\mathbf{T}(A \sqsubseteq C) \in \Delta$. One can easily check that $\mathcal{G}_\Delta$ satisfies (G1) and (G2).

Expansion of a graph $\mathcal{G}$. Let $\mathcal{G} = \langle \mathcal{N}, \mathcal{E}, \textsc{Pf}, \textsc{Sf}, \textsc{Tb}, \textsc{Df} \rangle$ be a finite graph and $W \in \textsc{Form}(\mathcal{G})$. *Expansion rules* are defined in Fig. 3. Given W, the corresponding expansion rule transforms $\mathcal{G}$ in a new graph $\mathcal{G}' = \langle \mathcal{N}', \mathcal{E}', \textsc{Pf}', \textsc{Sf}', \textsc{Tb}, \textsc{Df}' \rangle$. In the rules, $\mathcal{S}_W$ denotes the sign of W. We only indicate the components of the graph that are actually modified; if an element E of $\mathcal{G}$ is not mentioned, it is understood that the corresponding element E' of $\mathcal{G}'$ coincides with E. In some cases rules have no effect (for instance, in the case $W = \mathbf{F}(c : C \to D)$ when the if condition does not hold).

We repeatedly apply expansion rules to $\mathcal{G}$ until no rule is applicable. Let $\text{Exp}(\mathcal{G})$ denote the *expanded graph* $\mathcal{G}_e = \langle \mathcal{N}_e, \mathcal{E}_e, \textsc{Pf}_e, \textsc{Sf}_e, \textsc{Tb}, \textsc{Df}_e \rangle$ obtained at the end of the expansion step; $\text{Mod}(\mathcal{G}_e)$ is the model $(\mathcal{D}^\alpha, \cdot^\alpha)$ for $\mathcal{L}_{\mathcal{N}_e}$ representing the world α such that:

- $\mathcal{D}^\alpha = \mathcal{N}_e$ and, for every $c \in \mathcal{N}_e$, $c^\alpha = c$;
- for every $A \in \texttt{NC}$, A^α is the set of c such that $\mathbf{T}(c : A) \in \textsc{Pf}_e(c)$;
- for every $R \in \texttt{NR}$, R^α is the set of pairs (c, d) such that $(c, d, R) \in \mathcal{E}_e$.

The following properties are crucial to prove the finiteness of $\text{Exp}(\mathcal{G})$.

(P1) For every $c \in \mathcal{N}_e$, the set of R-successors of c in $\mathcal{G}_e$ is finite.
(P2) Let $c_0 \in \mathcal{N}_e \setminus \mathcal{N}$, let $\sigma = c_0, c_1, \ldots$ be an R-chain of nodes of $\mathcal{G}_e$, namely: $(c_k, c_{k+1}, R) \in \mathcal{E}_e$ for every $k \geq 0$. Let A be a concept name and $\mathcal{B}(\sigma, A)$ the set of c in σ such that $\mathbf{T}(c : A) \in \textsc{Pf}_e(c)$. Then, $\mathcal{B}(\sigma, A)$ is finite.

We only give a sketch of the proof. As for (P1), d is an R-successor of c in $\mathcal{G}_e$ iff $(c, d, R) \in \mathcal{E}$ or d has been generated by a formula $W = \mathbf{T}(c : \exists R.C)$ or $W = \mathbf{F}(c : \forall R.C)$. In the former case the assertion follows by finiteness of $\mathcal{G}$. In

Formula	Expansion rule
$W = \mathbf{T}(c : C \sqcap D)$ $W = \mathbf{F}(c : C \sqcup D)$	$\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathcal{S}_W(c : C), \mathcal{S}_W(c : D)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$
$W = \mathbf{F}(c : C \sqcap D)$ $W = \mathbf{T}(c : C \sqcup D)$	If $(\Delta \setminus \{W\}) \cup \{\mathcal{S}_W(c : C)\}$ is consistent, then $\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathcal{S}_W(c : C)\}$ else $\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathcal{S}_W(c : D)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$
$W = \mathbf{F}(c : C \to D)$	If $(\Delta \setminus \{W\}) \cup \{\mathbf{T}(c : C), \mathbf{F}(c : D)\}$ is consistent then $\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathbf{T}(c : C), \mathbf{F}(c : D)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$
$W = \mathbf{T}(c : C \to D)$	If $(\Delta \setminus \{W\}) \cup \{\mathbf{T}(c : D)\}$ is consistent then $\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathbf{T}(c : D)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$ else if $(\Delta \setminus \{W\}) \cup \{\mathbf{F}(c : C), \mathbf{T_s}(c : D)\}$ is consistent then $\mathrm{P_F}'(c) = (\mathrm{P_F}(c) \setminus \{W\}) \cup \{\mathbf{F}(c : C), \mathbf{T_s}(c : D)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$
$W = \mathbf{T}(A \sqsubseteq C)$	Let $c \in \mathcal{N} \setminus \mathrm{D_F}(W)$ If $\mathbf{T}(c : A) \in \mathrm{P_F}(c)$ then $\mathrm{P_F}'(c) = \mathrm{P_F}(c) \cup \{\mathbf{T}(c : C)\}$ $\mathrm{D_F}'(W) = \mathrm{D_F}(W) \cup \{c\}$
$W = \mathbf{T}(c : \exists R.C)$	Let $q \notin \mathcal{N}$. $\mathcal{N}' = \mathcal{N} \cup \{q\}$ $\quad \mathcal{E}' = \mathcal{E} \cup \{(c, q, R)\}$ $\mathrm{P_F}'(c) = \mathrm{P_F}(c) \setminus \{W\}$ $\quad \mathrm{P_F}'(q) = \{\mathbf{T}(q : C)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$ $\quad \mathrm{S_F}'(q) = \emptyset$
$W = \mathbf{F}(c : \exists R.C)$ $W = \mathbf{T}(c : \forall R.C)$	Let $d \in \mathcal{N}$ such that $(c, d, R) \in \mathcal{E}$ and $d \notin \mathrm{D_F}(W)$ $\mathrm{P_F}'(d) = \mathrm{P_F}(d) \cup \{\mathcal{S}_W(d : C)\}$ $\mathrm{D_F}'(W) = \mathrm{D_F}(W) \cup \{d\}$
$W = \mathbf{F}(c : \forall R.C)$	Let $q \notin \mathcal{N}$ If $(\Delta \setminus \{W\}) \cup \{\mathbf{T}((c, q) : R), \mathbf{F}(q : C)\}$ is consistent $\mathcal{N}' = \mathcal{N} \cup \{q\}$ $\quad \mathcal{E}' = \mathcal{E} \cup \{(c, q, R)\}$ $\mathrm{P_F}'(c) = \mathrm{P_F}(c) \setminus \{W\}$ $\quad \mathrm{P_F}'(q) = \{\mathbf{F}(q : C)\}$ $\mathrm{S_F}'(c) = \mathrm{S_F}(c) \cup \{W\}$ $\quad \mathrm{S_F}'(q) = \emptyset$

$\mathcal{S}_W$ denotes the sign of W

Fig. 3. Expansion rules

the latter two cases, W must be a subformula of a formula in $\textsc{Form}^*(\mathcal{G})$, and only finitely many such W exist.

Let $\mathrm{T_B} = \mathbf{T}(\mathcal{T})$ and let $\prec$ be the "uses" relation induced by the TBox $\mathcal{T}$. We prove (P2) by induction on $\prec$ (recall that $\mathcal{T}$ is finite and acyclic, hence $\prec$ is well-founded). If A is minimal w.r.t. $\prec$ then, for every $\mathbf{T}(A' \sqsubseteq C) \in \mathrm{T_B}$, A is not a subformula of C. Thus, $c \in \mathcal{B}(\sigma, A)$ iff $\mathbf{T}(c : A)$ has been generated by some formula in $\mathrm{P_F}_e(c_0)$ or $\mathrm{S_F}_e(c_0)$, and this implies that $\mathcal{B}(\sigma, A)$ is finite. Suppose that A is not minimal. If $\mathcal{B}(\sigma, A)$ is infinite, there must exist a formula $W = \mathbf{T}(A' \sqsubseteq C) \in \mathrm{T_B}$ such that A is a subformula of C and the rule $\mathbf{T}\,\sqsubseteq$ has been applied infinitely many times on W. It follows that $\mathcal{B}(\sigma, A')$ is infinite. Since $A' \prec A$, this contradicts the induction hypothesis.

Formula	Successor graph
$W = \mathbf{F}(c : C \to D)$ $W = \mathbf{T}(c : C \to D)$	$\mathcal{N}' = \mathcal{N} \quad \mathcal{E}' = \mathcal{E} \quad \textsc{Df}' = \textsc{Df}_s$ $\textsc{Pf}'(c) = (\textsc{Pf}(c))_s \cup \mathcal{R}_W \quad \textsc{Pf}'(d) = (\textsc{Pf}(d))_s$ for every $d \neq c$ $\textsc{Sf}'(c) = (\textsc{Sf}(c))_s \cup \{W\} \quad \textsc{Sf}'(d) = (\textsc{Sf}(d))_s$ for every $d \neq c$ where $\mathcal{R}_{\mathbf{F}(c:C\to D)} = \{\mathbf{T}(c : C), \mathbf{F}(c : D)\}$ and $\mathcal{R}_{\mathbf{T}(c:C\to D)} = \{\mathbf{F}(c : C), \mathbf{T_s}(c : D)\}$
$W = \mathbf{F}(c : \forall R.C)$	Let $q \notin \mathcal{N}$. $\mathcal{N}' = \mathcal{N} \cup \{q\} \quad \mathcal{E}' = \mathcal{E} \cup \{(c, q, R)\} \quad \textsc{Df}' = \textsc{Df}_s$ $\textsc{Pf}'(q) = \{\mathbf{F}(q : C)\} \quad \textsc{Pf}'(d) = (\textsc{Pf}(d))_s$ for every $d \in \mathcal{N}$ $\textsc{Sf}'(c) = (\textsc{Sf}(c))_s \cup \{W\} \quad \textsc{Sf}'(q) = \emptyset$ $\textsc{Sf}'(e) = (\textsc{Sf}(e))_s$ for every $e \in \mathcal{N} \setminus \{c\}$

$\textsc{Df}_s(Z) = \textsc{Df}(Z)$ if $Z = \mathbf{T}(H)$, otherwise $\textsc{Df}_s(Z) = \emptyset$

Fig. 4. Successor graphs

By (P2) it follows that $\mathcal{G}_e$ does not contain infinite R-chains starting from a node $c_0 \in \mathcal{N}_e \setminus \mathcal{N}$. We conclude:

Lemma 2. $\textsc{Exp}(\mathcal{G})$ *is finite.* □

Successor of an expanded graph $\mathcal{G}$. Let W be a formula of $\textsc{Form}(\mathcal{G})$. The *successor graph* of $\mathcal{G}$ generated by W is the graph $\mathcal{G}' = \langle \mathcal{N}', \mathcal{E}', \textsc{Pf}', \textsc{Sf}', \textsc{Tb}, \textsc{Df}' \rangle$ defined according to the form of W as specified in Fig. 4.

Countermodel construction. Let Δ be a finite acyclic consistent set of signed formulas. The countermodel $\underline{K}(\Delta) = \langle P, \leq, \rho, \iota \rangle$ for Δ is built as follows.

- The root ρ coincides with $\textsc{Mod}(\textsc{Exp}(\mathcal{G}_\Delta))$, where $\mathcal{G}_\Delta$ is the starting graph.
- Let $\alpha = \textsc{Mod}(\mathcal{G}_\alpha)$ be a world of $\underline{K}(\Delta)$ and let $\mathcal{G}_1, \ldots, \mathcal{G}_m$ be all the successors of $\mathcal{G}_\alpha$. Then, the immediate successors of α in $\underline{K}(\Delta)$ are the models $\textsc{Mod}(\textsc{Exp}(\mathcal{G}_1)), \ldots, \textsc{Mod}(\textsc{Exp}(\mathcal{G}_m))$.
- $\leq$ is the reflexive and transitive closure of the immediate successor relation.

The termination of the countermodel construction procedure is guaranteed by the following property.

(T) Let $\mathcal{G}'$ be obtained by applying to $\mathcal{G}$ one of the rules of Fig. 3 and 4 defined by W. Then, one of the following facts holds ($|W|$ denotes the size of W):
  (1) $\textsc{Form}(\mathcal{G}')$ is obtained by replacing W with one or more formulas W' such that $|W'| < |W|$, possibly substituting $\mathbf{T_s}$ with $\mathbf{T}$ and discharging the $\mathbf{F}$-formulas.
  (2) If W is a dup-formula, $\textsc{Form}(\mathcal{G}') = \textsc{Form}(\mathcal{G}) \cup \{W'\}$, with $|W'| < |W|$, and $\textsc{Df}(W) \subset \textsc{Df}'(W)$.

By Lemma 2 the sets $\textsc{Df}(W)$ can not increase indefinitely, thus we cannot apply the transformation rules infinitely many times.

We now state the main results of this section.

Lemma 3. *Let Δ be a finite acyclic consistent set of signed formulas.*

(i) The model $\underline{K}(\Delta)$ is finite.
(ii) Let $\alpha = \mathrm{Mod}(\mathcal{G}_\alpha)$ be a world of $\underline{K}(\Delta)$. Then, $\underline{K}(\Delta), \alpha \rhd \textsc{Form}^*(\mathcal{G}_\alpha)$.
(iii) $\underline{K}(\Delta), \rho \rhd \Delta$.

Proof. Point (i) follows by Property (T). To prove (ii), one has to show that $W \in \textsc{Form}^*(\mathcal{G}_\alpha)$ implies $\underline{K}(\Delta), \alpha \rhd W$; the proof is by induction on W, using (G1) and (G2). Point (iii) follows by (ii), being $\Delta \subseteq \textsc{Form}(\mathcal{G}_\Delta) \subseteq \textsc{Form}^*(\mathcal{G}_\rho)$. □

By the previous lemma and by the Soundness Theorem we conclude:

Theorem 3 (Completeness). *Let Δ be a finite acyclic set of signed formulas. Then, Δ is realizable iff Δ is consistent.* □

The countermodel construction procedure can be used to decide the realizability of an acyclic Δ. Indeed, one tries to build $\underline{K}(\Delta)$ by applying the transformation rules in all possible ways; by Property (T), the search space is finite. If all the attempts fail, yielding a clashing set $\textsc{Pf}(c)$, Δ is not consistent (Lemma 3), hence it is not realizable (Theorem 3). In this case, the failed branches correspond to the branches of a closed proof table for Δ.

5 Related Works and Conclusions

The logic $\mathcal{KALC}$ we have introduced is strongly connected with the constructive DL presented in [5], let us call $\mathcal{KALC}'$. Indeed, a Kripke model $\underline{K} = \langle P \leq, \rho, \iota \rangle$ for $\mathcal{KALC}'$ is a $\mathcal{KALC}$ model where P can be infinite and, for every $\alpha \in P$, there is a final element $\phi \in P$ such that $\alpha \leq \phi$. The restriction to finite models is crucial to prove the decidability of $\mathcal{KALC}$ (whereas $\mathcal{KALC}'$ is semidecidable). Clearly, $\mathcal{KALC}' \subseteq \mathcal{KALC}$. If, as we conjecture, $\mathcal{KALC} \subseteq \mathcal{KALC}'$, we can conclude that $\mathcal{KALC} = \mathcal{KALC}'$ has the finite model property.

It is well-known that DLs have multi-modal logic counterparts [2]; likewise, intuitionistic DLs are related to intuitionistic multi-modal logics [10,15], via the standard translation between the involved languages. It is easy to prove that the multi-modal version of Fischer-Servi logic **FS** [10] is contained in $\mathcal{KALC}$. On the other hand **FS** $\neq \mathcal{KALC}$, since the formula $H = c : \forall R.\neg\neg A \rightarrow \neg\neg\forall R.A$ belongs to $\mathcal{KALC}$, while the corresponding formula $\Box\neg\neg A \rightarrow \neg\neg\Box A$ does not belong to **FS** (see the countermodel in [15]). We remark that H belongs to $\mathcal{KALC}'$ as well, due to the fact that $\mathcal{KALC}'$ models have final elements.

As for the comparison with other approaches, we notice that our notion of refinement (induced by the partial order relation of Kripke models) concerns the whole state of knowledge. So it is closer to the usual Kripke interpretation than those given in [13,14], which concern single individuals. Note that in [13] the knowledge about roles is not monotonic. We plan to investigate the relation between $\mathcal{KALC}$ and the constructive description logic $\mathcal{BCDL}$ [8], which exploits a different semantics. Finally, we aim to extend the decision procedure to treat general TBoxes and transitive and inverse role relations by introducing loop-checking mechanisms, such as *blocking* and its variants [11].

References

1. Avellone, A., Ferrari, M., Miglioli, P.: Duplication-free tableau calculi and related cut-free sequent calculi for the interpolable propositional intermediate logics. Logic J. of the IGPL 7(4), 447–480 (1999)
2. Baader, F., Nutt, W.: Basic description logics. In: [7], pp. 43–95
3. Bozzato, L., Ferrari, M., Fiorentini, C., Fiorino, G.: A constructive semantics for $\mathcal{ALC}$. In: Calvanese, D., et al. (eds.) 2007 International Workshop on Description Logics. CEUR Proceedings, vol. 250, pp. 219–226 (2007)
4. Bozzato, L., Ferrari, M., Villa, P.: Actions over a constructive semantics for $\mathcal{ALC}$. In: Baader, F., et al. (eds.) 2008 International Workshop on Description Logics. CEUR Proceedings, vol. 353 (2008)
5. Bozzato, L., Ferrari, M., Villa, P.: A note on constructive semantics for description logics. In: 24-esimo Convegno Italiano di Logica Computazionale (2009), http://www.ing.unife.it/eventi/cilc09/accepted.shtml
6. de Paiva, V.: Constructive description logics: what, why and how. Technical report, Xerox Parc (2003)
7. Baader, F., et al. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
8. Ferrari, M., Fiorentini, C., Fiorino, G.: BCDL: Basic Constructive Description Logic. J. of Automated Reasoning 44(4), 371–399 (2010)
9. Fitting, M.C.: Proof Methods for Modal and Intuitionistic Logics. Reidel, Dordrechtz (1983)
10. Gabbay, D.M., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional modal logics: theory and applications. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam (2003)
11. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic J. of the IGPL 8(3), 239–264 (2000)
12. Kaneiwa, K.: Negations in description logic - contraries, contradictories, and subcontraries. In: Proc. of the 13th International Conference on Conceptual Structures (ICCS 2005), pp. 66–79. Kassel University Press (2005)
13. Mendler, M., Scheele, S.: Towards Constructive DL for Abstraction and Refinement. J. of Automated Reasoning 44(3), 207–243 (2010)
14. Odintsov, S.P., Wansing, H.: Inconsistency-tolerant description logic. Part II: A tableau algorithm for $CALC^C$. J. of Applied Logic 6(3), 343–360 (2008)
15. Simpson, A.K.: The Proof Theory and Semantics of Intuitionistic Modal Logic. PhD thesis, University of Edinburgh (1994)

A Normal Form for Linear Temporal Equilibrium Logic*

Pedro Cabalar

Dept. Computación,
University of Corunna, Spain
cabalar@udc.es

Abstract. In previous work, the so-called Temporal Equilibrium Logic (TEL) was introduced. This formalism provides an extension of the Answer Set semantics for logic programs to arbitrary theories in the syntax of Linear Temporal Logic. It has already been shown that, in the non-temporal case, arbitrary propositional theories can always be reduced to logic program rules (with disjunction and negation in the head) independently on the context. That is, logic programs constitute a normal form for the non-temporal case. In this paper we show that TEL can be similarly reduced to a normal form consisting of a set of implications (embraced by a necessity operator) quite close to logic program rules. This normal form may be useful both for a practical implementation of TEL and a simpler analysis of theoretical problems.

1 Introduction

Logic programs under the answer set (or stable model) semantics [1] have become a succesful paradigm for practical knowledge representation. The success of *Answer Set Programming* (ASP) partly comes from a combination of solid theoretical foundations with the availability of efficient solvers [2] that allowed its use for real world applications. Among these typical applications of ASP we frequently find dealing with transition systems and action theories. In this setting, the nonmonotonic reasoning capabilities of ASP play a crucial role for a suitable treatment of problems like prediction, explanation, planning or diagnostics, allowing a natural representation of default rules like the well-known *inertia default* for solving the frame problem [3]. However, the use of ASP solvers for action domains has an important limitation: it requires fixing a finite length for the sequence of transitions *a priori*, so that the program can be properly grounded. In this way, it is impossible to deal with problems like the non-existence of solution (of any length) for a given planning problem or the study of properties like the equivalence of two representations.

A natural choice for dealing with this kind of problems is extending ASP with modal operators, as those used in Propositional Linear Temporal Logic [4] (LTL). Defining such an extension becomes quite straightforward if we start from a purely logical characterisation of ASP, like the one provided by Equilibrium Logic [5,6].

* This research was partially supported by Spanish MEC project TIN2009-14562-C05-04 and Xunta de Galicia project INCITE08-PXIB105159PR.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 64–76, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Equilibrium Logic has been proved to be a powerful tool for the theoretical analysis of ASP, motivating the study of strong equivalence[1] between logic programs [7], covering most syntactic extensions considered up to date, or being closely related to the conception of new definitions of stable models for arbitrary propositional [8] and first order theories [9]. Another important advantage is that its formal definition is extremely simple: it amounts to a selection criterion among models of the (monotonic) intermediate logic of *Here-and-There* (HT) [10].

An extension of Equilibrium Logic for dealing with LTL operators was first introduced in [11] under the name of *Temporal Equilibrium Logic* (TEL). This modal extension has been already used for encoding action languages [11] or for checking strong equivalence of temporal logic programs by a reduction to LTL [12]. However, the interest of TEL has mostly remained theoretical, as there does not exist any automated method for computing the temporal equilibrium models of an arbitrary modal theory yet. An important step in this direction has to do with reducing the arbitrary syntax of temporal theories into a normal form closer to logic programming rules. For instance, in the non-temporal case, it has been already proved [13] that any arbitrary propositional theory is strongly equivalent to a logic program (allowing disjunction and negation in the head), so that logic programs constitute a normal form for Equilibrium Logic. Similarly, in the case of (monotonic) LTL, an implicational clause-like normal form introduced in [14] was used for designing a temporal resolution method.

In this paper we show that TEL can be similarly reduced (under strong equivalence) to a normal form consisting of a set of implications (embraced by a necessity operator) quite close to logic program rules. The reduction into normal form starts from the structure-preserving polynomial transformation presented in [15] for the non-temporal case. This transformation has as a main feature the introduction of an auxiliary atom per each subformula in the original theory. We then combine this technique with the inductive definitions of temporal operators used for LTL in [14]. The obtained normal form considerably reduces the possible uses of modal operators and may be useful both for a future practical implementation of TEL and a simpler analysis of theoretical problems.

The rest of the paper is organised as follows. In Section 2, we introduce the (monotonic) temporal extension of HT. In the next section, we then define the model selection criterion that gives raise to TEL, providing some concepts and definitions and introducing the normal form. Section 4 details the translation and contains the proof of its correctness. Finally, Section 5 discusses related work and Section 6 concludes the paper.

2 Linear Temporal Here-and-There (THT)

The logic of *Linear Temporal Here-and-There* (THT) is defined as follows. We start from a finite set of atoms V called the *propositional signature*. A (temporal) *formula* is defined as any (well-formed) combination of the classical connectives

[1] Two programs are strongly equivalent when they yield the same answer sets even when they are included in a common larger program or context.

$\wedge, \vee, \rightarrow, \bot$ with the unary temporal operator $\bigcirc$ (read "next"), the binary temporal operators $\mathcal{U}$ (read "until") and $\mathcal{R}$ (read "release"), and the atoms in V. A formula is said to be *non-modal* if it does not contain temporal operators. Negation is defined as $\neg\varphi \stackrel{\text{def}}{=} \varphi \rightarrow \bot$ whereas $\top \stackrel{\text{def}}{=} \neg\bot$. As usual, $\varphi \leftrightarrow \psi$ stands for $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. Other usual temporal operators can be defined in terms of $\mathcal{U}$ and $\mathcal{R}$ as follows:

$$\Box\varphi \stackrel{\text{def}}{=} \bot \,\mathcal{R}\, \varphi \qquad \Diamond\varphi \stackrel{\text{def}}{=} \top \,\mathcal{U}\, \varphi \qquad \varphi \,\mathcal{W}\, \psi \stackrel{\text{def}}{=} (\varphi \,\mathcal{U}\, \psi) \vee \Box\varphi$$

Given a formula Γ, by $size(\Gamma)$ we understand the number of occurrences of atoms and connectives $\wedge, \vee, \rightarrow, \bot, \bigcirc, \mathcal{U}, \mathcal{R}$ in Γ. A *theory* is any set of formulas. When Γ is a finite theory, we assume we deal with the conjunction of all its formulas. For any theory Γ, $subf(\Gamma)$ will denote the set of all subformulas of Γ.

A (temporal) *interpretation* $\mathbf{M}$ is an infinite sequence of pairs $m_i = \langle H_i, T_i\rangle$ with $i = 0, 1, 2, \ldots$ where $H_i \subseteq T_i$ are sets of atoms standing for *here* and *there* respectively. For simplicity, given a temporal interpretation, we write $\mathbf{H}$ (resp. $\mathbf{T}$) to denote the sequence of pair components $H_0, H_1, \ldots$ (resp. $T_0, T_1, \ldots$). Using this notation, we will sometimes abbreviate the interpretation as $\mathbf{M} = \langle \mathbf{H}, \mathbf{T}\rangle$. An interpretation $\mathbf{M} = \langle \mathbf{H}, \mathbf{T}\rangle$ is said to be *total* when $\mathbf{H} = \mathbf{T}$.

Let $\mathbf{M}$ be an interpretation for a signature U and let $V \subset U$. The expression $\mathbf{M} \cap V$ denotes the interpretation $\mathbf{M}$ restricted to signature V, that is $\mathbf{M} \cap V$ is a sequence of pairs $\langle H_i \cap V, T_i \cap V\rangle$ for any $\langle H_i, T_i\rangle$ with $i \geq 0$ in $\mathbf{M}$.

Given an interpretation $\mathbf{M}$ and an integer number $k > 0$, by $\mathbf{M}_k$ we denote a new interpretation that results from "shifting" $\mathbf{M}$ in k positions, that is, the sequence of pairs $\langle H_k, T_k\rangle, \langle H_{k+1}, T_{k+1}\rangle, \langle H_{k+2}, T_{k+2}\rangle, \ldots$ Note that $\mathbf{M}_0 = \mathbf{M}$.

Definition 1 (satisfaction). *An interpretation* $\mathbf{M} = \langle \mathbf{H}, \mathbf{T}\rangle$ *satisfies a formula* φ*, written* $\mathbf{M} \models \varphi$*, when:*

1. $\mathbf{M} \models p$ *if* $p \in H_0$*, for any atom* p*.*
2. $\mathbf{M} \models \varphi \wedge \psi$ *if* $\mathbf{M} \models \varphi$ *and* $\mathbf{M} \models \psi$*.*
3. $\mathbf{M} \models \varphi \vee \psi$ *if* $\mathbf{M} \models \varphi$ *or* $\mathbf{M} \models \psi$*.*
4. $\langle \mathbf{H}, \mathbf{T}\rangle \models \varphi \rightarrow \psi$ *if* $\langle x, \mathbf{T}\rangle \not\models \varphi$ *or* $\langle x, \mathbf{T}\rangle \models \psi$ *for all* $x \in \{\mathbf{H}, \mathbf{T}\}$*.*
5. $\mathbf{M} \models \bigcirc\varphi$ *if* $\mathbf{M}_1 \models \varphi$*.*
6. $\mathbf{M} \models \varphi \,\mathcal{U}\, \psi$ *if* $\exists j \geq 0$*,* $\mathbf{M}_j \models \psi$ *and* $\forall k$ *s.t.* $0 \leq k < j$*,* $\mathbf{M}_k \models \varphi$
7. $\mathbf{M} \models \varphi \,\mathcal{R}\, \psi$ *if* $\forall j \geq 0$*,* $\mathbf{M}_j \models \psi$ *or* $\exists k$ *s.t.* $0 \leq k < j$*,* $\mathbf{M}_k \models \varphi$

A formula φ is *valid* if $\mathbf{M} \models \varphi$ for any $\mathbf{M}$. An interpretation $\mathbf{M}$ is a *model* of a theory Γ, written $\mathbf{M} \models \Gamma$, if $\mathbf{M} \models \alpha$, for all formula $\alpha \in \Gamma$.

We assume that a finite sequence $\mathbf{M} = m_1, m_2, \ldots, m_n$ is an abbreviation of an infinite sequence where the remaining elements coincide with m_n, that is, that for $i > n$, $m_i = m_n$. The logic of THT is an orthogonal combination of the logic of HT and the (standard) linear temporal logic (LTL) [4]. When we restrict temporal interpretations to finite sequences of length 1, that is $\langle H_0, T_0\rangle$ and disregard temporal operators, we obtain the logic of HT. On the other hand, if we restrict the semantics to total interpretations, $\langle \mathbf{T}, \mathbf{T}\rangle \models \varphi$ corresponds to satisfaction of formulas $\mathbf{T} \models \varphi$ in LTL. In this sense, item 4 of Definition 1 can be rephrased as:

4'. $\langle \mathbf{H}, \mathbf{T} \rangle \models \varphi \rightarrow \psi$ if both (1) $\langle \mathbf{H}, \mathbf{T} \rangle \models \varphi$ implies $\langle \mathbf{H}, \mathbf{T} \rangle \models \psi$; and (2) $\mathbf{T} \models \varphi \rightarrow \psi$ in LTL.

Similarly $\langle \mathbf{H}, \mathbf{T} \rangle \models \varphi \leftrightarrow \psi$ if both (1) $\langle \mathbf{H}, \mathbf{T} \rangle \models \varphi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle \models \psi$; and (2) $\mathbf{T} \models \varphi \leftrightarrow \psi$ in LTL. The following proposition can also be easily checked.

Proposition 1. *For any* Γ *and any* $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$*, if* $\mathbf{M} \models \Gamma$ *then* $\mathbf{T} \models \Gamma$*.* □

The next result shows that, for formulas not containing implications, equivalence in LTL and THT coincides.

Proposition 2. *Let* φ *and* ψ *be two formulas not containing implication*[2]*. Then* $\varphi \leftrightarrow \psi$ *is a THT tautology iff it is an LTL tautology.*

Proof. As LTL models correspond to THT total models, it is obvious that any THT tautology is an LTL tautology too. For the other direction, assume $\varphi \leftrightarrow \psi$ is LTL valid but for some intepretation $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$, $\mathbf{M} \not\models \varphi \leftrightarrow \psi$. This means that, either (i) $\mathbf{T} \models \varphi$ is not equivalent to $\mathbf{T} \models \psi$ or (ii) $\mathbf{M} \models \varphi$ is not equivalent to $\mathbf{M} \models \psi$. The former immediately contradicts that $\varphi \leftrightarrow \psi$ is an LTL tautology. So, suppose (ii) and, without loss of generality, that $\mathbf{M} \models \varphi$ but $\mathbf{M} \not\models \psi$. Looking at the definition of THT satisfaction, it is easy to observe that the only way to refer to the $\mathbf{T}$ component in $\langle \mathbf{H}, \mathbf{T} \rangle$ is via implication. Since φ and ψ do not contain implications, the $\mathbf{T}$ component is irrelevant and we conclude that for any interpretation $\mathbf{M}' = \langle \mathbf{H}, \mathbf{T}' \rangle$, $\mathbf{M}' \models \varphi$ and $\mathbf{M}' \not\models \psi$, including the case $\mathbf{M}' = \langle \mathbf{H}, \mathbf{H} \rangle$. But this means there exists a LTL intepretation $\mathbf{H}$ for which $\mathbf{H} \models \varphi$ and $\mathbf{H} \not\models \psi$ contradicting that $\varphi \leftrightarrow \psi$ is an LTL tautology. □

In particular, the following LTL valid formulas are also THT valid:

$$\varphi \,\mathcal{U}\, \psi \leftrightarrow \psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi)) \tag{1}$$

$$\varphi \,\mathcal{R}\, \psi \leftrightarrow \psi \wedge (\varphi \vee \bigcirc(\varphi \,\mathcal{R}\, \psi)) \tag{2}$$

We can alternatively represent any interpretation $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ by seeing each $m_i = \langle H_i, T_i \rangle$ as a three-valued mapping $m_i : V \rightarrow \{0, 1, 2\}$ so that, for any atom p, $m_i(p) = 0$ when $p \notin T_i$ (the atom is false), $m_i(p) = 2$ when $p \in H_i$ (the atom is true), and $m_i(p) = 1$ when $p \in T_i \setminus H_i$ (the atom is undefined). We can then define a valuation for any formula φ, written[3] $\mathbf{M}(\varphi)$, by similarly considering which formulas are satisfied by $\langle \mathbf{H}, \mathbf{T} \rangle$ (which will be assigned 2), not satisfied by $\langle \mathbf{T}, \mathbf{T} \rangle$ (which will be assigned 0) or none of the two (which will take value 1). By $\mathbf{M}_i(\varphi)$ we mean the 3-valuation of φ induced by the temporal interpretation $\mathbf{M}_i$, that is, $\mathbf{M}$ shifted i positions. From the definitions in the previous section, we can easily derive the following conditions:

[2] Remember that negation is a form of implication.

[3] We use the same name $\mathbf{M}$ for a temporal interpretation and for its induced three-valued valuation function – ambiguity is removed by the way in which it is applied (a structure or a function on formulas).

1. $\mathbf{M}(p) \stackrel{\text{def}}{=} m_0(p)$
2. $\mathbf{M}(\varphi \wedge \psi) \stackrel{\text{def}}{=} min(\mathbf{M}(\varphi), \mathbf{M}(\psi)); \quad \mathbf{M}(\varphi \vee \psi) \stackrel{\text{def}}{=} max(\mathbf{M}(\varphi), \mathbf{M}(\psi))$
3. $\mathbf{M}(\varphi \rightarrow \psi) \stackrel{\text{def}}{=} \begin{cases} 2 & \text{if } \mathbf{M}(\varphi) \leq \mathbf{M}(\psi) \\ \mathbf{M}(\psi) & \text{otherwise} \end{cases}$
4. $\mathbf{M}(\bigcirc\varphi) \stackrel{\text{def}}{=} \mathbf{M}_1(\varphi)$
5. $\mathbf{M}(\varphi \; \mathcal{U} \; \psi) \stackrel{\text{def}}{=} \begin{cases} 2 \text{ if } \exists j \geq 0 : \mathbf{M}_j(\psi) = 2 \text{ and } \forall k, 0 \leq k < j \Rightarrow \mathbf{M}_k(\varphi) = 2 \\ 0 \text{ if } \forall j \geq 0 : \mathbf{M}_j(\psi) = 0 \text{ or } \exists k, 0 \leq k < j, \mathbf{M}_k(\varphi) = 0 \\ 1 \text{ otherwise} \end{cases}$
6. $\mathbf{M}(\varphi \; \mathcal{R} \; \psi) \stackrel{\text{def}}{=} \begin{cases} 2 \text{ if } \forall j \geq 0 : \mathbf{M}_j(\psi) = 2 \text{ or } \exists k, 0 \leq k < j, \mathbf{M}_k(\varphi) = 2 \\ 0 \text{ if } \exists j \geq 0 : \mathbf{M}_j(\psi) = 0 \text{ and } \forall k, 0 \leq k < j \Rightarrow \mathbf{M}_k(\varphi) = 0 \\ 1 \text{ otherwise} \end{cases}$

From their definition, the interpretation of the temporal derived operators becomes $\mathbf{M}(\Box\varphi) = min\ \{\mathbf{M}_i(\varphi) \mid i \geq 0\}$ and $\mathbf{M}(\Diamond\varphi) = max\ \{\mathbf{M}_i(\varphi) \mid i \geq 0\}$.

Under this alternative three-valued definition, an interpretation $\mathbf{M}$ *satisfies* a formula φ when $\mathbf{M}(\varphi) = 2$. When $\mathbf{M} = \langle \mathbf{T}, \mathbf{T} \rangle$, its induced valuation will be just written as $\mathbf{T}(\varphi)$ and obviously becomes a two-valued function, that is $\mathbf{T}(\varphi) \in \{0, 2\}$. A pair of useful observations:

Observation 1. *For any interpretation* $\mathbf{M}$, $\mathbf{M} \models \varphi \leftrightarrow \psi$ *iff* $\mathbf{M}(\varphi) = \mathbf{M}(\psi)$ *whereas,* $\mathbf{M} \models \Box(\varphi \leftrightarrow \psi)$ *iff for all* $i \geq 0$, $\mathbf{M}_i(\varphi) = \mathbf{M}_i(\psi)$.

Observation 2. *Given* $\mathbf{M} = \{\mathbf{H}, \mathbf{T}\}$ *and a pair of formulas* φ, ψ, *if* $\mathbf{M}(\varphi) = \mathbf{M}(\psi)$ *then also* $\mathbf{T}(\varphi) = \mathbf{T}(\psi)$. □

3 Linear Temporal Equilibrium Logic (TEL)

We can now proceed to describe the model selection criterion that defines temporal equilibrium models. Given two interpretations $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$ and $\mathbf{M}' = \langle \mathbf{H}', \mathbf{T}' \rangle$ we say that $\mathbf{M}'$ is *lower or equal than* $\mathbf{M}$, written $\mathbf{M}' \leq \mathbf{M}$, when $\mathbf{T}' = \mathbf{T}$ and for all $i \geq 0$, $H'_i \subseteq H_i$. As usual, $\mathbf{M}' < \mathbf{M}$ stands for $\mathbf{M}' \leq \mathbf{M}$ but $\mathbf{M}' \neq \mathbf{M}$.

Definition 2 (Temporal Equilibrium Model). *An interpretation* $\mathbf{M}$ *is a* temporal equilibrium model *of a theory* Γ *if* $\mathbf{M}$ *is a total model of* Γ *and there is no other* $\mathbf{M}' < \mathbf{M}$, $\mathbf{M}' \models \Gamma$. □

Note that any temporal equilibrium model is total, that is, it has the form $\langle \mathbf{T}, \mathbf{T} \rangle$ and so can be actually seen as an interpretation $\mathbf{T}$ in the standard LTL. By $Eq(V, \Gamma)$ we denote the set of temporal equilibrium models under signature V of a theory $\Gamma \subseteq \mathcal{L}_V$. Note that the consequence relation induced by temporal equilibrium models is nonmonotonic. Thus, when dealing with equivalence of two theories, Γ_1, Γ_2, the mere coincidence of equilibrium models $Eq(V, \Gamma_1) =$

$Eq(V, \Gamma_2)$ will not suffice for safely replacing one by each other, since they may behave in a different way in the presence of additional information. Two theories Γ_1, Γ_2 are said to be *strongly equivalent* when $Eq(V, \Gamma_1 \cup \Gamma) = Eq(V, \Gamma_2 \cup \Gamma)$ for any arbitrary theory Γ.

We will further refine this idea of strong equivalence for transformations that deal with an extended signature possibly containing auxiliary atoms.

Definition 3 (Strong faithfulness). *We say that a translation $\sigma(\Gamma) \subseteq \mathcal{L}_U$ of some theory $\Gamma \subseteq \mathcal{L}_V$ with $V \subseteq U$ is* strongly faithful *if, for any theory $\Gamma' \subseteq \mathcal{L}_V$:*

$$Eq(V, \Gamma \cup \Gamma') = \{\mathbf{M} \cap V \mid \mathbf{M} \in Eq(U, \sigma(\Gamma) \cup \Gamma')\}$$

Finally, we describe the normal form we are interested in. Given a signature V, we define a *temporal literal* as any expression in the set[4] $\{p, \bigcirc p, \neg p \mid p \in V\}$.

Definition 4 (Temporal rule). *A* temporal rule *is either:*

1. *an atom $p \in V$;*
2. *an implication like $\Box(B_1 \wedge \cdots \wedge B_n \rightarrow C_1 \vee \cdots \vee C_m)$ where the B_i and C_j are temporal literals, $n \geq 0$ and $m \geq 0$;*
3. *or an implication like $\Box(\Box p \rightarrow q)$ or like $\Box(p \rightarrow \Diamond q)$ with p, q atoms.* □

A *temporal logic program* (TLP for short) is a finite set of temporal rules.

4 Reduction to Temporal Logic Programs

The translation uses an extended signature $V_{\mathbf{L}}$ that contains an atom (a label) for each non-constant formula in the original language[5] $\mathcal{L}_V$, that is $V_{\mathbf{L}} = \{\mathbf{L}_\varphi \mid \varphi \in \mathcal{L}_V \setminus \{\bot, \top\}\}$. For convenience, we use $\mathbf{L}_\varphi \stackrel{\text{def}}{=} \varphi$ when φ is $\top$, $\bot$ or an atom $p \in V$. This allows us to consider $V_{\mathbf{L}}$ as a superset of V. For any non-atomic formula γ, we call its *definition*, $df(\gamma)$ to:

$$df(\gamma) \stackrel{\text{def}}{=} \begin{cases} \Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\varphi \bullet \mathbf{L}_\psi) & \text{if } \gamma = (\varphi \bullet \psi) \text{ with } \bullet \in \{\wedge, \vee, \rightarrow\}; \\ \Box(\mathbf{L}_\gamma \leftrightarrow \bigcirc \mathbf{L}_\varphi) & \text{if } \gamma = \bigcirc\varphi; \\ \Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\psi \vee (\mathbf{L}_\varphi \wedge \bigcirc \mathbf{L}_\gamma)\ ) & \\ \quad \wedge \Box(\mathbf{L}_\gamma \rightarrow \Diamond \mathbf{L}_\psi) & \text{if } \gamma = (\varphi\ \mathcal{U}\ \psi); \\ \Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\psi \wedge (\mathbf{L}_\varphi \vee \bigcirc \mathbf{L}_\gamma)\ ) & \\ \quad \wedge \Box(\Box \mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma) & \text{if } \gamma = (\varphi\ \mathcal{R}\ \psi). \end{cases}$$

Definition 5. *For any theory Γ in $\mathcal{L}_V$, we define the translation $\sigma(\Gamma)$ as:*

$$\sigma(\Gamma) \stackrel{\text{def}}{=} \{\mathbf{L}_\varphi \mid \varphi \in \Gamma\} \cup \{df(\gamma) \mid \gamma \in subf(\Gamma)\}$$

[4] Expressions like $\neg \bigcirc p$ are not temporal literals: the three types above suffice.

[5] In this way, $V_{\mathbf{L}}$ is infinite, but when we later translate a given theory Γ, we can just take $V_{\mathbf{L}}$ as a label per each subformula.

That is, $\sigma(\Gamma)$ collects the labels for all the formulas in Γ plus the definitions for all the subformulas in Γ. When the main connective in γ is a derived operator $\neg, \Diamond, \Box$, after simplifying truth constants, we obtain the following $df(\gamma)$:

$$df(\gamma) = \begin{cases} \Box(\mathbf{L}_\gamma \leftrightarrow \neg \mathbf{L}_\varphi\ ) & \text{if } \gamma = \neg\varphi; \\ \Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\varphi \vee \bigcirc \mathbf{L}_\gamma\ ) \wedge \Box(\mathbf{L}_\gamma \rightarrow \Diamond \mathbf{L}_\varphi) & \text{if } \gamma = \Diamond\varphi; \\ \Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\varphi \wedge \bigcirc \mathbf{L}_\gamma\ ) \wedge \Box(\Box \mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma) & \text{if } \gamma = \Box\varphi. \end{cases}$$

Lemma 1. *Let* $\mathbf{M}$ *be a model of a theory* Γ *in* $\mathcal{L}_V$*. Then, there exists some* $\mathbf{M}'$ *such that* $\mathbf{M} = \mathbf{M}' \cap V$ *and* $\mathbf{M}' \models \sigma(\Gamma)$*.*

Proof. Take $\mathbf{M}'$ as the sequence of 3-valued mappings $\mathbf{M}' = m'_1, m'_2, \ldots$ for signature $V_\mathbf{L}$ so that:

$$m'_i(\mathbf{L}_\varphi) \stackrel{\text{def}}{=} \mathbf{M}_i(\varphi) \tag{3}$$

for any formula $\varphi \in \mathcal{L}_V$. When φ is an atom p, $m'_i(p) = m'_i(\mathbf{L}_p) = \mathbf{M}_i(p) = m_i(p)$ for all $i \geq 0$, thus, the valuations for atoms in $\mathbf{M}$ and $\mathbf{M}'$ coincide. This means that $\mathbf{M}' \cap V = \mathbf{M}$.

Furthermore, as $\mathbf{M} \models \Gamma$, for any $\varphi \in \Gamma$ we get $2 = \mathbf{M}(\varphi) = \mathbf{M}_0(\varphi) \stackrel{(3)}{=} m'_0(\mathbf{L}_\varphi) = \mathbf{M}'(\mathbf{L}_\varphi)$. In other words $\mathbf{M}' \models \{\mathbf{L}_\varphi \mid \varphi \in \Gamma\}$. To prove that $\mathbf{M}' \models \sigma(\Gamma)$ it remains to be shown that $\mathbf{M}' \models df(\gamma)$ for any $\gamma \in subf(\Gamma)$. We will show it by cases, depending on each type of subformula γ.

1. For $\gamma = (\varphi \bullet \psi)$ with $\bullet \in \{\wedge, \vee, \rightarrow\}$ we have to prove $\mathbf{M}'_i(\mathbf{L}_\gamma) = \mathbf{M}'_i(\mathbf{L}_\varphi \bullet \mathbf{L}_\psi)$ for all $i \geq 0$:

$$\begin{aligned} \mathbf{M}'_i(\mathbf{L}_\gamma) = m'_i(\mathbf{L}_{\varphi\bullet\psi}) &\stackrel{(3)}{=} \mathbf{M}_i(\varphi \bullet \psi) = f^\bullet(\mathbf{M}_i(\varphi), \mathbf{M}_i(\psi)) \\ &\stackrel{(3)}{=} f^\bullet(m'_i(\mathbf{L}_\varphi), m'_i(\mathbf{L}_\psi)) = \mathbf{M}'_i(\mathbf{L}_\varphi \bullet \mathbf{L}_\psi) \end{aligned}$$

   where $f^\bullet$ denotes, for each $\bullet \in \{\wedge, \vee, \rightarrow\}$ their corresponding three-valued mappings.
2. For $\gamma = \bigcirc\varphi$ we have:

$$\begin{aligned} \mathbf{M}'_i(\mathbf{L}_\gamma) = m'_i(\mathbf{L}_{\bigcirc\varphi}) &\stackrel{(3)}{=} \mathbf{M}_i(\bigcirc\varphi) = \mathbf{M}_{i+1}(\varphi) \\ &\stackrel{(3)}{=} m'_{i+1}(\mathbf{L}_\varphi) = \mathbf{M}'_{i+1}(\mathbf{L}_\varphi) = \mathbf{M}'_i(\bigcirc \mathbf{L}_\varphi) \end{aligned}$$

3. For $\gamma = (\varphi\ \mathcal{U}\ \psi)$ we prove first $\mathbf{M}'_i(\mathbf{L}_\gamma) = \mathbf{M}'_i(\mathbf{L}_\psi \vee (\mathbf{L}_\varphi \wedge \bigcirc \mathbf{L}_\gamma))$ for any $i \geq 0$.

$$\begin{aligned} \mathbf{M}'_i(\mathbf{L}_\gamma) &= m'_i(\mathbf{L}_{\varphi\mathcal{U}\psi}) \stackrel{(3)}{=} \mathbf{M}_i(\varphi\ \mathcal{U}\ \psi) \stackrel{(1)}{=} \mathbf{M}_i(\psi \vee (\varphi \wedge \bigcirc(\varphi \mathcal{U} \psi)\ )\ ) \\ &= max(\ \mathbf{M}_i(\psi),\ min\ (\mathbf{M}_i(\varphi), \mathbf{M}_{i+1}(\varphi\mathcal{U}\psi))\ ) \\ &\stackrel{(3)}{=} max(\ m'_i(\mathbf{L}_\psi),\ min\ (m'_i(\mathbf{L}_\varphi), m'_{i+1}(\mathbf{L}_{\varphi\mathcal{U}\psi}))\ ) \\ &= \mathbf{M}'_i(\mathbf{L}_\psi \vee (\mathbf{L}_\varphi \wedge \bigcirc \mathbf{L}_{\varphi\mathcal{U}\psi})\ ) \end{aligned}$$

We have to prove now that $\mathbf{M}' \models \Box(\mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\psi)$, that is, the implication holds at any $i \geq 0$. Assume at some $i \geq 0$, $\mathbf{M}'_i \models \mathbf{L}_\gamma$. By construction of $\mathbf{M}'$ this means $\mathbf{M}_i \models \varphi \,\mathcal{U}\, \psi$. This implies $\mathbf{M}_i \models \Diamond\psi$, that is, for some $j \geq i$, $\mathbf{M}_j \models \psi$. But then, by construction of $\mathbf{M}'$ again, $\mathbf{M}'_j \models \mathbf{L}_\psi$ for some $j \geq i$, and this implies $\mathbf{M}'_i \models \Diamond\mathbf{L}_\psi$. The same reasoning can be repeated replacing $\mathbf{M}'$ by $\mathbf{T}'$, and thus $\mathbf{M}'_i \models \mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\psi$ for any $i \geq 0$.

4. For $\gamma = (\varphi \,\mathcal{R}\, \psi)$ the proof is completely analogous to 3 replacing the use of equivalence (1) by (2) and exchanging the roles of conjunction/min and disjunction/max. □

Lemma 2. *Let Γ be a THT theory in $\mathcal{L}_V$ and $\mathbf{M}$ a model for $\sigma(\Gamma)$. Then for any $\gamma \in subf(\Gamma)$ and any $i \geq 0$, $\mathbf{M}_i(\mathbf{L}_\gamma) = \mathbf{M}_i(\gamma)$.*

Proof. We use structural induction on γ.

1. When the subformula γ has the shape $\top$, $\bot$ or an atom p this is trivial, since $\mathbf{L}_\gamma = \gamma$ by definition.
2. When $\gamma = \varphi \bullet \psi$ for any connective $\bullet \in \{\wedge, \vee, \rightarrow\}$ then:

$$\begin{aligned}
\mathbf{M}_i(\mathbf{L}_{\varphi\bullet\psi}) &= \mathbf{M}_i(\mathbf{L}_\varphi \bullet \mathbf{L}_\psi) && \text{because } \mathbf{M} \models df(\varphi \bullet \psi)\\
&= f^\bullet(\, \mathbf{M}_i(\mathbf{L}_\varphi), \mathbf{M}_i(\mathbf{L}_\psi)\,)\\
&= f^\bullet(\mathbf{M}_i(\varphi), \mathbf{M}_i(\psi)) && \text{applying induction on } \mathbf{L}_\varphi, \mathbf{L}_\psi\\
&= \mathbf{M}_i(\varphi \bullet \psi)
\end{aligned}$$

3. When $\gamma = \bigcirc\varphi$:

$$\begin{aligned}
\mathbf{M}_i(\mathbf{L}_{\bigcirc\varphi}) &= \mathbf{M}_i(\bigcirc\mathbf{L}_\varphi) \text{ because } \mathbf{M} \models df(\bigcirc\varphi)\\
&= \mathbf{M}_{i+1}(\mathbf{L}_\varphi)\\
&= \mathbf{M}_{i+1}(\varphi) \quad \text{applying induction on } \mathbf{L}_\varphi\\
&= \mathbf{M}_i(\bigcirc\varphi)
\end{aligned}$$

4. When $\gamma = (\varphi \,\mathcal{U}\, \psi)$, if we apply structural induction using $df(\gamma)$ as we did in the previous cases, we can only prove that, for any $i \geq 0$:

$$\begin{aligned}
\mathbf{M}_i(\mathbf{L}_\gamma) &= \mathbf{M}_i(\mathbf{L}_\psi \vee \mathbf{L}_\varphi \wedge \bigcirc\mathbf{L}_\gamma) \quad \text{because } \mathbf{M} \models df(\varphi \,\mathcal{U}\, \psi)\\
&= \mathbf{M}_i(\psi \vee (\varphi \wedge \bigcirc\mathbf{L}_\gamma)\,) \quad \text{by induction on } \mathbf{L}_\varphi, \mathbf{L}_\psi \qquad (4)
\end{aligned}$$

but we cannot get rid of $\mathbf{L}_\gamma$, since γ itself is the formula to be proved in the induction step. To prove $\mathbf{M}_i(\mathbf{L}_\gamma) = \mathbf{M}_i(\gamma)$, we will equivalently show that $\mathbf{M}_i \models \mathbf{L}_\gamma \leftrightarrow \gamma$, i.e., both ($\mathbf{M}_i \models \mathbf{L}_\gamma$ iff $\mathbf{M}_i \models \gamma$) and ($\mathbf{T}_i \models \mathbf{L}_\gamma$ iff $\mathbf{T}_i \models \gamma$).

4.a We prove first the two directions of $\mathbf{M}_i \models \mathbf{L}_\gamma$ iff $\mathbf{M}_i \models \gamma$.

From left to right, given $\mathbf{M}_i \models \mathbf{L}_\gamma$ we get $\mathbf{M}_i \models \psi \vee (\varphi \wedge \bigcirc\mathbf{L}_\gamma)$ due to (4). From $\mathbf{M} \models df(\gamma)$ we also conclude $\mathbf{M}_i \models \mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\psi$ and thus $\mathbf{M}_i \models \Diamond\mathbf{L}_\psi$. Applying structural induction on $\mathbf{L}_\psi$, we get $\mathbf{M}_i \models \Diamond\psi$. But then, there exists $j \geq i$ such that $\mathbf{M}_j \models \psi$. Take the smallest j satisfying $\mathbf{M}_j \models \psi$, so that we further have $\mathbf{M}_k \not\models \psi$ for any k, $i \leq k < j$ (when $j = i$ we simply have no k). We will inductively prove that $\mathbf{M}_k \models \varphi \wedge \bigcirc\mathbf{L}_\gamma$ for all $k = i, i+1, \ldots, j-1$ which, together with $\mathbf{M}_j \models \psi$ implies $\mathbf{M}_i \models (\varphi \,\mathcal{U}\, \psi) = \gamma$. For $j = i$ this is

trivial, so take $j > i$. For $k = i$ we know $\mathbf{M}_i \not\models \psi$ and so $\mathbf{M}_i \models \varphi \wedge \bigcirc \mathbf{L}_\gamma$. Assume proved for k with $i \leq k < j-1$ and we want to prove it for $k+1$. By induction, $\mathbf{M}_k \models \bigcirc \mathbf{L}_\gamma$ which is equivalent to $\mathbf{M}_{k+1} \models \mathbf{L}_\gamma$. This corresponds in its turn to $\mathbf{M}_{k+1} \models \psi \vee (\varphi \wedge \bigcirc \mathbf{L}_\gamma)$ but as $k+1 < j$ we also have $(\mathbf{M}, k+1) \not\models \psi$ so that $\mathbf{M}_{k+1} \models \varphi \wedge \bigcirc \mathbf{L}_\gamma$.

From right to left, suppose $\mathbf{M}_i \models \gamma$, that is, $\mathbf{M}_i \models \varphi \,\mathcal{U}\, \psi$. This means there exists some $j \geq i$ such that $\mathbf{M}_j \models \psi$ and $\mathbf{M}_k \models \varphi$ for all k, $i \leq k < j$. We will inductivley show that for any $k = j, j-1, \ldots, i$, $\mathbf{M}_k \models \mathbf{L}_\gamma$ which includes the case $k = i$ we really want to prove. For $k = j$, we saw that $\mathbf{M}_j \models \psi$ and, from (4), this implies $\mathbf{M}_j \models \mathbf{L}_\gamma$. Assume proved for $k+1$ with $i \leq k < j$ and we want to prove it for k. As $i \leq k < j$, we had that $\mathbf{M}_k \models \varphi$. On the other hand, by induction $\mathbf{M}_{k+1} \models \mathbf{L}_\gamma$ and so $\mathbf{M}_k \models \bigcirc \mathbf{L}_\gamma$. Altogether, we get $\mathbf{M}_j \models \varphi \wedge \bigcirc \mathbf{L}_\gamma$ which again from (4) implies $\mathbf{M}_k \models \mathbf{L}_\gamma$.

4.b Now, we must prove $\mathbf{T}_i \models \mathbf{L}_\gamma$ iff $\mathbf{T}_i \models \gamma$. Note that, due to Observation 2, the inductive hypothesis $\mathbf{M}_i(\varphi) = \mathbf{M}_i(\mathbf{L}_\varphi)$ also holds for $\mathbf{T}_i(\varphi) = \mathbf{T}_i(\mathbf{L}_\varphi)$, and the same happens with subformula ψ. Following the same reasoning as in (4) we also have $\mathbf{T}_i(\mathbf{L}_\gamma) = \mathbf{T}_i(\psi \vee (\varphi \wedge \bigcirc \mathbf{L}_\gamma))$. From $\mathbf{M}' \models df(\gamma)$ we also conclude $\mathbf{T}_i \models \mathbf{L}_\gamma \rightarrow \Diamond \mathbf{L}_\psi$. Using these premises, it is easy to check that the proof of 4.a still applies when replacing $\mathbf{M}$ by $\mathbf{T}$.

5. When $\gamma = (\varphi \,\mathcal{R}\, \psi)$, analogously to case 4 we can use $df(\gamma)$ to obtain, for any $i \geq 0$: $\mathbf{M}_i(\mathbf{L}_\gamma) = \mathbf{M}_i(\psi \wedge (\varphi \vee \bigcirc \mathbf{L}_\gamma))$. The rest of the proof is dual to case 4 switching the roles of $\wedge$ and $\vee$, and of '$\models$' with '$\not\models$.' □

Theorem 1. *For any theory Γ in $\mathcal{L}_V$:* $\{\mathbf{M} \mid \mathbf{M} \models \Gamma\} = \{\mathbf{M}' \cap V \mid \mathbf{M}' \models \sigma(\Gamma)\}$.

Proof. The '$\subseteq$' direction immediately follows from Lemma 1. For proving the '$\supseteq$' direction, suppose we have some $\mathbf{M}'$ model of $\sigma(\Gamma)$. This implies $\mathbf{M}' \models \{\mathbf{L}_\varphi \mid \varphi \in \Gamma\}$, i.e. $\mathbf{M}'(\mathbf{L}_\varphi) = 2$ for all $\varphi \in \Gamma$. As $\Gamma \subseteq subf(\Gamma)$, we can apply Lemma 2 to conclude $\mathbf{M}'_i(\mathbf{L}_\varphi) = \mathbf{M}'_i(\varphi)$ for any $i \geq 0$. But then $\mathbf{M}'_0(\varphi) = 2$ for any $\varphi \in \Gamma$, that is, $\mathbf{M}' \models \Gamma$. Finally, it follows that $\mathbf{M}' \cap V \models \Gamma$ since Γ is a theory in language $\mathcal{L}_V$. □

Clearly, including an arbitrary theory $\Gamma' \subseteq \mathcal{L}_V$ in Theorem 1 as follows $\{\mathbf{M} \mid \mathbf{M} \models \Gamma \cup \Gamma'\} = \{\mathbf{M}' \cap V \mid \mathbf{M}' \models \sigma(\Gamma) \cup \Gamma'\}$ and then taking the minimal models on both sides trivially preserves the equality.

Corollary 1. *Translation $\sigma(\Gamma)$ is strongly faithful.*

Transformation $\sigma(\Gamma)$ is obviously modular, and its polynomial complexity can be easily deduced, but is not a temporal logic program yet, as it contains nested implications. However, we can apply some simple transformations on implication, conjunction and disjunction that have been shown to be strongly equivalent at the (non-temporal) propositional level[6] [15], and obtain a TLP without changing

[6] These transformations for propositional operators contain expressions that are redundant in classical logic, but not in the logic of Here-and-There. The method in [16] can be used to show that these are, in fact, their possible minimal representations as sets of program rules.

γ	$df(\gamma)$	$df^*(\gamma)$
$\varphi \wedge \psi$	$\Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\varphi \wedge \mathbf{L}_\psi)$	$\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\varphi)$ $\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\psi)$ $\Box(\mathbf{L}_\varphi \wedge \mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$
$\varphi \vee \psi$	$\Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\varphi \vee \mathbf{L}_\psi)$	$\Box(\mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\varphi \vee \mathbf{L}_\psi)$
$\varphi \rightarrow \psi$	$\Box(\mathbf{L}_\gamma \leftrightarrow (\mathbf{L}_\varphi \rightarrow \mathbf{L}_\psi)$	$\Box(\mathbf{L}_\gamma \wedge \mathbf{L}_\varphi \rightarrow \mathbf{L}_\psi)$ $\Box(\neg\mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\varphi \vee \neg\mathbf{L}_\psi \vee \mathbf{L}_\gamma)$
$\varphi \,\mathcal{U}\, \psi$	$\Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\psi \vee (\mathbf{L}_\varphi \wedge \bigcirc\mathbf{L}_\gamma))$ $\wedge\, \Box(\mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\psi)$	$\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\psi \vee \mathbf{L}_\varphi)$ $\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\psi \vee \bigcirc\mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\varphi \wedge \bigcirc\mathbf{L}_\gamma \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\psi)$
$\varphi \,\mathcal{R}\, \psi$	$\Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\psi \wedge (\mathbf{L}_\varphi \vee \bigcirc\mathbf{L}_\gamma))$ $\wedge\, \Box(\Box\mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$	$\Box(\mathbf{L}_\psi \wedge \mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\psi \wedge \bigcirc\mathbf{L}_\gamma \rightarrow \mathbf{L}_\gamma)$ $\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\psi)$ $\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\varphi \vee \bigcirc\mathbf{L}_\gamma)$ $\Box(\Box\mathbf{L}_\psi \rightarrow \mathbf{L}_\gamma)$

Fig. 1. Transformation $\sigma^*(\gamma)$ generating a temporal logic program

the signature $V_{\mathbf{L}}$. For each definition $df(\gamma)$, we define the strongly equivalent set (understood as the conjunction) of temporal logic program rules $df^*(\gamma)$ as shown in Figure 1. The temporal logic program $\sigma^*(\Gamma)$ is obtained by replacing in $\sigma(\Gamma)$ each subformula definition $df(\varphi)$ by the corresponding set of rules $df^*(\varphi)$. Note that, as $\sigma^*(\Gamma)$ is strongly equivalent to $\sigma(\Gamma)$ (under the same vocabulary) it preserves strong faithfulness with respect to Γ. Figure 2 shows the translation that results for derived operators after applying their definitions. To illustrate the effect of σ^* consider the example theory Γ_1 just consisting of $\Box(\neg p \rightarrow q \,\mathcal{U}\, p)$. The translation $\sigma^*(\Gamma_1)$ consists of the conjunction of $\mathbf{L}_4$ plus the rules in the $df^*(\gamma)$ columns of tables in Figure 3.

Although $\sigma^*(\Gamma)$ is systematically applied on any subformula, for a practical implementation, we can frequently avoid the introduction of new labels, when the obtained expressions are already a TLP. For instance, in the example above, it would actually suffice with considering $df^*(p \,\mathcal{U}\, q)$ that introduces label $\mathbf{L}_1$ and then replacing Γ_1 with $\Box(\neg p \rightarrow \mathbf{L}_1)$. The next results shows that $\sigma^*(\Gamma)$ keeps polynomial (in fact, linear) complexity on the size of Γ.

Theorem 2. *Translation $\sigma^*(\Gamma)$ is linear and its size can be bounded as follows:* $size(\sigma^*(\Gamma)) \leq 2\ |\Gamma| + 34\ size(\Gamma)$.

γ	$df^*(\gamma)$	
$\neg\varphi$	$\Box(\mathbf{L}_\gamma \wedge \mathbf{L}_\varphi \rightarrow \bot)$	$\Box(\neg\mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$
$\Diamond\varphi$	$\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\varphi \vee \bigcirc\mathbf{L}_\gamma)$	$\Box(\mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$
	$\Box(\mathbf{L}_\gamma \rightarrow \Diamond\mathbf{L}_\varphi)$	$\Box(\bigcirc\mathbf{L}_\gamma \rightarrow \mathbf{L}_\gamma)$
$\Box\varphi$	$\Box(\mathbf{L}_\varphi \wedge \bigcirc\mathbf{L}_\gamma \rightarrow \mathbf{L}_\gamma)$	$\Box(\mathbf{L}_\gamma \rightarrow \mathbf{L}_\varphi)$
	$\Box(\Box\mathbf{L}_\varphi \rightarrow \mathbf{L}_\gamma)$	$\Box(\mathbf{L}_\gamma \rightarrow \bigcirc\mathbf{L}_\gamma)$

Fig. 2. Transformation $\sigma^*(\gamma)$ that results for derived operators

γ	$df^*(\gamma)$
$p\ \mathcal{U}\ q$	$\Box(\mathbf{L}_1 \rightarrow p \vee q)$
	$\Box(\mathbf{L}_1 \rightarrow p \vee \bigcirc\mathbf{L}_1)$
	$\Box(p \rightarrow \mathbf{L}_1)$
	$\Box(q \wedge \bigcirc\mathbf{L}_1 \rightarrow \mathbf{L}_1)$
	$\Box(\mathbf{L}_1 \rightarrow \Diamond p)$
$\neg p$	$\Box(\mathbf{L}_2 \wedge p \rightarrow \bot)$
	$\Box(\neg p \rightarrow \mathbf{L}_2)$

γ	$df^*(\gamma)$
$\neg p \rightarrow p\ \mathcal{U}\ q$	$\Box(\mathbf{L}_3 \wedge \mathbf{L}_2 \rightarrow \mathbf{L}_1)$
	$\Box(\neg\mathbf{L}_2 \rightarrow \mathbf{L}_3)$
	$\Box(\mathbf{L}_1 \rightarrow \mathbf{L}_3)$
	$\Box(\mathbf{L}_2 \vee \neg\mathbf{L}_1 \vee \mathbf{L}_3)$
$\Box(\neg p \rightarrow q\ \mathcal{U}\ p)$	$\Box(\mathbf{L}_3 \wedge \bigcirc\mathbf{L}_4 \rightarrow \mathbf{L}_4)$
	$\Box(\mathbf{L}_4 \rightarrow \mathbf{L}_3)$
	$\Box(\mathbf{L}_4 \rightarrow \bigcirc\mathbf{L}_4)$
	$\Box(\Box\mathbf{L}_3 \rightarrow \mathbf{L}_4)$

Fig. 3. Transformation $\sigma^*(\Gamma_1)$ for example theory $\Gamma_1 = \{\Box(\neg p \rightarrow q\ \mathcal{U}\ p)\}$

Proof. Theory $\sigma^*(\Gamma)$ can be written as $\left(\bigwedge_{\gamma\in\Gamma}\mathbf{L}_\gamma\right) \wedge df^*(\Gamma)$. For the size of the first conjunct, we have an atom plus a conjunction connective per each formula in Γ (this also includes the last $\wedge$ connecting to $df^*(\Gamma)$), so we get $2\ |\Gamma|$. The second conjunct, $df^*(\Gamma)$, corresponds to the conjunction of all $df^*(\gamma)$ per each subformula γ in Γ. In the worst case, operators $\mathcal{U}$ and $\mathcal{R}$, we have 5 temporal rules using a total of 13 atom occurrences and 16 connectives. These 5 rules will be joined by 4 implicit conjunctions, and we can use an additional one to join them to the rest of subformulas. Thus, we obtain $size(df^*(\Gamma)) \leq 34\ |subf(\Gamma)|$. Finally, observe that the number of subformulas can be bounded by $size(\Gamma)$ (it will be strictly lower only if repeated subformulas occur). □

5 Discussion and Related Work

It is perhaps interesting to compare the obtained TLP form to the so-called *Separated Normal Form* (SNF) previously introduced in [14] for the case of LTL. An LTL formula is in SNF if it is a conjunction of formulas having one of the following forms[7]:

1. $C_1 \vee \cdots \vee C_n$ — an *initial* rule
2. $\Box(B_1 \vee \cdots \vee B_m \rightarrow \bigcirc C_1 \vee \cdots \vee \bigcirc C_n)$ — a *global* $\Box$-rule
3. $\Diamond C$ — an *initial* $\Diamond$-rule
4. $\Box(B_1 \vee \cdots \vee B_m \rightarrow \Diamond C)$ — a *global* $\Diamond$-rule

[7] For comparison purposes, we have adapted the original formulation that dealt with both future and past operators, to the case in which only future operators are used.

where B_i, C_j, B and C are (non-modal) literals (that is, an atom or its negation). Apart from the minor difference in initial rules, which can be easily removed in favour of auxiliary atoms, we can observe that the main difference is that the bodies of rules do not contain modal operators, whereas the heads always refer to a modal operator, either a disjunction of $\bigcirc C_i$'s or a single $\Diamond C$. In LTL, an obvious way for obtaining SNF from our TLP form would be moving any $\bigcirc B$ in the body to $\bigcirc\neg B$ in the head, and vice versa, moving any non-modal literal C in the head to $\neg C$ in the body, removing double negations afterwards. For instance, a TLP rule like $\Box(\neg p \wedge \bigcirc q \rightarrow r \vee \bigcirc s)$ becomes the LTL-equivalent SNF global $\Box$-rule $\Box(\neg p \wedge \neg r \rightarrow \bigcirc s \vee \bigcirc \neg q)$. For the case of rules like $\Box(\Box p \rightarrow q)$ we could similarly transform them into the SNF global $\Diamond$-rule: $\Box(\neg q \rightarrow \Diamond \neg p)$. Unfortunately, in the logic of HT (even in the non-modal case) exchanging literals between the body and the head in this way is not generally possible. This is because, in this logic, $\neg\neg C \leftrightarrow C$ is not valid. As a result, we cannot replace, for instance $\Box C$ with $\neg\Diamond\neg C$ and we must maintain rules $\Box(p \rightarrow \Diamond q)$ and their dual $\Box(\Box p \rightarrow q)$. By this same reason, in THT it is not possible to define operator $\mathcal{R}$ in terms of $\mathcal{U}$ or vice versa, by just applying De Morgan laws.

Another interesting question is why the $\mathcal{U}$ operator cannot be simply encoded with the formula $\Box(\mathbf{L}_\gamma \leftrightarrow \mathbf{L}_\psi \vee (\mathbf{L}_\varphi \wedge \bigcirc \mathbf{L}_\gamma))$ that results from applying the inductive definition (1). The reason is that, in this way, we could infinitely make the auxiliary atom $\mathbf{L}_\gamma$ true without guaranteeing that at some finite future $\mathbf{L}_\psi$ is made true. The latter is accomplished by the formula $\Box(\mathbf{L}_\gamma \rightarrow \Diamond \mathbf{L}_\psi)$. Thus, we cannot get rid of $\Diamond$ operator in the rule heads (as happens in SNF too). The explanation for the $\mathcal{R}$ operator and the use of $\Box$ in the body is completely dual.

6 Conclusions

We have introduced a normal form for Temporal Equilibrium Logic, a formalism that provides an answer set semantics for arbitrary theories in the syntax of propositional linear temporal logic. This normal form, called Temporal Logic Programs, is close to logic programming rules (with disjunction and negation in the head), embraced with necessity operators. As a result, we can disregard the arbitrary nesting of temporal operators, or even the whole use of operators like "until" and "release." Besides, the close similarity of the obtained form to standard logic programming may help in the future to apply well-known techniques like the use of *loop formulas* [17] or the technique of *splitting* [18] to (some families of) temporal programs.

References

1. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming: Proc. of the Fifth International Conference and Symposium, vol. 2, pp. 1070–1080. MIT Press, Cambridge (1988)

2. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The second Answer Set Programming competition. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS (LNAI), vol. 5753, pp. 637–654. Springer, Heidelberg (2009)
3. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence Journal 4, 463–512 (1969)
4. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1991)
5. Pearce, D.: A new logical characterisation of stable models and answer sets. In: Dix, J., Przymusinski, T.C., Moniz Pereira, L. (eds.) NMELP 1996. LNCS(LNAI), vol. 1216. Springer, Heidelberg (1997)
6. Pearce, D.: Equilibrium logic. Annals of Mathematics and Artificial Intelligence 47(1-2), 3–41 (2006)
7. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. Computational Logic 2(4), 526–541 (2001)
8. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
9. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 372–379 (2007)
10. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse, pp. 42–56 (1930)
11. Cabalar, P., Vega, G.P.: Temporal equilibrium logic: a first approach. In: Moreno Díaz, R., Pichler, F., Quesada Arencibia, A. (eds.) EUROCAST 2007. LNCS, vol. 4739, pp. 241–248. Springer, Heidelberg (2007)
12. Aguado, F., Cabalar, P., Pérez, G., Vidal, C.: Strongly equivalent temporal logic programs. In: Hölldobler, S., Lutz, C., Wansing, H. (eds.) JELIA 2008. LNCS (LNAI), vol. 5293, pp. 8–20. Springer, Heidelberg (2008)
13. Cabalar, P., Ferraris, P.: Propositional theories are strongly equivalent to logic programs. Theory and Practice of Logic Programming 7(6), 745–759 (2007)
14. Fisher, M.: A resolution method for temporal logic. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991), pp. 99–104. Morgan Kaufmann Publishers Inc., San Francisco (1991)
15. Cabalar, P., Valverde, A., Pearce, D.: Reducing propositional theories in equilibrium logic to logic programs. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 4–17. Springer, Heidelberg (2005)
16. Cabalar, P., Valverde, A., Pearce, D.: Minimal logic programs. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 104–118. Springer, Heidelberg (2007)
17. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. In: Artificial Intelligence, pp. 112–117 (2002)
18. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proceedings of the 11th International Conference on Logic programming (ICLP 1994), pp. 23–37 (1994)

Rational Closure for Defeasible Description Logics

Giovanni Casini[1] and Umberto Straccia[2]

[1] Scuola Normale Superiore, Pisa, Italy
giovanni.casini@gmail.com
[2] Istituto di Scienza e Tecnologie dell'Informazione (ISTI - CNR), Pisa, Italy
straccia@isti.cnr.it

Abstract. In the field of non-monotonic logics, the notion of *rational closure* is acknowledged as a landmark, and we are going to see that such a construction can be characterised by means of a simple method in the context of propositional logic. We then propose an application of our approach to rational closure in the field of Description Logics, an important knowledge representation formalism, and provide a simple decision procedure for this case.

1 Introduction

A lot of attention has been dedicated to *non-monotonic* reasoning (see, *e.g.* [20]). Relatively less investigated is the application of such reasoning models to Description Logics (DLs) [3]. In what follows we take under consideration one central non-monotonic reasoning model, that is, the *rational closure* [28], and we are going to apply such a construction to $\mathcal{ALC}$, a significant and expressive representative of the various DLs.

The contributions of this work can be summarised as follows: (i) we provide a characterisation of rational closure in the context of propositional logic, based on classical entailment tests only and, thus, amenable of a simple implementation; and (ii) we apply this characterisation to the context of DLs (we provide a construct $C \mathrel{|\!\sim} D$ stating 'an instance of the concept C, typically is an instance of the concept D'), inheriting a simple reasoning procedure to decide entailment under rational closure.

While there have been several non-monotonic extensions of DLs, such as [1, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 21, 22, 23, 24, 25, 27, 31, 32, 34, 35, 36], which integrate several kind of non-monotonic reasoning mechanism into DLs, to the best of our knowledge, none of them address specifically the issue to model rational closure in DLs. Somewhat related to our proposal are [11, 23], but, beside other points, both model rational consequence relations, while we refer to a rational consequence relation that is recognised as particularly well-behaved, that is, rational closure.

We proceed as follows: first, we present a particular construction of the rational closure, based on the default-assumption approach (see, *e.g.* [30, 33]); then we implement such a construction in $\mathcal{ALC}$; in the end, we conclude with a summary of our contribution and future issues we plan to address.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 77–90, 2010.
© Springer-Verlag Berlin Heidelberg 2010

2 Propositional Rational Closure

Consider a finitely generated classical propositional language ℓ, defined in the usual way [1]. We shall use δ to indicate *default formulae*. The symbols $\models$, $\mid\sim$ will represent different kinds of consequence relations. In particular, $\models$ will be the classical consequence relation and $\mid\sim$ a defeasible inference relation. An element of a consequence relation, $\Gamma \mid\sim C$, will be called a *sequent* and has to be read as 'If Γ, then typically C'.

To start with, a *conditional knowledge base* will be characterised by a pair $\langle \mathcal{T}, \mathcal{B} \rangle$, where $\mathcal{T}$ is a set of formulae, representing certain knowledge, and $\mathcal{B}$ is a set of *sequents* $C \mid\sim D$, representing default information (see [28]).

Example 1. The typical 'penguin' example can be encoded as [2]: $\mathcal{K} = \langle \mathcal{T}, \mathcal{B} \rangle$ with $\mathcal{T} = \{P \rightarrow B\}$ and $\mathcal{B} = \{P \mid\sim \neg F, B \mid\sim F\}$. □

Another way to formalise defeasible information may be based on the *default-assumption approach*, where a *default knowledge base* is a pair $\langle \mathcal{T}, \Delta \rangle$, where now Δ is a set of *formulae* representing what the agent considers as typically true.

Example 2. The 'penguin' example can, for instance, be encoded as: $\mathcal{K} = \langle \mathcal{T}, \Delta \rangle$ with $\mathcal{T} = \{P \rightarrow B\}$ and $\Delta = \{(B \rightarrow F) \wedge (P \rightarrow \neg F), P \rightarrow \neg F\}$. □

Our proposal, using the results of Freund [19], will consist in mapping a conditional knowledge base into a default knowledge base (*e.g.*, we will transform the KB in Example 1 into the KB of Example 2), and then we show a simple procedure to reason within the latter, by relying on a decision procedure for $\models$ only. We then suggest to transpose such an approach, into the framework of DLs.

We proceed next in this way: (i) first, we define the notion of *rational consequence relation* (see *e.g.* [29]) and we present the notion of *rational closure*, (ii) then, we briefly present the *default-assumption approach* and show how to map, by preserving rational closure, a conditional knowledge base into a default knowledge base. Eventually, we describe a procedure to build a *rational closure* using the default-assumption approach.

Rational Consequence Relations. A particularly appreciated non-monotonic consequence relation is represented by the class of *rational consequence relations* (see [28]). A consequence relation $\mid\sim$ is *rational* iff it satisfies the following properties:

(REF) $C \mid\sim C$ Reflexivity

(CT) $\dfrac{C \mid\sim D \quad C \wedge D \mid\sim F}{C \mid\sim F}$ Cut (Cumulative Trans.)

(RW) $\dfrac{C \mid\sim D \quad D \models F}{C \mid\sim F}$ Right Weakening

(CM) $\dfrac{C \mid\sim D \quad C \mid\sim F}{C \wedge D \mid\sim F}$ Cautious Monotony

(OR) $\dfrac{C \mid\sim F \quad D \mid\sim F}{C \vee D \mid\sim F}$ Left Disjunction

(LLE) $\dfrac{C \mid\sim F \quad \models C \leftrightarrow D}{D \mid\sim F}$ Left Logical Equival.

(RM) $\dfrac{C \mid\sim F \quad C \not\mid\sim \neg D}{C \wedge D \mid\sim F}$ Rational Monotony

Rational consequence relations represent a particular subclass of the *preferential inference relations* (see [26]), which are defined by the above properties (REF – OR), without (RM); these are generally considered as the core properties defining a satisfying

[1] We use $\neg, \wedge, \vee, \rightarrow$ as connectives, $C, D, \ldots$ as sentences, $\Gamma, \Delta, \ldots$ as finite sets of sentences, $\top$ and $\bot$ as $A \vee \neg A$ and $A \wedge \neg A$ for some A.

[2] Read B as 'Bird', P as 'Penguin' and F as 'Flying'.

non-monotonic inference relation. Hence, rational consequence relations are preferential relations characterised by the property (RM), which can be read as 'If C typically implies F, and we are not aware that C typically implies $\neg D$, then we are authorised to consider F as a typical consequence of $C \wedge D$'. (RM) is generally considered as the strongest form of monotonicity we can use in the characterisation of a reasoning system in order to formalise a well-behaved form of defeasible reasoning.

Semantically, rational consequence relations can be characterised by means of a particular kind of possible-worlds model, that is, ranked preferential models, but we shall not deepen the connection with such a semantical characterisation here (see [28]).

Rational Closure. Consider $\mathcal{B} = \{C_1 \mathrel{|\!\sim} E_1, \ldots, C_n \mathrel{|\!\sim} E_n\}$. We want the agent to be able to reason about its defeasible information, that is, to be able to derive new sequents from his conditional base. A way to derive new default information is by defining a closure operation $\mathbb{P}$ that, given $\mathcal{B}$, gives back a preferential consequence relation $\mathrel{|\!\sim}$ containing the sequents in $\mathcal{B}$ and is closed under the rules (REF)–(OR). Such a closure operation under the rules (REF)–(OR) is unique (see [26], Corollary 1 and p.31). Formally, given $\mathcal{B}$, a sequent $C \mathrel{|\!\sim} D$ is in its preferential closure $\mathbb{P}(\mathcal{B})$ iff it is derivable from $\mathcal{B}$ using the preferential rules (REF)–(OR). However, the preferential closure is generally considered too weak to be satisfactory, and so it is natural to look for stronger forms of closure. The closure under the rule (RM) is considered, between the interesting rules, the strongest one. Lehmann and Magidor have defined in [28] a rational closure operation $\mathbb{R}$ that satisfies a set of desiderata: namely, (i) $\mathbb{P}(\mathcal{B}) \subseteq \mathbb{R}(\mathcal{B})$; (ii) $C \mathrel{|\!\sim} \bot \in \mathbb{R}(\mathcal{B})$ iff $C \mathrel{|\!\sim} \bot \in \mathbb{P}(\mathcal{B})$; (iii) $\top \mathrel{|\!\sim} C \in \mathbb{R}(\mathcal{B})$ iff $\top \mathrel{|\!\sim} C \in \mathbb{P}(\mathcal{B})$; (iv) If $C \mathrel{|\!\sim} F$ in $\mathbb{P}(\mathcal{B})$, and $C \mathrel{|\!\sim} \neg D, C \wedge D \mathrel{|\!\sim} F \notin \mathbb{P}(\mathcal{B})$ then $C \wedge D \mathrel{|\!\sim} F \in \mathbb{R}(\mathcal{B})$ whenever is possible(see [28], Section 5, for the justification of these desiderata). We shall not describe Lehmann and Magidor's rational closure operation referring to [28]. However, we shall directly refer to a correspondent, more simple construction, based on the default-assumption approach and defined by Freund in [19].

Default-Assumption Consequence Relations and Rational Closure. Consider a default knowledge base $\mathcal{K} = \langle \mathcal{T}, \Delta \rangle$. If the agent is confronted with a piece of information Γ, representing what actually holds, then he has to 'merge' the information in Γ with his background theory $\mathcal{T}$ and his default information Δ. Such an interaction is determined by a consistency check, formalised referring to the notion of *maxiconsistent subset*. Formally, let Δ, Φ be two sets of formulae, then Ψ is a *Φ-maxiconsistent subset* of Δ iff (i) $\Psi \subseteq \Delta$; (ii) $\Psi \not\models \neg(\bigwedge \Phi)$; and (iii) there is no set Ψ' such that $\Psi \subset \Psi' \subseteq \Delta$, and $\Psi' \not\models \neg(\bigwedge \Phi)$. Now, to determine what the agent presumes to be true in a situation in which Γ holds, he takes under consideration all the $(\mathcal{T} \cup \Gamma)$-maxiconsistent subsets of Δ, *i.e.* he considers all the default information that is compatible with what he knows to be true. That is, we say that D is a *default-assumption consequence* of the premise set Γ, given a background theory $\mathcal{T}$ and a set of default-assumptions Δ, written $\Gamma \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D$, if and only if D is a classical consequence of the union of Γ with $\mathcal{T}$ and whichever $(\mathcal{T} \cup \Gamma)$-maxiconsistent subset of Δ, *i.e.*

$$\Gamma \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D \textit{ iff } (\mathcal{T} \cup \Gamma \cup \Delta') \models D \textit{ for every } (\mathcal{T} \cup \Gamma)\textit{-maxiconsistent } \Delta' \subseteq \Delta \ .$$

As next, we want to characterize the rational closure by means of the default-assumption construction, *i.e.* we start from a defeasible KB $\langle \mathcal{T}, \mathcal{B} \rangle$ and from it we build a correspon-

dent default KB $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$. So, consider $\langle \mathcal{T}, \mathcal{B} \rangle$, with $\mathcal{B} = \{C_1 |\sim E_1, \ldots, C_n |\sim E_n\}$. The steps for the construction of $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$ (obtained combining the results in [19] with some results from [6]) are the following.

Step 1. We translate $\mathcal{T}$ into a sequential form and add it to $\mathcal{B}$, that is, we move from a characterisation $\langle \mathcal{T}, \mathcal{B} \rangle$ to $\langle \emptyset, \mathcal{B}' \rangle$, where $\mathcal{B}' = \mathcal{B} \cup \{\neg C |\sim \bot \mid C \in \mathcal{T}\}$. Intuitively, C is valid is equivalent to saying that its negation is an absurdity ($\neg C |\sim \bot$) ([6], Section 6.5).

Step 2. We define $\mathit{\Gamma}_{\mathcal{B}'}$ as the set of the *materializations* of the sequents in $\mathcal{B}'$, *i.e.* the material implications corresponding to such sequents: $\mathit{\Gamma}_{\mathcal{B}'} = \{C \rightarrow D \mid C |\sim D \in \mathcal{B}'\}$. Also, we indicate by $\mathfrak{A}_{\mathcal{B}'}$ the set of the antecedents of the sequents in $\mathcal{B}'$: $\mathfrak{A}_{\mathcal{B}'} = \{C \mid C |\sim D \in \mathcal{B}'\}$.

Step 3. Now we define an *exceptionality ranking* of sequents with respect to (w.r.t.) $\mathcal{B}'$:

Step 3.1. Lehmann and Magidor [28] call a formula C *exceptional* for a set of sequents $\mathcal{D}$ iff $\mathcal{D}$ preferentially entails $\top |\sim \neg C$ (*i.e.* $\top |\sim \neg C \in \mathbb{P}(\mathcal{D})$). $C |\sim D$ is said to be exceptional for $\mathcal{D}$ iff its antecedent C is exceptional for $\mathcal{D}$. Exceptionality of sequents can be decided based on $\models$ only (see [28], Corollary 5.22), as C is exceptional for a set of sequents $\mathcal{D}$ (*i.e.* $\top |\sim \neg C \in \mathbb{P}(\mathcal{D})$) iff $\mathit{\Gamma}_{\mathcal{D}} \models \neg C$.

Step 3.2. Given a set of sequents $\mathcal{D}$, indicate by $E(\mathfrak{A}_{\mathcal{D}})$ the set of the antecedents that result exceptional w.r.t. $\mathcal{D}$, that is $E(\mathfrak{A}_{\mathcal{D}}) = \{C \in \mathfrak{A}_{\mathcal{D}} \mid \mathit{\Gamma}_{\mathcal{D}} \models \neg C\}$, and with $E(\mathcal{D})$ the exceptional sequents in $\mathcal{D}$, *i.e.* $E(\mathcal{D}) = \{C |\sim D \in \mathcal{D} \mid C \in E(\mathfrak{A}_{\mathcal{D}})\}$. Obviously, for every $\mathcal{D}$, $E(\mathcal{D}) \subseteq \mathcal{D}$.

Step 3.3. We can construct iteratively a sequence $\mathcal{E}_0, \mathcal{E}_1 \ldots$ of subsets of the conditional base $\mathcal{B}'$ in the following way: $\mathcal{E}_0 = \mathcal{B}'$, $\mathcal{E}_{i+1} = E(\mathcal{E}_i)$. Since $\mathcal{B}'$ is a finite set, the construction will terminate with an empty set ($\mathcal{E}_n = \emptyset$) or a fixed point of E.

Step 3.4. Using such a sequence, we can define a ranking function r that associates to every sequent in $\mathcal{B}'$ a number, representing its level of exceptionality:

$$r(C |\sim D) = \begin{cases} i & \text{if } C |\sim D \in \mathcal{E}_i \text{ and } C |\sim D \notin \mathcal{E}_{i+1} \\ \infty & \text{if } C |\sim D \in \mathcal{E}_i \text{ for every } i\ . \end{cases}$$

Step 4. In Step 3, we defined the materialisation of $\mathcal{B}'$ and the rank of every sequent in it. Now,

Step 4.1. we can determine if $\mathcal{B}'$ is inconsistent. A conditional base is inconsistent if in its preferential closure we obtain the sequent $\top |\sim \bot$ (from this sequent we can derive any other sequent using RW and CM). Given the result in Step 3.1, we can check the consistency of $\mathcal{B}'$ using $\mathit{\Gamma}_{\mathcal{B}'}$: $\top |\sim \bot \in \mathbb{P}(\mathcal{B}')$ iff $\mathit{\Gamma}_{\mathcal{B}'} \models \bot$.

Step 4.2. if $\mathcal{B}'$ is consistent and given the ranking, we define the *background theory* $\widetilde{\mathcal{T}}$ of the agent as $\widetilde{\mathcal{T}} = \{\neg C \mid C |\sim D \in \mathcal{B}' \text{ and } r(C |\sim D) = \infty\}$ [3] (one may verify that $\mathcal{T} \subseteq \widetilde{\mathcal{T}}$).

Step 4.3. once we have $\widetilde{\mathcal{T}}$, we can also identify the set of sequents $\widetilde{\mathcal{B}}$, *i.e.*, the defeasible part of the information contained in $\mathcal{B}'$: $\widetilde{\mathcal{B}} = \{C |\sim D \in \mathcal{B}' \mid r(C |\sim D) < \infty\}$ (one may verify that $\widetilde{\mathcal{B}} \subseteq \mathcal{B}$).

[3] One may easily verify the correctness of this definition referring to the following results in [6]: the definition of *clash* (p.175), Corollary 7.5.2, Definition 7.5.2, and Lemma 7.5.5. It suffices to show that the set of the sequents with ∞ as ranking value represents the greatest clash of $\mathcal{B}$.

Essentially, so far we have moved the non-defeasible knowledge 'hidden' in $\mathcal{B}$ to $\mathcal{T}$.

Step 5. Now we build the default-assumption characterisation of the rational closure of $\langle \widetilde{\mathcal{T}}, \widetilde{B} \rangle$. To do so, we translate $\widetilde{\mathcal{B}}$ into a set of default-assumptions, *i.e.* a set of formulae, $\widetilde{\Delta}$. Specifically, given the rank value of the sequents in $\widetilde{\mathcal{B}}$, we construct a set of default assumptions $\widetilde{\Delta} = \{\delta_0, \ldots, \delta_n\}$ (with n the highest rank-value in $\widetilde{\mathcal{B}}$), with

$$\delta_i = \bigwedge \{C \rightarrow D \mid C \mathrel{|\!\sim} D \in \widetilde{\mathcal{B}} \text{ and } r(C \mathrel{|\!\sim} D) \geq i\} \,. \tag{1}$$

Following this construction, presented by Freund in [19], we obtain a set of default formulae, each one associated with a rank value, s.t. every default formula is classically derivable from the preceding ones, that is, $\delta_i \models \delta_{i+1}$, for $0 \leq i < n$.

Step 6. Given the background theory $\widetilde{\mathcal{T}}$ and the default-assumption set $\widetilde{\Delta}$, we associate to the agent the pair $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$ according to the steps defined so far.

Using [19], Theorem 24, we can prove that the default-assumption characterisation of the agent by means of the pair $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$ is equivalent to the rational closure of the pair $\langle \mathcal{T}, \mathcal{B} \rangle$ defined by Lehmann and Magidor. That is,

Proposition 1. *$\Gamma \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} D$ iff $\Gamma \mathrel{|\!\sim} D \in \mathbb{R}(\mathcal{B}')$.*

As a consequence, using the following knowledge base transformations

$$\mathcal{K} = \langle \mathcal{T}, \mathcal{B} \rangle \rightsquigarrow \langle \emptyset, \mathcal{B}' \rangle \rightsquigarrow \langle \widetilde{\mathcal{T}}, \widetilde{B} \rangle \rightsquigarrow \langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle \,, \tag{2}$$

we can characterise the rational closure of $\langle \mathcal{T}, \mathcal{B} \rangle$ via $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$ by means of Proposition 1. Note that, given the Eq. (1), the default set Δ is linearly ordered by $\delta_0 \models \delta_1 \models \ldots \models \delta_n$. Hence, given a set of premises Γ there will be *just one* $(\Gamma \cup \widetilde{\mathcal{T}})$-maxiconsistent subset of $\widetilde{\Delta}$, represented by a δ_i and every δ_j with $j \geq i$. However, since every such δ_j is classically implied by δ_i, we can associate to the set Γ just the default formula δ_i. Hence we can show that

Proposition 2. *$\Gamma \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} D$ iff $\Gamma \cup \widetilde{\mathcal{T}} \cup \{\delta_i\} \models D$, where δ_i is the first $(\Gamma \cup \widetilde{\mathcal{T}})$-consistent formula[4] of the sequence $\langle \delta_0, \ldots, \delta_n \rangle$.*

So, we have a simple method to decide defeasible consequence under rational closure. Given a defeasible knowledge base $\langle \mathcal{T}, \mathcal{B} \rangle$, certain facts Γ and a formula D,

1. Once for all, apply to $\langle \mathcal{T}, \mathcal{B} \rangle$ the transformations (2);
2. Given Γ, determine δ_i as the first $(\Gamma \cup \widetilde{\mathcal{T}})$-consistent formula of the sequence $\langle \delta_0, \ldots, \delta_n \rangle$.
3. Then decide if D follows under rational closure from Γ w.r.t. $\langle \mathcal{T}, \mathcal{B} \rangle$ by determining whether $\Gamma \cup \widetilde{\mathcal{T}} \cup \{\delta_i\} \models D$.

Furthermore, it is easily verified that all transformations (2) require at most $\mathcal{O}(|\mathcal{K}|)$ entailment tests and, thus, by Proposition 2,

Corollary 1. *Deciding defeasible consequence under rational closure is coNP-complete.*

Hence, the computational complexity does not increase w.r.t. classical entailment.

Let us illustrate the method with the following simple example.

[4] That is, $\widetilde{\mathcal{T}} \cup \Gamma \not\models \neg\delta_i$.

Example 3. Consider Example 1. By **Step 1** we transform $\mathcal{K}$ in $\mathcal{B}' = \{P \wedge \neg B \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F, B \mathrel{|\!\sim} F\}$. By **Step 2**, the set of the materializations of $\mathcal{B}'$ is $\Gamma_{\mathcal{B}'} = \{P \wedge \neg B \rightarrow \bot, P \rightarrow \neg F, B \rightarrow F\}$, with $\mathfrak{A}_{\mathcal{B}'} = \{P \wedge \neg B, P, B\}$. By **Step 3**, we obtain the following exceptionality ranking over the sequents: $\mathcal{E}_0 = \{P \wedge \neg B \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F, B \mathrel{|\!\sim} F\}$, $\mathcal{E}_1 = \{P \wedge \neg B \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F\}$, $\mathcal{E}_2 = \{P \wedge \neg B \mathrel{|\!\sim} \bot\}$ and $\mathcal{E}_3 = \{P \wedge \neg B \mathrel{|\!\sim} \bot\}$. So the ranking value of the sequents is: $r(B \mathrel{|\!\sim} F) = 0$, $r(P \mathrel{|\!\sim} \neg F) = 1$ and $r(P \wedge \neg B \mathrel{|\!\sim} \bot) = \infty$. By **Step 4**, from such a ranking, we obtain a background theory $\widetilde{\mathcal{T}} = \{\neg(P \wedge \neg B)\}$ (hence, the background theory and the defeasible part of the knowledge base were already correctly separated in the original $\mathcal{K}$), and, by **Step 5**, a default-assumption set $\widetilde{\Delta} = \{\delta_0, \delta_1\}$, with $\delta_0 := (B \rightarrow F) \wedge (P \rightarrow \neg F)$ and $\delta_1 := P \rightarrow \neg F$, as in Example 2.

Now, to check if a flying creature presumably is not a penguin (*i.e.*, $F \mathrel{|\!\sim} \neg P$), we take our premise F and our background theory $\widetilde{\mathcal{T}} = \{\neg(P \wedge \neg B)\}$, and we look for the first default δ_i that is consistent with F and $\widetilde{\mathcal{T}}$, *i.e.* $\widetilde{\mathcal{T}} \cup \{F\} \not\models \neg\delta_i$, that is δ_0. Now we have simply to check if $F \wedge \neg(P \wedge \neg B) \wedge (B \rightarrow F) \wedge (P \rightarrow \neg F) \models \neg P$. Since this holds, we have $F \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} \neg P$. Similarly, with such a procedure we can obtain a series of desirable results, as $\neg F \mathrel{|\!\sim} \neg B$, $\neg F \mathrel{|\!\sim} \neg P$, $B \mathrel{|\!\sim} \neg P$, $\neg B \mathrel{|\!\sim} \neg P$, $B \wedge P \mathrel{|\!\sim} \neg F$, $B \wedge green \mathrel{|\!\sim} F$, $P \wedge black \mathrel{|\!\sim} \neg F$. Instead, other counterintuitive connections are not valid, such as $B \wedge \neg F \mathrel{|\!\sim} P$, $B \wedge \neg F \mathrel{|\!\sim} \neg P$, or $P \mathrel{|\!\sim} F$. □

3 Rational Closure in DLs

We consider a significant DL representative, namely $\mathcal{ALC}$ (see e.g. [3], Chap. 2). $\mathcal{ALC}$ corresponds to a fragment of first order logic, using monadic predicates, called *concepts*, and diadic ones, called *roles*. In order to stress the parallel between the procedure presented in Section 2 and the proposal in $\mathcal{ALC}$, we are going to use the same notation for the components playing an analogous role in the two construction: we use $C, D, E, \ldots$ to indicate *concepts*, instead of propositions, and $\models$ and $\mathrel{|\!\sim}$ to indicate, respectively, the 'classical' consequence relation of $\mathcal{ALC}$ and a non-monotonic consequence relation in $\mathcal{ALC}$. δ will indicate a *default concept*, that is, a concept that we assume as applying to every individual, if not informed of the contrary. We have a finite set of *concept names* $\mathcal{C}$, a finite set of *role names* $\mathcal{R}$ and the set $\mathcal{L}$ of $\mathcal{ALC}$ *-concepts* is defined inductively as follows: (i) $\mathcal{C} \subset \mathcal{L}$; (ii) $\top, \bot \in \mathcal{L}$; (iii) $C, D \in \mathcal{L} \Rightarrow C \sqcap D, C \sqcup D, \neg C \in \mathcal{L}$; and (iii) $C \in \mathcal{L}, R \in \mathcal{R} \Rightarrow \exists R.C, \forall R.C \in \mathcal{L}$. Concept $C \rightarrow D$ is used as a shortcut of $\neg C \sqcup D$. The symbols $\sqcap$ and $\sqcup$ correspond, respectively, to the conjunction $\wedge$ and the disjunction $\vee$ of classical logic. Given a set of *individuals* $\mathcal{O}$, an *assertion* is of the form $a{:}C$ ($C \in \mathcal{L}$) or of the form $(a, b){:}R$ ($R \in \mathcal{R}$), respectively indicating that the individual a is an instance of concept C, and that the individuals a and b are connected by the role R. A *general inclusion axiom* (GCI) is of the form $C \sqsubseteq D$ ($C, D \in \mathcal{L}$) and indicates that any instance of C is also an instance of D. We use $C = D$ as a shortcut of the pair of $C \sqsubseteq D$ and $D \sqsubseteq C$.

From a FOL point of view, concepts, roles, assertions and GCIs, may be seen as formulae obtained by the following transformation

$$
\begin{array}{ll}
\tau(a{:}C) & = \tau(a, C) \\
\tau((a, b){:}R) & = R(a, b) \\
\tau(C \sqsubseteq D) & = \forall x.\tau(x, C) \rightarrow \tau(x, D) \\
\tau(x, A) & = A(x) \\
\tau(x, \neg C) & = \neg\tau(x, C)
\end{array}
\qquad
\begin{array}{ll}
\tau(x, C \sqcap D) & = \tau(x, C) \wedge \tau(x, D) \\
\tau(x, C \sqcup D) & = \tau(x, C) \vee \tau(x, D) \\
\tau(x, \exists R.C) & = \exists y.R(x, y) \wedge \tau(y, C) \\
\tau(x, \forall R.C) & = \forall y.R(x, y) \rightarrow \tau(y, C)\,.
\end{array}
$$

Now, a classical knowledge base is defined by a pair $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{T}$ is a finite set of GCIs (a *TBox*) and $\mathcal{A}$ is a finite set of assertions (the *ABox*), whereas a *defeasible knowledge base* is represented by a triple $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{B} \rangle$, where additionally $\mathcal{B}$ is a finite set of sequents of the form $C|\sim D$ ('an instance of a concept C is typically an instance of a concept D'), with $C, D \in \mathcal{L}$.

Example 4. Consider Example 3. Just add a role $Prey$ in the vocabulary, where a role instantiation (a, b):$Prey$ is read as 'a preys for b', and add also two more concepts, I (Insect) and Fi (Fish). A defeasible KB is $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{B} \rangle$ with $\mathcal{A} = \{a{:}P, b{:}B, (a, c){:}Prey, (b, c){:}Prey\}$; $\mathcal{T} = \{P \sqsubseteq B, I \sqsubseteq \neg Fi\}$ and $\mathcal{B} = \{P|\sim\neg F,\ B|\sim F,\ P|\sim\forall Prey.Fi, B|\sim\forall Prey.I\}$. □

The particular structure of a defeasible KB allows for the 'isolation' of the pair $\langle \mathcal{T}, \mathcal{B} \rangle$, that we could call the *conceptual system* of the agent, from the information about the individuals (formalised in $\mathcal{A}$) that will play the role of the facts known to be true. In the next section we are going to work with the information about concepts $\langle \mathcal{T}, \mathcal{B} \rangle$ first, exploiting the immediate analogy with the homonymous pair of Section 2, then we will address the case involving individuals as well.

Construction of the Default-Assumption System. We apply to $\langle \mathcal{T}, \mathcal{B} \rangle$ an analogous transformation (2), in order to obtain from $\langle \mathcal{T}, \mathcal{B} \rangle$ a pair $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$, where $\widetilde{\mathcal{T}}$ is a set of GCIs, representing the background knowledge, and $\widetilde{\Delta}$ is a set of concepts, playing the role of default-assumptions, that is, concepts that, modulo consistency, apply to each individual. Hence, starting with $\langle \mathcal{T}, \mathcal{B} \rangle$, we apply the following steps.

Step 1. Define $\mathcal{B}' = \mathcal{B} \cup \{C \sqcap \neg D|\sim\bot \mid C \sqsubseteq D \in \mathcal{T}\}$. Now our agent is characterised by the pair $\langle \emptyset, \mathcal{B}' \rangle$.

Step 2. Define $\Gamma_{\mathcal{B}'} = \{\top \sqsubseteq C \rightarrow D \mid C|\sim D \in \mathcal{B}'\}$, and define a set $\mathfrak{A}_{\mathcal{B}'}$ as the set of the antecedents of the conditionals in $\mathcal{B}'$, *i.e.* $\mathfrak{A}_{\mathcal{B}'} = \{C \mid C|\sim D \in \mathcal{B}'\}$.

Step 3. We determine the exceptionality ranking of the sequents in $\mathcal{B}'$ using the set of the antecedents $\mathfrak{A}_{\mathcal{B}'}$ and the materializations in $\Gamma_{\mathcal{B}'}$, where a concept C is *exceptional* w.r.t. a set of sequents $\mathcal{D}$ iff $\Gamma_{\mathcal{D}} \models \top \sqsubseteq \neg C$. The steps are the same of the propositional case (**Steps 3.1 – 3.4**), we just replace the expression $\Gamma_{\mathcal{D}} \models \neg C$ with the expression $\Gamma_{\mathcal{D}} \models \top \sqsubseteq \neg C$. In this way we define a ranking function r.

Step 4. From $\Gamma_{\mathcal{B}'}$ and the ranking function r we obtain two kinds of information. First (**Step 4.1.**), we can verify if the conceptual system of the agent is consistent, by checking the consistency of $\Gamma_{\mathcal{B}'}$. Then (**Steps 4.2.-4.3.**), we can define the real background theory and the defeasible information of the agent, respectively the sets $\widetilde{\mathcal{T}}$ and $\widetilde{\mathcal{B}}$ as:

$$\widetilde{\mathcal{T}} = \{\top \sqsubseteq \neg C \mid C|\sim D \in \mathcal{B}' \text{ and } r(C|\sim D) = \infty\}$$
$$\widetilde{\mathcal{B}} = \{C|\sim D \mid C|\sim D \in \mathcal{B}' \text{ and } r(C|\sim D) < \infty\}\ .$$

Step 5. Again, we define the set of our 'default assumptions' by using the materialisation of the sequents in $\widetilde{\mathcal{B}}$ and the ranking function r. That is, $\widetilde{\Delta} = \{\delta_0, \ldots, \delta_n\}$, where

$$\delta_i = \bigsqcap\{C \rightarrow D \mid C|\sim D \in \widetilde{\mathcal{B}} \text{ and } r(C|\sim D) \geq i\}\ .$$

Hence, we obtain an analogous of the default-assumption characterisation defined in the propositional case by substituting the conceptual system $\langle \mathcal{T}, \mathcal{B} \rangle$ with the pair $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$, where $\widetilde{\Delta}$ is a set of concepts, instead of a set of sequents $C \mathrel{|\!\sim} D$. It is not difficult to see that $\widetilde{\Delta}$ presents the same characteristics described at the end of Section 3, that is, for every δ_i, $0 \leq i < n$, $\models \delta_i \sqsubseteq \delta_{i+1}$.

Closure Operation over Concepts. Consider now $\widetilde{\mathcal{T}} = \{\top \sqsubseteq C_1, \ldots, \top \sqsubseteq C_m\}$ and $\widetilde{\Delta} = \{\delta_0, \ldots, \delta_n\}$. We call $\mathfrak{T}$ the set of the concepts in $\widetilde{\mathcal{T}}$, that is, $\mathfrak{T} = \{C_1, \ldots, C_m\}$. Next we define the notion of default-assumption consequence relation between the concepts, that is, a relation $\mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle}$ that tells us what presumably follows from a finite set of concepts. Formally, E is a *default-assumption consequence* of the set of concepts Γ, given a background theory $\mathcal{T}$ and a set of default-assumptions Δ, written $\Gamma \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E$, if and only if E is implied by the union of Γ with $\mathfrak{T}$ and every $(\mathfrak{T} \cup \Gamma)$-maxiconsistent subset of Δ, *i.e.*

$$\Gamma \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E \textit{ iff } \models (\mathfrak{T} \cup \Gamma \cup \Delta') \sqsubseteq E \textit{ for every } (\mathfrak{T} \cup \Gamma)\textit{-maxiconsistent } \Delta' \subseteq \Delta \ .$$

Given that, also in the DL case, every element δ_i of the default set Δ classically implies the subsequent elements (for every i, $0 \leq i < n$, $\models \delta_i \sqsubseteq \delta_{i+1}$), we obtain, in exactly the same way as in the propositional case, the analogous of Proposition 2 that every $\mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle}$-sequent is determined by a single element of Δ. That is:

Proposition 3. *$\Gamma \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} D$ iff $\models \bigsqcap \Gamma \sqcap \bigsqcap \mathfrak{T} \sqcap \delta_i \sqsubseteq D$, where δ_i is the first $(\Gamma \cup \mathfrak{T})$-consistent formula[5] of the sequence $\langle \delta_0, \ldots, \delta_n \rangle$.*

Hence, as in the propositional case, we have an unique default-assumption extension at the level of concepts. From now on, talking about a default set Δ, we assume that it is a linearly ordered set $\Delta = \{\delta_0, \ldots, \delta_n\}$ s.t. for every i, $0 \leq i < n$, $\models \delta_i \sqsubseteq \delta_{i+1}$.

Now, the main point is: if $\mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle}$ has been generated from $\mathcal{K} = \langle \mathcal{T}, \mathcal{B} \rangle$, then $\mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle}$ is a rational consequence relation validating $\mathcal{K}$ (*i.e.*, if $C \sqsubseteq E \in \mathcal{T}$, then $C \sqcap \neg E \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} \bot$, and if $C \mathrel{|\!\sim} E \in \mathcal{B}$, then $C \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} E$).

Proposition 4. *$\mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle}$ is a rational consequence relation validating $\mathcal{K} = \langle \mathcal{T}, \mathcal{B} \rangle$.*

This can be shown by noting that the analogous properties of the propositional rational consequence relation are satisfied, namely:

$$\text{(REF)}\ \ C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} C$$

$$\text{(LLE)}\ \frac{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E \qquad \models C = D}{D \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E} \qquad \text{(RW)}\ \frac{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D \qquad \models D \sqsubseteq E}{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E}$$

$$\text{(CT)}\ \frac{C \sqcap D \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E \qquad C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D}{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E} \qquad \text{(CM)}\ \frac{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E \qquad C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D}{C \sqcap D \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E}$$

$$\text{(OR)}\ \frac{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E \qquad D \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E}{C \sqcup D \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} E} \qquad \text{(RM)}\ \frac{C \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D \qquad C \not\mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} \neg E}{C \sqcap E \mathrel{|\!\sim}_{\langle \mathcal{T}, \Delta \rangle} D}$$

[5] That is, $\not\models \bigsqcap \mathfrak{T} \sqcap \bigsqcap \Gamma \sqsubseteq \neg \delta_i$.

Let us work out the analogue of Example 3 in the DL context.

Example 5. Consider the KB of Example 4 without the ABox. Hence, we start with $\mathcal{K} = \langle \mathcal{T}, \mathcal{B} \rangle$. Then $\mathcal{K}$ is changed into $\mathcal{B}' = \{P \sqcap \neg B \mathrel{|\!\sim} \bot, I \sqcap Fi \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F, B \mathrel{|\!\sim} F, P \mathrel{|\!\sim} \forall Prey.Fi, B \mathrel{|\!\sim} \forall Prey.I\}$. The set of the materializations of $\mathcal{B}'$ is $I_{\mathcal{B}'} = \{\top \sqsubseteq P \wedge \neg B \rightarrow \bot, \top \sqsubseteq I \sqcap Fi \rightarrow \bot, \top \sqsubseteq P \rightarrow \neg F, \top \sqsubseteq B \rightarrow F, \top \sqsubseteq P \rightarrow \forall Prey.Fi, \top \sqsubseteq B \rightarrow \forall Prey.I\}$, with $\mathfrak{A}_{\mathcal{B}'} = \{P \wedge \neg B, I \sqcap Fi, P, B\}$. Following the procedure at **Step 3**, we obtain the exceptionality ranking of the sequents: $\mathcal{E}_0 = \{P \sqcap \neg B \mathrel{|\!\sim} \bot, I \sqcap Fi \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F, B \mathrel{|\!\sim} F, P \mathrel{|\!\sim} \forall Prey.Fi, B \mathrel{|\!\sim} \forall Prey.I\}$; $\mathcal{E}_1 = \{P \sqcap \neg B \mathrel{|\!\sim} \bot, I \sqcap Fi \mathrel{|\!\sim} \bot, P \mathrel{|\!\sim} \neg F, P \mathrel{|\!\sim} \forall Prey.Fi\}$; $\mathcal{E}_2 = \{P \sqcap \neg B \mathrel{|\!\sim} \bot, I \sqcap Fi \mathrel{|\!\sim} \bot\}$ and $\mathcal{E}_3 = \{P \sqcap \neg B \mathrel{|\!\sim} \bot, I \sqcap Fi \mathrel{|\!\sim} \bot\}$. Automatically, we have the ranking values of every sequent in $\mathcal{B}'$: namely, $r(B \mathrel{|\!\sim} F) = r(B \mathrel{|\!\sim} \forall Prey.I) = 0$; $r(P \mathrel{|\!\sim} \neg F) = r(P \mathrel{|\!\sim} \forall Prey.Fi) = 1$ and $r(P \sqcap \neg B \mathrel{|\!\sim} \bot) = r(I \sqcap Fi \mathrel{|\!\sim} \bot) = \infty$. From such a ranking, we obtain a background theory $\widetilde{\mathcal{T}} = \{\top \sqsubseteq \neg(P \wedge \neg B), \top \sqsubseteq \neg(I \sqcap Fi)\}$, and a default-assumption set $\widetilde{\Delta} = \{\delta_0, \delta_1\}$, with

$$\delta_0 = (B \rightarrow F) \sqcap (B \rightarrow \forall Prey.I) \sqcap (P \rightarrow \neg F) \sqcap (P \rightarrow \forall Prey.Fi)$$
$$\delta_1 = (P \rightarrow \neg F) \sqcap (P \rightarrow \forall Prey.Fi)\ .$$

Now by using Proposition 3, we obtain the analogue sequents as in the propositional case, and avoid the same undesirable ones. Moreover we can derive also sequents connected to the roles, such as $B \mathrel{|\!\sim} \forall Prey.\neg Fi$ and $P \mathrel{|\!\sim} \forall Prey.\neg I$. □

We conclude by noting that from a computational complexity point of view, as deciding entailment in $\mathcal{ALC}$ is EXPTIME-complete [16] [6], we obtain immediately that

Corollary 2. *Deciding $C \mathrel{|\!\sim}_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} D$ in $\mathcal{ALC}$ is an EXPTIME-complete problem.*

More generally, defeasible consequence under rational closure inherits the computational complexity of entailment of the underlying DL language and, thus, *e.g.* is polynomial for the DL $\mathcal{EL}$ [2].

Closure Operation over Individuals. So far, we left out the ABox that we will consider next. Given $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \Delta \rangle$, we would like to infer whether a certain individual a is presumably an instance of a concept C or not. The basic idea remains to associate to every individual every default-assumption information that is consistent with our knowledge base. As we will see, the major problem to be addressed here is to guarantee the uniqueness of the default-assumption extension.

Example 6. Consider $\mathcal{K} = \langle \mathcal{A}, \emptyset, \Delta \rangle$, with $\mathcal{A} = \{(a, b){:}R\}$ and $\Delta = \{A \sqcap \forall R.\neg A\}$. Informally, if we apply the default to a first, we get $b{:}\neg A$ and we cannot apply the default to b, while if we apply the default to b first, we get $b{:}A$ and we cannot apply the default to a. Hence, we may have *two* extensions. □

The possibility of multiple extensions is due to the presence of the roles, that allow the transmission of information from an individual to another; if every individual was 'isolated', without role-connections, then the addition of the default information to each

[6] Recall that for any deterministic complexity class $\mathcal{C}$, $\mathcal{C} = co\mathcal{C}$, so, *e.g.* $EXPTIME = coEXPTIME$.

individual would have been a 'local' problem, treatable without considering the concepts associated to the other individuals in the dominion, and the default-assumption extension would have been unique. On the other hand, while considering a specific individual, the presence of the roles forces to consider also the information associated to other individuals in order to maintain the consistency of the knowledge base, and, as show in example 6, the addition of default information to one individual could prevent the association of default information to another.

Now, first of all, we will assume that $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \Delta \rangle$ has already been transformed into $\langle \mathcal{A}, \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$, where $\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle$ have been computed as in the previous section and, thus, the defaults in $\widetilde{\Delta} = \{\delta_0, \ldots, \delta_n\}$ are ordered ($0 \leq i < n$, $\models \delta_i \sqsubseteq \delta_{i+1}$). We also assume that $\langle \mathcal{A}, \mathcal{T} \rangle$ is consistent, *i.e.* $\langle \mathcal{A}, \mathcal{T} \rangle \not\models a{:}\bot$, for any a. For the sake of this paper, we will assume that $\mathcal{T}$ is *unfoldable*, that is defined as follows: (i) $\mathcal{T}$ contains axioms of the form $A \sqsubseteq C$ or $A = C$, where A is a concept name and C a concept; (ii) for any concept name A, there is at most one axiom having A on the left-hand side; (iii) $\mathcal{T}$ is *acyclic*, *i.e.* there is no concept name A that depends on A [7]. Besides having a high practical interest, unfoldable TBoxes have the characteristics that they can be removed in the following way: given $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \Delta \rangle$, (i) replace any inclusion axiom $A \sqsubseteq C \in \mathcal{T}$ with $A = C \sqcap A'$, where A' is a new concept name; (ii) in $\mathcal{A}$ and Δ, replace recursively any occurrence of concept names with their definition in $\mathcal{T}$; and (iii) remove $\mathcal{T}$ from $\mathcal{K}$. Hence, we may assume that $\mathcal{K}$ is of the form $\mathcal{K} = \langle \mathcal{A}, \Delta \rangle$. We may also assume that any concept in $\mathcal{A}$ is in *Negation Normal Form*, that is, a negation may occur in front of a concept name only (this is achieved in the usual way by removing double negations and pushing negation inwards [8]). Without loss of generality, we will further assume that $\mathcal{A}$ is closed under the following 'completion' rules: (i) if $a{:}C \sqcap D \in \mathcal{A}$ then both $a{:}C$ and $a{:}D$ are in $\mathcal{A}$; (ii) if $a{:}\exists R.C \in \mathcal{A}$ then there are $(a, b){:}R$ and $b{:}C$ in $\mathcal{A}$; and (iii) if $a{:}\forall R.C$ and $(a, b){:}R$ are in $\mathcal{A}$ then so is $b{:}C$. In this way, $\mathcal{A}$ contains all the information that is shared among all models of $\mathcal{A}$. Now, with $\mathcal{O}_\mathcal{A}$ we indicate the individuals occurring in $\mathcal{A}$. Given $\mathcal{K} = \langle \mathcal{A}, \Delta \rangle$ (recall that $\Delta = \{\delta_0, \ldots, \delta_n\}$), we say that a knowledge base $\widetilde{\mathcal{K}} = \langle \mathcal{A}_\Delta \rangle$ is a *default-assumption extension* of $\mathcal{K}$ iff

- $\widetilde{\mathcal{K}}$ is classically consistent and $\mathcal{A} \subseteq \mathcal{A}_\Delta$.
- For any $a \in \mathcal{O}_\mathcal{A}$, $a{:}C \in \mathcal{A}_\Delta \setminus \mathcal{A}$ iff $C = \delta_i$ for some i and for every δ_h, $h < i$, $\mathcal{A}_\Delta \cup \{a{:}\delta_h\} \models \bot$.
- There is no $\mathcal{K}' \supset \widetilde{\mathcal{K}}$ satisfying these conditions.

Essentially, we assign to any individual $a \in \mathcal{O}_\mathcal{A}$, the strongest default to it.

Example 7. Referring to Example 6, consider $\mathcal{K} = \langle \mathcal{A}, \Delta \rangle$, with $\mathcal{A} = \{(a, b) : R\}$ and $\Delta = \{A \sqcap \forall R.\neg A, \top\}$. Then we have two default-assumption extensions, namely $\widetilde{\mathcal{K}}_1 = \mathcal{A} \cup \{a{:}A, a{:}\forall R.\neg A, b{:}\top\}$ and $\widetilde{\mathcal{K}}_2 = \mathcal{A} \cup \{b{:}A, b{:}\forall R.\neg A, a{:}\top\}$. □

A simple procedure to obtain extensions is as follows:

[7] *A depends directly on B* iff there is an axiom in $\mathcal{T}$ having A in the left-hand side and B in the right-hand side. The relation *depends on* is defined as the transitive closure of the relation *depends directly on*.

[8] Note that $\neg\forall R.C$ is the same as $\exists R.\neg C$.

1. fix a linear order $s = \langle a_1, \ldots, a_m \rangle$ of the individuals in $\mathcal{O}_\mathcal{A}$;
2. for any individual a_j processed in this order, consider the first default δ_i such that $\mathcal{A} \cup \{a_j{:}\delta_i\}$ is consistent;
3. update $\mathcal{A}$ by adding $a{:}\delta_i$ to it and process the next individual.

It can be shown that

Proposition 5. *Given a linear order of the individuals in $\mathcal{K}$, the above procedure determines a default-assumption extension of $\mathcal{K}$. Vice-versa, every default-assumption extension of $\mathcal{K}$ corresponds to the knowledge base generated by some linear order of the individuals in $\mathcal{K}$.*

For instance, related to Example 7, $\widetilde{\mathcal{K}}_1$ is obtained from the order $\langle a, b \rangle$, while $\widetilde{\mathcal{K}}_2$ is obtained from the order $\langle b, a \rangle$.

Example 8. Refer to Example 4 and 5, and let $\mathcal{K} = \{\mathcal{A}, \mathcal{T}, \Delta\}$, where $\mathcal{A} = \{a{:}P, b{:}B, (a,c){:}Prey, (b,c){:}Prey\}$, $\mathcal{T} = \{P = B \sqcap B', I = \neg Fi \sqcap I'\}$, $\Delta = \{\delta_0, \delta_1\}$, $\delta_0 = (B \rightarrow F) \sqcap (B \rightarrow \forall Prey.I) \sqcap (P \rightarrow \neg F) \sqcap (P \rightarrow \forall Prey.Fi)$ and $\delta_1 = (P \rightarrow \neg F) \sqcap (P \rightarrow \forall Prey.Fi)$. After expanding the TBox and 'applying' the completion rules to $\mathcal{A}$, we get $\mathcal{K} = \{\mathcal{A}, \Delta\}$, where $\mathcal{A} = \{a{:}B \sqcap B', a{:}B, a{:}B', b{:}B, (a,c){:}Prey, (b,d){:}Prey\}$, $\Delta = \{\delta_0, \delta_1\}$, $\delta_0 = (B \rightarrow F) \sqcap (B \rightarrow \forall Prey.(\neg Fi \sqcap I')) \sqcap ((B \sqcap B') \rightarrow \neg F) \sqcap ((B \sqcap B') \rightarrow \forall Prey.Fi)$ and $\delta_1 = ((B \sqcap B') \rightarrow \neg F) \sqcap ((B \sqcap B') \rightarrow \forall Prey.Fi)$. If we consider an order where a is considered before b then we associate δ_1 to a, and consequently c is presumed to be a fish and we are prevented in the association of δ_0 to b. If we consider b before a, c is not a fish and we cannot apply δ_1 to a. □

Now, if we fix a priori a linear order s on the individuals, we may define a consequence relation depending on the default-assumption extension generated from it: we say that $a{:}C$ is a *defeasible consequence* of $\mathcal{K}$, written $\mathcal{K} \mid\!\sim_s a{:}C$, iff $\widetilde{\mathcal{K}} \models a{:}C$, where $\widetilde{\mathcal{K}}$ is the default-assumption extension generated from $\mathcal{K}$ based on the order s.

For instance, related to Example 7 and order $s_1 = \langle a, b \rangle$, we may infer that $\mathcal{K} \mid\!\sim_{s_1} a{:}A$, while with order $s_2 = \langle b, a \rangle$, we may infer that $\mathcal{K} \mid\!\sim_{s_2} b{:}A$.

The interesting point of such a consequence relation is that it satisfies the properties of a *rational* consequence relation in the following way.

$$REF_{DL} \quad \langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}C \text{ for every } a{:}C \in \mathcal{A}$$

$$LLE_{DL} \quad \frac{\langle \mathcal{A} \cup \{b{:}D\}, \Delta \rangle \mid\!\sim_s a{:}C \quad \models D = E}{\langle \mathcal{A} \cup \{b{:}E\}, \Delta \rangle \mid\!\sim_s a{:}C}$$

$$RW_{DL} \quad \frac{\langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}C \quad \models C \sqsubseteq D}{\langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}D}$$

$$CT_{DL} \quad \frac{\langle \mathcal{A} \cup \{b{:}D\}, \Delta \rangle \mid\!\sim_s a{:}C \quad \langle \mathcal{A}, \Delta \rangle \mid\!\sim_s b{:}D}{\langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}C}$$

$$CM_{DL} \quad \frac{\langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}C \quad \langle \mathcal{A}, \Delta \rangle \mid\!\sim_s b{:}D}{\langle \mathcal{A} \cup \{b{:}D\}, \Delta \rangle \mid\!\sim_s a{:}C}$$

$$OR_{DL} \quad \frac{\langle \mathcal{A} \cup \{b{:}D\}, \Delta \rangle \mid\!\sim_s a{:}C \quad \langle \mathcal{A} \cup \{b{:}E\}, \Delta \rangle \mid\!\sim_s a{:}C}{\langle \mathcal{A} \cup \{b{:}D \sqcup E\}, \Delta \rangle \mid\!\sim_s a{:}C}$$

$$RM_{DL} \quad \frac{\langle \mathcal{A}, \Delta \rangle \mid\!\sim_s a{:}C \quad \langle \mathcal{A}, \Delta \rangle \not\mid\!\sim_s b{:}\neg D}{\langle \mathcal{A} \cup \{b{:}D\}, \Delta \rangle \mid\!\sim_s a{:}C}$$

We can show that

Proposition 6. *Given $\mathcal{K}$ and a linear order s of the individuals in $\mathcal{K}$, the consequence relation $\mid\!\sim_s$ satisfies the properties* $\mathrm{REF_{DL}} - \mathrm{RM_{DL}}$.

Note that from a computational complexity point of view, as entailment w.r.t. a $\mathcal{ALC}$ ABox is PSPACE-complete, we get immediately

Proposition 7. *Deciding $\mathcal{K} \Vdash_s a{:}C$ in $\mathcal{ALC}$ with unfoldable TBox is a PSPACE-complete problem.*

We conclude by illustrating a case with a unique extension.

Example 9. Consider the KB in Example 8, where $(b, c){:}Prey$ is replaced with $(b, d){:}Prey$. Then, whatever is the order on the individuals, we obtain the following association between the default formulae and the individuals: $a{:}\delta_1$, $b{:}\delta_0$, $c{:}\delta_0$, and $d{:}\delta_0$. Using the information in these defaults, we obtain a unique default-assumption extension. □

The above example suggest yet another consequence relation, namely based on the intersection of every default-assumption extension: given $\mathcal{K}$, we say that $a{:}C$ is a *strict defeasible consequence* of $\mathcal{K}$, written $\mathcal{K} \Vdash a{:}C$, iff $\mathcal{K} \Vdash_s a{:}C$ for any linear order s of the individuals in $\mathcal{K}$. Note that $\Vdash$ may not necessarily satisfy the properties of a rational consequence relation. However, if the default-assumption extension is unique then $\Vdash$ satisfies the properties $\mathrm{REF}_{\mathrm{DL}} - \mathrm{RM}_{\mathrm{DL}}$.

4 Conclusions

We have presented a non-monotonic extension for the DL $\mathcal{ALC}$, focussing on the migration of the properties of a rational closure consequence relation at the propositional level towards the DL level. We provided first an algorithmic propositional definition that we then adapted to the DL case. In particular, we have defined a consequence relation $C \mid\!\sim_{\langle \widetilde{\mathcal{T}}, \widetilde{\Delta} \rangle} D$ among concepts and have shown that it is a rational consequence relation. We then defined a consequence relation $\mathcal{K} \Vdash_s a{:}C$ among an unfoldable KB and assertions that, under a given linear order s of the individuals in $\mathcal{K}$, is a rational consequence relation as well. Note that here s denotes a priority on the individuals on which to focus the attention, which is different from approaches such as [11, 23] in which the order indicates that an individual is more typical than another one. Interestingly, both consequence relations we have defined additionally inherit the same computational complexity of the underlying DL language.

Besides trying to extend our method to more expressive DL languages, we conjecture the validity, as in the propositional case (see [28, 19]), of a representation result connecting rational consequence relations, default assumptions consequence relations (with Δ linearly ordered by $\models$ as above) and semantical models with a modular (*i.e.* reflexive, transitive and complete) typicality relation defined over the individuals.

References

1. Baader, F., Hollunder, B.: How to prefer more specific defaults in terminological default logic. In: Proc. of IJCAI, pp. 669–674. Morgan Kaufmann, San Francisco (1993)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI, pp. 364–369. Morgan Kaufmann, San Francisco (2005)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)

4. Baader, F., Hollunder, B.: Embedding defaults into terminological representation systems. J. Automated Reasoning 14, 149–180 (1995)
5. Baader, F., Hollunder, B.: Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. J. Automated Reasoning 15, 41–68 (1995)
6. Bochman, A.: A logical theory of nonmonotonic inference and belief change. Springer, Heidelberg (2001)
7. Bonatti, P.A., Faella, M., Sauro, L.: Defeasible inclusions in low-complexity DLs: Preliminary notes. In: Proc. of IJCAI, pp. 696–701. Morgan Kaufmann, San Francisco (2009)
8. Bonatti, P.A., Lutz, C., Wolter, F.: Description logics with circumscription. In: Proc. of KR, pp. 400–410. AAAI Press, Menlo Park (2006)
9. Bonatti, P.A., Lutz, C., Wolter, F.: The complexity of circumscription in description logic. J. Artif. Int. Res. 35(1), 717–773 (2009)
10. Brewka, G.: The logic of inheritance in frame systems. In: Proc. of IJCAI, pp. 483–488 (1987)
11. Britz, K., Heidema, J., Meyer, T.: Semantic preferential subsumption. In: Proc. of KR, pp. 476–484. Morgan Kaufmann, San Francisco (2008)
12. Britz, K., Heidema, J., Meyer, T.: Modelling object typicality in description logics. In: Proc. of the Australasian Joint Conf. on Advances in Artificial Intelligence, pp. 506–516. Springer, Heidelberg (2009)
13. Cadoli, M., Donini, F.M., Schaerf, M.: Closed world reasoning in hybrid systems. In: Proc. of ISMIS, pp. 474–481. North-Holland Publ. Co., Amsterdam (1990)
14. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: Adding epistemic operators to concept languages. In: Proc. of KR, pp. 342–353. Morgan Kaufmann, San Francisco (1992)
15. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. Artificial Intelligence 100(1-2), 225–274 (1998)
16. Donini, F.M., Massacci, F.: Exptime tableaux for $\mathcal{ALC}$. Artificial Intelligence 124(1), 87–138 (2000)
17. Donini, F.M., Nardi, D., Rosati, R.: Autoepistemic description logics. In: Proc. of IJCAI, pp. 136–141 (1997)
18. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic 3(2), 177–225 (2002)
19. Freund, M.: Preferential reasoning in the perspective of Poole default logic. Artif. Intell. 98(1-2), 209–235 (1998)
20. Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.): Handbook of logic in artificial intelligence and logic programming. Nonmonotonic reasoning and uncertain reasoning, vol. 3. Oxford University Press, Oxford (1994)
21. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.: Preferential description logics. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 257–272. Springer, Heidelberg (2007)
22. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.: Reasoning about typicality in preferential description logics. In: Hölldobler, S., Lutz, C., Wansing, H. (eds.) JELIA 2008. LNCS (LNAI), vol. 5293, pp. 192–205. Springer, Heidelberg (2008)
23. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.: On extending description logics for reasoning about typicality: a first step. Technical Report 116/09, Università degli studi di Torino (December 2009)
24. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.: Prototypical reasoning with low complexity description logics: Preliminary results. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 430–436. Springer, Heidelberg (2009)
25. Grimm, S., Hitzler, P.: A preferential tableaux calculus for circumscriptive $\mathcal{ALCO}$. In: Proc. of RR, pp. 40–54. Springer, Heidelberg (2009)

26. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artif. Intell. 44(1-2), 167–207 (1990)
27. Lambrix, P., Shahmehri, N., Wahlloef, N.: A default extension to description logics for use in an intelligent search engine. In: Proc. of HICSS, vol. 5, p. 28. IEEE Computer Society, Los Alamitos (1998)
28. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artif. Intell. 55(1), 1–60 (1992)
29. Makinson, D.: General patterns in nonmonotonic reasoning. In: Handbook of logic in artificial intelligence and logic programming: Nonmonotonic reasoning and uncertain reasoning, vol. 3, pp. 35–110. Oxford University Press, Oxford (1994)
30. Makinson, D.: Bridges from Classical to Nonmonotonic Logic. King's College Publications, London (2005)
31. Padgham, L., Nebel, B.: Combining classification and non-monotonic inheritance reasoning: A first step. In: Komorowski, J., Raś, Z.W. (eds.) ISMIS 1993. LNCS, vol. 689. Springer, Heidelberg (1993)
32. Padgham, L., Zhang, T.: A terminological logic with defaults: A definition and an application. In: Proc. of IJCAI, pp. 662–668. Morgan Kaufmann, San Francisco (1993)
33. Poole, D.: A logical framework for default reasoning. Artif. Intell. 36(1), 27–47 (1988)
34. Quantz, J., Royer, V.: A preference semantics for defaults in terminological logics. In: Proc. of KR, pp. 294–305. Morgan Kaufmann, San Francisco (1992)
35. Rector, A.L.: Defaults, context, and knowledge: Alternatives for owl-indexed knowledge bases. In: Pacific Symposium on Biocomputing, pp. 226–237. World Scientific, Singapore (2004)
36. Straccia, U.: Default inheritance reasoning in hybrid KL-ONE-style logics. In: Proc. of IJCAI, pp. 676–681. Morgan Kaufmann, San Francisco (1993)

Extensional Higher-Order Logic Programming*

Angelos Charalambidis[1], Konstantinos Handjopoulos[1], Panos Rondogiannis[1], and William W. Wadge[2]

[1] Department of Informatics & Telecommunications, University of Athens, Greece
[2] Department of Computer Science, University of Victoria, Canada

Abstract. We propose a purely extensional semantics for higher-order logic programming. Under this semantics, every program has a unique minimum Herbrand model which is the greatest lower bound of all Herbrand models of the program and the least fixed-point of the immediate consequence operator of the program. We also propose an SLD-resolution proof procedure which is sound and complete with respect to the minimum model semantics. In other words, we provide a purely extensional theoretical framework for higher-order logic programming which generalizes the familiar theory of classical (first-order) logic programming.

1 Introduction

The extension of logic programming to support higher-order constructs (in a semantically clean way) is an intriguing research problem. The initial attitude of logic programmers towards this problem was somewhat skeptical: it was argued (see for example [War82]) that higher-order extensions may not be that necessary since there exist ways of simulating higher-order programming inside Prolog itself. Later on, more genuine approaches were developed. The main such examples are λ-Prolog [NM98] and Hilog [CKW93]. These two systems share a common idea, namely they are both *intensional*: two predicates are not considered equal unless their names are the same. The intensional approach has its merits, and this is evidenced by the fact that both of the above systems continued to develop and to explore various application domains.

There have also been a few attempts to define extensional higher-order logic programming systems. In such a system, two predicates are considered equal if they have the same extensions, namely they are true for the same arguments. The first extensional higher-order approach was proposed in [Wad91]. Subsequently, M. Bezem [Bez99] also considered an alternative extensional approach which however appears to differ from classical extensionality and has a more proof-theoretical flavor. In [Wad91] it is suggested that by restricting the syntax of higher-order programs, we can get a language in which the defined relations are *continuous*. Continuity guarantees that every such program has a well-defined meaning which can be computed with standard domain-theoretic techniques. There are two main issues that remained unresolved in [Wad91]:

* This work has been partially supported by the University of Athens under the project "Kapodistrias" (grant no. 70/4/5827).

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 91–103, 2010.
© Springer-Verlag Berlin Heidelberg 2010

- The higher-order fragment considered in [Wad91], does not allow uninstantiated higher-order variables to appear in clause bodies or in queries.
- A proof procedure is not provided (but it is conjectured that a sound and complete such procedure exists that even covers the extension with uninstantiated higher-order variables).

In this paper we remedy the above issues and provide the first (to our knowledge) framework for extensional higher-order logic programming that is complete both from a semantic as-well-as from a proof theoretic point of view. To demonstrate the key idea of our approach, consider the following higher-order program:

```
ordered(R,[]).
ordered(R,[X]).
ordered(R,[X,Y|T]):-R(X,Y),ordered(R,[Y|T]).
```

Consider the query $\leftarrow$ `ordered(R,[a,b,c,d])`. At first sight this appears to be an unreasonable query, since the set of possible solutions is infinite (and actually uncountable). However, at a closer look we realize that all such relations extend the simple relation $r = \{(\mathtt{a},\mathtt{b}),(\mathtt{b},\mathtt{c}),(\mathtt{c},\mathtt{d})\}$. In other words, an implementation that would return the answer `R={(a,b),(b,c),(c,d)}` would be satisfactory in the sense that `R` can be taken to represent all its superset relations (whose additional information is of no interest to the programmer). In the fragment that we consider, if a relation satisfies a predicate then there exists a simple (or *basic*) relation (like r in our example) that also satisfies the predicate. The set of such simple relations is countable and therefore a proof procedure can be devised that is not only sound but also complete. However, we need to formally characterize which are these basic relations for every possible type.

Fortunately, there exists a branch of domain theory that examines exactly the above issues. The key notion that we need is that of an *ω-algebraic complete lattice* (a special case of an *ω-algebraic domain*, see for example [AJ94]), namely a complete lattice which has an enumerable basis of *compact elements* (which correspond to the simple elements that we have been talking about). Every element of an ω-algebraic complete lattice can be represented as the least upper bound of such compact elements. In our example, all the relations (even the infinite ones) for which the `ordered` predicate could be true of, can be represented by some simple relations (like r above).

The main task of the paper is therefore to develop a semantics for higher-order logic programming that is based on ω-algebraic complete lattices. Our semantics refines and extends the work of [Wad91], and has an extra important advantage: it leads to a relatively simple sound and complete proof procedure for higher-order logic programming.

2 Algebraic Complete Lattices

In the rest of the paper we assume a basic familiarity with the basic notions regarding partially ordered sets and in particular complete lattices (see for example [Llo87]). Given a partially ordered set (poset) P, we write $\sqsubseteq_P$ (or simply $\sqsubseteq$) for the corresponding partial order.

We will be interested in a certain type of complete lattices in which every element can be "created" by using a set of "basic" elements of the lattice:

Definition 1. *Let L be a complete lattice. An element $c \in L$ is called* compact *if for every directed set $A \subseteq L$ with $c \sqsubseteq \bigsqcup A$, there exists $a \in A$ such that $c \sqsubseteq a$. The set of all compact elements of L is denoted by $\mathcal{K}(L)$.*

Let P be a poset. Given $B \subseteq P$ and $x \in P$, we write $B_{[x]} = \{b \in B \mid b \sqsubseteq_P x\}$.

Definition 2. *A complete lattice L is called an* algebraic complete lattice *if for every $x \in L$, the set $\mathcal{K}(L)_{[x]}$ is a directed subset of L with least upper bound x. The set $\mathcal{K}(L)$ is called the* basis of L. *If additionally, $\mathcal{K}(L)$ is countable, then L is called an* ω-algebraic complete lattice.

We can now introduce the notion of "step functions" which form a subset of monotonic functions with interesting properties:

Definition 3. *Let A be a poset and L be an algebraic complete lattice. Let also $\bot_L$ be the least element of L. For each $a \in A$ and $c \in \mathcal{K}(L)$, we define the* step function $(a \searrow c) : A \to L$ *as*

$$(a \searrow c)(x) = \begin{cases} c, & \text{if } a \sqsubseteq_A x \\ \bot_L, & \text{otherwise} \end{cases}$$

Given posets P, Q, we write $[P \xrightarrow{m} Q]$ to denote the set of all monotonic functions from P to Q. The following lemma, which will prove useful later on, can be easily established:

Lemma 1. *Let A be a poset and L an algebraic complete lattice. Then, $[A \xrightarrow{m} L]$ is an algebraic complete lattice whose basis is the set of all least upper bounds of finitely many step functions from A to L. If A is countable and L is an ω-algebraic complete lattice then $[A \xrightarrow{m} L]$ is an ω-algebraic complete lattice.*

3 The Higher-Order Language $\mathcal{H}$: Syntax

In this section we define the higher-order language $\mathcal{H}$, which will be the basis of the higher-order logic programming language that we will subsequently develop. The language $\mathcal{H}$ is based on a simple type system that supports two base types: o, the boolean domain, and ι, the domain of individuals (data objects). The composite types are partitioned into three classes: functional (assigned to function symbols), argument (assigned to parameters of predicates) and predicate (assigned to predicate symbols).

Definition 4. *A type τ can either be* functional, argument, *or* predicate*:*

$$\begin{aligned} \sigma &:= \iota \mid (\iota \to \sigma) \\ \rho &:= \iota \mid \pi \\ \pi &:= o \mid (\rho \to \pi) \end{aligned}$$

The binary operator $\to$ is right-associative. A functional type that is different than ι will often be written in the form $\iota^n \to \iota$, $n \geq 1$, and a predicate type that is different than o will be written in the form $\rho_1 \to \cdots \to \rho_n \to o$, $n \geq 1$.

Definition 5. *The* alphabet *of the higher-order language $\mathcal{H}$ consists of the following:*

1. *Predicate variables of every predicate type π (such as $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$).*
2. *Argument variables of every argument type ρ (such as $\mathsf{Q}, \mathsf{R}, \mathsf{V}, \mathsf{X}, \ldots$).*
3. *Individual constant symbols of type ι (such as $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$).*
4. *Function symbols of every functional type $\sigma \neq \iota$ (such as $\mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$).*
5. *The following logical constant symbols: the propositional constants $\mathsf{0}$ and $\mathsf{1}$ of type o; the equality constant $\approx$; the generalized disjunction and conjunction constants $\bigvee_\pi$ and $\bigwedge_\pi$ for every predicate type π; the generalized inverse implication constants $\leftarrow_\pi$ for every predicate type π.*
6. *The quantifier $\exists$.*
7. *The parentheses "(" and ")".*

Based on the above alphabet, the expressions of $\mathcal{H}$ can be composed as follows:

Definition 6. *The set of expressions of the higher-order language $\mathcal{H}$ is recursively defined as follows:*

1. *Every predicate variable of type π is an expression of type π; every argument variable of type ρ is an expression of type ρ; every individual constant symbol of type ι is an expression of type ι.*
2. *If f is an n-ary function symbol and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are expressions of type ι, then $(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$ is an expression of type ι.*
3. *If E_1 is an expression of type $\rho \to \pi$ and E_2 is an expression of type ρ, then $(\mathsf{E}_1 \mathsf{E}_2)$ is an expression of type π.*
4. *If X is an argument variable of type ρ and E is an expression of predicate type π, then $(\lambda \mathsf{X}.\mathsf{E})$ is an expression of type $\rho \to \pi$.*
5. *If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of predicate type π, then $(\mathsf{E}_1 \leftarrow_\pi \mathsf{E}_2)$ is an expression of type o, and $(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)$ and $(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)$ are expressions of type π.*
6. *If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of type ι, then $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is an expression of type o.*
7. *If E is an expression of type o and Q is an argument variable (of any argument type), then $(\exists \mathsf{Q}\, \mathsf{E})$ is an expression of type o.*

To denote that an expression E has type τ, we will often write $\mathsf{E} : \tau$; additionally, we write $type(\mathsf{E})$ to denote the type of expression E. Expressions of type ι will be called *terms* and of type o will be called *formulas*. We will write $\leftarrow$, $\wedge$ and $\vee$ instead of $\leftarrow_o$, $\bigwedge_o$ and $\bigvee_o$. When writing an expression, the usual precedence rules will be used to avoid the excessive use of parentheses.

The notions of *free* and *bound* variables of an expression are defined as usual. An expression is called *closed* if it does not contain any free variables. Given an expression E, we denote by $FV(\mathsf{E})$ the set of all free variables of E. By overloading notation, we will also write $FV(S)$, where S is a set of expressions.

In order to build a programming language based on $\mathcal{H}$, a certain syntactic subset of $\mathcal{H}$ must be considered:

Definition 7. *A* positive expression *of $\mathcal{H}$ is one that does not contain $\leftarrow_\pi$.*

Definition 8. *A* program clause *of $\mathcal{H}$ is an expression of the form $\mathsf{p} \leftarrow_\pi \mathsf{E}$, where p is a predicate variable called the* head *of the clause, and E is a closed positive expression of $\mathcal{H}$. A* program *for $\mathcal{H}$ is a set of clauses.*

Definition 9. *A* goal clause *of $\mathcal{H}$ is an expression of the form $\leftarrow \mathsf{E}$ where E is a positive expression of type o. The empty clause is denoted by $\Box$.*

Notice that goal clauses may contain free argument variables (apart from the argument variables that are existentially quantified). Operationally speaking, the free argument variables that appear in a goal are the ones for which an answer is sought for by the proof procedure.

Definition 10. *A* Horn clause *of $\mathcal{H}$ is either a program clause or a goal clause.*

Example 1. The following is a higher order program that computes the closure of its input binary relation $\mathtt{R}$. The type of $\mathtt{closure}$ is $\pi = (\iota \to \iota \to o) \to \iota \to \iota \to o$.

$$\mathtt{closure} \leftarrow_\pi \lambda\mathtt{R}.\lambda\mathtt{X}.\lambda\mathtt{Y}.(\mathtt{R}\,\mathtt{X}\,\mathtt{Y})$$
$$\mathtt{closure} \leftarrow_\pi \lambda\mathtt{R}.\lambda\mathtt{X}.\lambda\mathtt{Y}.\exists\mathtt{Z}((\mathtt{R}\,\mathtt{X}\,\mathtt{Z}) \wedge (\mathtt{closure}\,\mathtt{R}\,\mathtt{Z}\,\mathtt{Y}))$$

A possible query could be: $\leftarrow \mathtt{closure}\,\mathtt{R}\,\mathtt{a}\,\mathtt{b}$ (which intuitively requests for all binary relations such that the pair $(\mathtt{a}, \mathtt{b})$ belongs to their transitive closure). $\Box$

4 The Semantics of $\mathcal{H}$

The semantics of $\mathcal{H}$ is built upon the notion of algebraic complete lattice. We start with the semantics of types and proceed with the semantics of expressions.

4.1 The Semantics of Types

The set-theoretic meaning of the types of $\mathcal{H}$ are specified with respect to a set D (where D is later going to be the domain of our interpretations). The fact that a given type π denotes a set $[\![\pi]\!]_D$ will mean that an expression of type π denotes an element of $[\![\pi]\!]_D$. Similarly, an expression of type ρ denotes an element of $[\![\rho]\!]_D$. In the following definition we define simultaneously (and recursively) two things: the semantics $[\![\tau]\!]_D$ of a type τ and the corresponding partial order $\sqsubseteq_\tau$.

Definition 11. *Let D be a non-empty set. Then:*

- $[\![\iota]\!]_D = D$ *and $\sqsubseteq_\iota$ is the binary relation such that $d \sqsubseteq_\iota d$, for all $d \in D$.*
- $[\![\iota^n \to \iota]\!]_D = D^n \to D$. *A partial order for this case will not be needed.*
- $[\![o]\!]_D = \{0, 1\}$ *and $\sqsubseteq_o$ is the numerical ordering on $\{0, 1\}$.*
- $[\![\iota \to \pi]\!]_D = D \to [\![\pi]\!]_D$. *Moreover, for all $f, g \in [\![\iota \to \pi]\!]_D$, $f \sqsubseteq_{\iota\to\pi} g$ if and only if $f(d) \sqsubseteq_\pi g(d)$, for all $d \in D$.*
- $[\![\pi_1 \to \pi_2]\!]_D = [\mathcal{K}([\![\pi_1]\!]_D) \xrightarrow{m} [\![\pi_2]\!]_D]$. *Moreover, for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D$, $f \sqsubseteq_{\pi_1\to\pi_2} g$ if and only if $f(d) \sqsubseteq_{\pi_2} g(d)$, for all $d \in \mathcal{K}([\![\pi_1]\!]_D)$.*

Obviously each $[\![\pi]\!]_D$ is an algebraic complete lattice (ω-algebraic if D is countable), due to the fact that the poset $\{0,1\}$ is an ω-algebraic complete lattice and Lemma 1. The following definition gives us a convenient shorthand that will be used in various places of the paper:

Definition 12. *Let D be a non-empty set and let ρ be an argument type. Define:*

$$\mathcal{F}_D(\rho) = \begin{cases} D, & \text{if } \rho = \iota \\ \mathcal{K}([\![\rho]\!]_D), & \text{otherwise} \end{cases}$$

The set $\mathcal{F}_D(\rho)$ will be called the set of basic elements of type ρ.

Example 2. Consider the type $\iota \to o$ (a first-order predicate with one argument has this type). Then, $[\![\iota \to o]\!]_D$ is the set of all functions from D to $\{0,1\}$ (or equivalently, of arbitrary subsets of D). Moreover, it can be verified by Definitions 1 and 12 that the set $\mathcal{F}_D(\iota \to o)$ is the set of all *finite* functions from D to $\{0,1\}$ (or equivalently, of finite subsets of D).

As a second example, consider the type $(\iota \to o) \to o$. This is the type of a predicate which takes as its only parameter another predicate which is first-order. Then, $[\![(\iota \to o) \to o]\!]_D$ is the set of all monotonic functions from finite sets (ie., elements of $\mathcal{F}_D(\iota \to o)$) to $\{0,1\}$. In other words, in the semantics of the higher-order language that we will develop, a predicate of type $(\iota \to o) \to o$ will denote a monotonic function from finite subsets of D to $\{0,1\}$, ie., it will denote a set of finite subsets of D that respects monotonicity. On the other hand, it can be verified that the set $\mathcal{F}_D((\iota \to o) \to o)$ contains all the relations that have the following property: each one of them can be written as the union of a finite number of simpler relations each one of which consists of a finite set of elements of D together with all its finite supersets. □

4.2 The Semantics of Expressions

We can now proceed to give meaning to the expressions of $\mathcal{H}$. This is performed by first defining the notions of *interpretation* and *state* for $\mathcal{H}$, that are similar to the corresponding notions for first-order languages:

Definition 13. *An interpretation I of $\mathcal{H}$ consists of:*

1. *a nonempty set D, called the* domain *of I*
2. *an assignment to each individual constant symbol c, of an element $I(\mathsf{c}) \in D$*
3. *an assignment to each predicate symbol $\mathsf{p} : \pi$, of an element $I(\mathsf{p}) \in [\![\pi]\!]_D$*
4. *an assignment to each function symbol f of type $\iota^n \to \iota$, of a function $I(\mathsf{f}) \in D^n \to D$.*

Definition 14. *Let I be a given interpretation with domain D. Then, a* state s over I *is a function that assigns to each argument variable Q of type ρ of $\mathcal{H}$, an element $s(\mathsf{Q}) \in \mathcal{F}_D(\rho)$.*

The key technical difficulty we now have to confront is the definition of the semantics of application. The problem that arises can be explained by an example (written in Prolog-like syntax):

```
p(Q):-Q(0),Q(s(0)).
nat(0).
nat(s(X)):-nat(X).
```

Consider the query ← `p(nat)`. The type of `p` is $(\iota \to o) \to o$ while the type of `nat` is $\iota \to o$. Let I be an interpretation of our program with underlying domain D. Then, $I(\mathtt{p})$ must be a function from $\mathcal{F}_D(\iota \to o)$ to $\{0,1\}$. According to Example 2, $\mathcal{F}_D(\iota \to o)$ consists of *finite* sets of elements of D. But $I(\mathtt{nat})$ is obviously an infinite set. How can we apply $I(\mathtt{p})$ to $I(\mathtt{nat})$? The key idea is that if a higher-order predicate of our language is true of a relation, then this fact can be established by examining a "finite number of facts" about this relation. In our case, `p` just examines for its input relation `Q` whether it is true of `0` and `s(0)`. These remarks suggest that the meaning of `p(nat)` can be established as follows: we apply $I(\mathtt{p})$ to the "finite approximations" of $I(\mathtt{nat})$, ie., to all elements of $\mathcal{F}_D(\iota \to o)_{[I(\mathtt{nat})]}$, and then take the least upper bound of the results. In our case `p(nat)` will be true since there exists a finite fragment of $I(\mathtt{nat})$ for which $I(\mathtt{p})$ is true (namely the set $\{I(\mathtt{0}), I(\mathtt{s(0)})\}$.

In the following definition, $s[d/\mathsf{X}]$ is used to denote a state that is identical to s the only difference being that the new state assigns to X the value d.

Definition 15. *Let I be an interpretation of $\mathcal{H}$, let D be the domain of I, and let s be a state over I. Then, the semantics of expressions of $\mathcal{H}$ with respect to I and s, is defined as follows:*

1. $[\![\mathsf{0}]\!]_s(I) = 0$
2. $[\![\mathsf{1}]\!]_s(I) = 1$
3. $[\![\mathsf{c}]\!]_s(I) = I(\mathsf{c})$, *for every individual constant* c
4. $[\![\mathsf{p}]\!]_s(I) = I(\mathsf{p})$, *for every predicate variable* p
5. $[\![\mathsf{Q}]\!]_s(I) = s(\mathsf{Q})$, *for every argument variable* Q
6. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I) \cdots [\![\mathsf{E}_n]\!]_s(I)$, *for every n-ary function symbol* f
7. $[\![(\mathsf{E}_1\mathsf{E}_2)]\!]_s(I) = \bigsqcup_{b_2 \in B_2}([\![\mathsf{E}_1]\!]_s(I)(b_2))$, *where* $B_2 = \mathcal{F}_D(type(\mathsf{E}_2))_{[[\![\mathsf{E}_2]\!]_s(I)]}$
8. $[\![(\lambda \mathsf{X}.\mathsf{E})]\!]_s(I) = \lambda d.[\![\mathsf{E}]\!]_{s[d/\mathsf{X}]}(I)$, *where d ranges over* $\mathcal{F}_D(type(\mathsf{X}))$
9. $[\![(\mathsf{E}_1 \leftarrow_\pi \mathsf{E}_2)]\!]_s(I) = \begin{cases} 1, \textit{ if } [\![\mathsf{E}_2]\!]_s(I) \sqsubseteq_\pi [\![\mathsf{E}_1]\!]_s(I) \\ 0, \textit{ otherwise} \end{cases}$
10. $[\![(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)]\!]_s(I) = \bigsqcup_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, *where* $\bigsqcup_\pi$ *is the least upper bound function on* $[\![\pi]\!]_D$
11. $[\![(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)]\!]_s(I) = \bigsqcap_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, *where* $\bigsqcap_\pi$ *is the greatest lower bound function on* $[\![\pi]\!]_D$
12. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(I) = \begin{cases} 1, \textit{ if } [\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I) \\ 0, \textit{ otherwise} \end{cases}$
13. $[\![(\exists \mathsf{Q}\ \mathsf{E})]\!]_s(I) = \begin{cases} 1, \textit{ if there exists } d \in \mathcal{F}_D(type(\mathsf{Q})) \textit{ such that } [\![\mathsf{E}]\!]_{s[d/\mathsf{Q}]}(I) = 1 \\ 0, \textit{ otherwise} \end{cases}$

For closed expressions E we will often write $[\![\mathsf{E}]\!](I)$ instead of $[\![\mathsf{E}]\!]_s(I)$ (in this case, the meaning of E is independent of s). We now define the notion of *model*:

Definition 16. *Let S be a set of closed formulas of $\mathcal{H}$ and let I be an interpretation of $\mathcal{H}$. We say that I is a* model *of S if for every $\mathsf{F} \in S$, $[\![\mathsf{F}]\!](I) = 1$.*

4.3 Herbrand Interpretations

Herbrand interpretations are a cornerstone of first-order logic programming. Analogously, we have:

Definition 17. *The Herbrand universe $U_{\mathcal{H}}$ of $\mathcal{H}$ is the set of all terms that can be formed out of the individual constants and the function symbols of $\mathcal{H}$.*

Definition 18. *A Herbrand interpretation I of $\mathcal{H}$ is an interpretation such that:*

1. *The domain of I is the Herbrand universe $U_{\mathcal{H}}$ of $\mathcal{H}$.*
2. *For every individual constant c, $I(\mathsf{c}) = \mathsf{c}$.*
3. *For every predicate symbol p of type π, $I(\mathsf{p}) \in [\![\pi]\!]_{U_{\mathcal{H}}}$.*
4. *For every n-ary function symbol f and all $\mathsf{t}_1, \ldots, \mathsf{t}_n \in U_{\mathcal{H}}$, $I(\mathsf{f})\,\mathsf{t}_1 \cdots \mathsf{t}_n = \mathsf{f}\,\mathsf{t}_1 \cdots \mathsf{t}_n$.*

Since all Herbrand interpretations have the same underlying universe, we will often refer to a "Herbrand state s", meaning a state whose underlying universe is $U_{\mathcal{H}}$. We will often also refer to an "interpretation of a set of formulas S" rather than the underlying language $\mathcal{H}$. In this case, we will implicitly assume that the set of individual constants and function symbols are those that appear in S. Under this assumption, we will often talk about the Herbrand universe U_S of a set of formulas S. The set of Herbrand interpretations of a given program, forms a complete lattice:

Definition 19. *Let P be a program and let $\mathcal{I}_{\mathsf{P}}$ be the set of Herbrand interpretations of P. We define the following partial order on $\mathcal{I}_{\mathsf{P}}$: for all $I, J \in \mathcal{I}_{\mathsf{P}}$, $I \sqsubseteq_{\mathcal{I}_{\mathsf{P}}} J$ iff for every predicate variable p of P, $I(\mathsf{p}) \sqsubseteq J(\mathsf{p})$.*

Lemma 2. *Let P be a program and let $\mathcal{I}_{\mathsf{P}}$ be the set of Herbrand interpretations of P. Then, $\mathcal{I}_{\mathsf{P}}$ is a complete lattice under $\sqsubseteq_{\mathcal{I}_{\mathsf{P}}}$.*

In the following we denote with $\bot_{\mathcal{I}_{\mathsf{P}}}$ the least element of $\mathcal{I}_{\mathsf{P}}$, ie., the interpretation that assigns to each predicate $\mathsf{p} : \pi$ of P the element $\bot_\pi$.

5 Minimum Herbrand Model Semantics

The basic properties of logic programming extend to the higher-order case:

Theorem 1 (Model Intersection Theorem). *Let P be a program and $\mathcal{M}$ a non-empty set of Herbrand models of P. Then, $\bigsqcap \mathcal{M}$ is a Herbrand model for P.*

It is straightforward to check that every higher-order program P has at least one Herbrand model I, namely the one which for every predicate symbol p and for all basic elements $b_1, \ldots, b_n$ of the appropriate types, $I(\mathsf{p})\, b_1 \cdots b_n = 1$. Therefore, the intersection of all Herbrand models is well-defined, and by the above theorem is a model of the program. We will denote this model by M_{P}.

Definition 20. *Let P be a higher order program. The* immediate consequence operator *$T_{\mathsf{P}} : \mathcal{I}_{\mathsf{P}} \to \mathcal{I}_{\mathsf{P}}$ is defined as $T_{\mathsf{P}}(I)(\mathsf{p}) = \bigsqcup_{(\mathsf{p} \leftarrow_\pi \mathsf{E}) \in \mathsf{P}} [\![\mathsf{E}]\!](I)$, for every $\mathsf{p} : \pi$ in P and for every $I \in \mathcal{I}_{\mathsf{P}}$.*

Lemma 3. *Let* P *be a program. Then the mapping* T_{P} *is continuous.*

Define now the following sequence of interpretations:

$$\begin{aligned} T_{\mathsf{P}} \uparrow 0 &= \bot_{\mathcal{I}_{\mathsf{P}}} \\ T_{\mathsf{P}} \uparrow (n+1) &= T_{\mathsf{P}}(T_{\mathsf{P}} \uparrow n) \\ T_{\mathsf{P}} \uparrow \omega &= \bigsqcup\{T_{\mathsf{P}} \uparrow n \mid n < \omega\} \end{aligned}$$

The following theorem is entirely analogous to the one for the first-order case:

Theorem 2. *Let* P *be a program. Then* $M_{\mathsf{P}} = T_{\mathsf{P}} \uparrow \omega$.

6 Proof Procedure

In this section we propose a sound and complete proof-procedure for extensional higher-order logic programming.

6.1 Basic Expressions

Basic elements (introduced in Section 4) have played an important role in the development of the semantics of our higher-order logic programming language. In order to devise a sound and complete proof procedure for our language, we first need to find a syntactic representation for basic elements:

Definition 21. *The set of* basic expressions of $\mathcal{H}$ *is recursively defined as follows. The* basic expressions *of type* ι *are all expressions of* $\mathcal{H}$ *of type* ι. *The basic expressions of type* o *are* $\mathsf{0}$ *and* $\mathsf{1}$.

A basic expression of type $\rho_1 \to \cdots \to \rho_n \to o$ *is a non-empty finite union of lambda abstractions each of which has one of the following forms:*

1. $\lambda \mathsf{X}_1 . \cdots \lambda \mathsf{X}_n . \mathsf{0}$
2. $\lambda \mathsf{X}_1 . \cdots \lambda \mathsf{X}_n . \mathsf{1}$
3. $\lambda \mathsf{X}_1 . \cdots \lambda \mathsf{X}_n . \mathsf{A}_1 \wedge \cdots \wedge \mathsf{A}_m$, *where* $m > 0$ *and each* A_i *is either*
   (a) $(\mathsf{X}_k \approx \mathsf{t})$, *if* X_k *is of type* ι *and* t *is a basic expression of type* ι *whose variables are different from* $\mathsf{X}_1, \ldots, \mathsf{X}_n$, *or*
   (b) X_k, *if* X_k *is of type* o, *or*
   (c) $\mathsf{X}_k(\mathsf{B}_1)\cdots(\mathsf{B}_r)$, *if* X_k *is of type* $\rho'_1 \to \cdots \to \rho'_r \to o$ *and each* B_j *is a basic expression of type* ρ'_j.

In the above definition, if any ρ_k *is of type* ι *then the body of each abstraction within the finite union must either be* $\mathsf{0}$ *or contain exactly one expression of the form* $(\mathsf{X}_k \approx \mathsf{t})$.

The proof procedure that will be developed later in this section, relies on a special form of basic expressions:

Definition 22. *A basic expression* B *is called a* basic template *if in every subexpression of the form* $(\mathsf{X} \approx \mathsf{t})$ *in* B, *the term* t *is a variable that does not appear in any other place of* B.

The following two lemmas suggest that basic expressions are the syntactic analogues of basic elements:

Lemma 4. *For every basic expression* $\mathsf{B} : \rho$*, for every interpretation* I *with domain* D*, and for every state* s *over* I*,* $[\![\mathsf{B}]\!]_s(I) \in \mathcal{F}_D(\rho)$*.*

The converse of the above lemma holds if we restrict attention to Herbrand interpretations and basic elements over the Herbrand universe.

Lemma 5. *Let* ρ *be any argument type and let* $b \in \mathcal{F}_{U_\mathcal{H}}(\rho)$*. Then, there exists a ground basic expression* $\mathsf{B} : \rho$ *such that for every Herbrand interpretation* I*,* $[\![\mathsf{B}]\!](I) = b$*.*

6.2 Substitutions and Unifiers

Definition 23. *A* substitution θ *is a finite set* $\{\mathsf{V}_1/\mathsf{E}_1, \ldots, \mathsf{V}_n/\mathsf{E}_n\}$*, where the* V_i*'s are different argument variables of* $\mathcal{H}$ *and each* E_i *is an expression of* $\mathcal{H}$ *having the same type as* V_i*. We write* $dom(\theta) = \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ *and* $range(\theta) = \{\mathsf{E}_1, \ldots, \mathsf{E}_n\}$*. A substitution is called* basic *is all* E_i *are basic expressions.*

Definition 24. *Let* θ *be a substitution and let* E *be a positive expression. Then,* $\mathsf{E}\theta$ *is an expression obtained from* E *as follows:*

- $\mathsf{E}\theta = \mathsf{E}$*, if* E *is* $\mathsf{0}$*,* $\mathsf{1}$*,* c*, or* p*.*
- $\mathsf{Q}\theta = \theta(\mathsf{Q})$ *if* $\mathsf{Q} \in dom(\theta)$*; otherwise,* $\mathsf{Q}\theta = \mathsf{Q}$*.*
- $(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)\theta = (\mathsf{f}\ \mathsf{E}_1\theta \cdots \mathsf{E}_n\theta)$*.*
- $(\mathsf{E}_1\mathsf{E}_2)\theta = (\mathsf{E}_1\theta\,\mathsf{E}_2\theta)$*.*
- $(\lambda\mathsf{X}.\mathsf{E}_1)\theta = (\lambda\mathsf{Z}.(\mathsf{E}_1\{\mathsf{X}/\mathsf{Z}\})\theta)$*, where* $\mathsf{Z} \notin FV(\mathsf{E}_1) \cup FV(dom(\theta)) \cup FV(range(\theta))$*.*
- $(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)\theta = (\mathsf{E}_1\theta \bigvee_\pi \mathsf{E}_2\theta)$*.*
- $(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)\theta = (\mathsf{E}_1\theta \bigwedge_\pi \mathsf{E}_2\theta)$*.*
- $(\mathsf{E}_1 \approx \mathsf{E}_2)\theta = (\mathsf{E}_1\theta \approx \mathsf{E}_2\theta)$*.*
- $(\exists\mathsf{Q}\,\mathsf{E}_1)\theta = (\exists\mathsf{Z}\,(\mathsf{E}_1\{\mathsf{Q}/\mathsf{Z}\})\theta)$*, where* $\mathsf{Z} \notin FV(\mathsf{E}_1) \cup FV(dom(\theta)) \cup FV(range(\theta))$*.*

The composition of substitutions can be defined in a similar way as in the first-order case:

Definition 25. *Let* $\theta = \{\mathsf{V}_1/\mathsf{E}_1, \ldots, \mathsf{V}_m/\mathsf{E}_m\}$ *and* $\sigma = \{\mathsf{Q}_1/\mathsf{E}'_1, \ldots, \mathsf{Q}_n/\mathsf{E}'_n\}$ *be substitutions. Then the composition* $\theta\sigma$ *of* θ *and* σ *is the substitution obtained from the set* $\{\mathsf{V}_1/\mathsf{E}_1\sigma, \ldots, \mathsf{V}_m/\mathsf{E}_m\sigma, \mathsf{Q}_1/\mathsf{E}'_1, \ldots, \mathsf{Q}_n/\mathsf{E}'_n\}$ *by deleting any* $\mathsf{V}_i/\mathsf{E}_i\sigma$ *for which* $\mathsf{V}_i = \mathsf{E}_i\sigma$ *and deleting any* $\mathsf{Q}_j/\mathsf{E}'_j$ *for which* $\mathsf{Q}_j \in \{\mathsf{V}_1, \ldots, \mathsf{V}_m\}$*.*

The substitution corresponding to the empty set will be called the *identity substitution* and will be denoted by ϵ. The following proposition is easy to establish:

Proposition 1. *Let* θ, σ *and* γ *be substitutions. Then:*

1. $\theta\epsilon = \epsilon\theta = \theta$*.*
2. *For all positive expressions* E*,* $(\mathsf{E}\theta)\sigma = \mathsf{E}(\theta\sigma)$*.*
3. $(\theta\sigma)\gamma = \theta(\sigma\gamma)$*.*

where equality should be understood as α-congruence ie., as syntactic equality subject to a possible renaming of bound variables.

Definition 26. *Let S be a set of terms of $\mathcal{H}$ (ie., expressions of type ι). A substitution θ will be called a* unifier *of the expressions in S if the set $S\theta = \{\mathsf{E}\theta \mid \mathsf{E} \in S\}$ is a singleton. The substitution θ will be called a* most general unifier *of S (denoted by $mgu(S)$), if for every unifier σ of the expressions in S, there exists a substitution γ such that $\sigma = \theta\gamma$.*

6.3 SLD Resolution

We now proceed to define the notions of *answer* and *correct answer*. Notice that both of these notions rely on *basic* (ie., not arbitrary) substitutions:

Definition 27. *Let P be a program and G a goal. An* answer *for $\mathsf{P} \cup \{\mathsf{G}\}$ is a basic substitution for free variables of G.*

Definition 28. *Let P be a program, $\mathsf{G} = \leftarrow \mathsf{E}$ a goal clause and θ an answer for $\mathsf{P} \cup \{\mathsf{G}\}$. We say that θ is* a correct answer *for $\mathsf{P} \cup \{\mathsf{G}\}$ if for every model M of P and for every state s over M, $[\![\mathsf{E}\theta]\!]_s(M) = 1$.*

Definition 29. *Let P be a program and let $\mathsf{G} = \leftarrow \mathsf{A}$ and $\mathsf{G}' = \leftarrow \mathsf{A}'$ be goal clauses. Then, we will say that A' is derived in one step from A using the basic substitution θ (or equivalently that G' is derived in one step from G using θ), and we denote this fact by $\mathsf{A} \xrightarrow{\theta} \mathsf{A}'$ (respectively, $\mathsf{G} \xrightarrow{\theta} \mathsf{G}'$) if one of the following conditions applies:*

1. $\mathsf{p}\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\epsilon} \mathsf{E}\ \mathsf{E}_1 \cdots \mathsf{E}_n$, *where* $\mathsf{p} \leftarrow_\pi \mathsf{E}$ *is a rule in* P.
2. $\mathsf{Q}\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\theta} (\mathsf{Q}\ \mathsf{E}_1 \cdots \mathsf{E}_n)\theta$, *where* $\theta = \{\mathsf{Q}/\mathsf{B}\}$ *and* B *is a basic template such that* $FV(\mathsf{B}) \cap FV(\{\mathsf{E}_1, \ldots, \mathsf{E}_n\}) = \emptyset$.
3. $(\lambda\mathsf{X}.\mathsf{E})\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\epsilon} (\mathsf{E}\{\mathsf{X}/\mathsf{E}_1\})\,\mathsf{E}_2 \cdots \mathsf{E}_n$.
4. $(\mathsf{E}' \bigvee_\pi \mathsf{E}'')\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\epsilon} \mathsf{E}'\ \mathsf{E}_1 \cdots \mathsf{E}_n$.
5. $(\mathsf{E}' \bigvee_\pi \mathsf{E}'')\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\epsilon} \mathsf{E}''\ \mathsf{E}_1 \cdots \mathsf{E}_n$.
6. $(\mathsf{E}' \bigwedge_\pi \mathsf{E}'')\ \mathsf{E}_1 \cdots \mathsf{E}_n \xrightarrow{\epsilon} (\mathsf{E}'\ \mathsf{E}_1 \cdots \mathsf{E}_n) \wedge (\mathsf{E}''\ \mathsf{E}_1 \cdots \mathsf{E}_n)$.
7. $(\mathsf{E}_1 \wedge \mathsf{E}_2) \xrightarrow{\theta} (\mathsf{E}_1' \wedge (\mathsf{E}_2\theta))$, *if* $\mathsf{E}_1 \xrightarrow{\theta} \mathsf{E}_1'$.
8. $(\mathsf{E}_1 \wedge \mathsf{E}_2) \xrightarrow{\theta} ((\mathsf{E}_1\theta) \wedge \mathsf{E}_2')$, *if* $\mathsf{E}_2 \xrightarrow{\theta} \mathsf{E}_2'$.
9. $(\Box \wedge \mathsf{E}) \xrightarrow{\epsilon} \mathsf{E}$
10. $(\mathsf{E} \wedge \Box) \xrightarrow{\epsilon} \mathsf{E}$
11. $(\mathsf{E}_1 \approx \mathsf{E}_2) \xrightarrow{\theta} \Box$, *where θ is an mgu of* E_1 *and* E_2.
12. $(\exists\mathsf{Q}\,\mathsf{E}) \xrightarrow{\epsilon} \mathsf{E}$

Definition 30. *Let P be a program and G a goal. An* SLD-derivation *of $\mathsf{P} \cup \{\mathsf{G}\}$ consists of (possibly infinite) sequences $\mathsf{G}_0 = \mathsf{G}, \mathsf{G}_1, \ldots$ of goals and $\theta_1, \theta_2, \ldots$ of basic substitutions such that each G_{i+1} is derived in one step from G_i using θ_{i+1}.*

Definition 31. *Let* P *be a program and* G *a goal. An* SLD-refutation *of* $\mathsf{P} \cup \{\mathsf{G}\}$ *is a finite SLD-derivation of* $\mathsf{P} \cup \{\mathsf{G}\}$ *which has the empty clause* $\square$ *as the last goal in the derivation. If* $\mathsf{G}_n = \square$*, then we say that the refutation has length* n*.*

Definition 32. *Let* P *be a program and* G *a goal. A computed answer* θ *for* $\mathsf{P} \cup \{\mathsf{G}\}$ *is the basic substitution obtained by restricting the composition* $\theta_1 \cdots \theta_n$ *to the free variables of* G*, where* $\theta_1, \ldots, \theta_n$ *is the sequence of the basic substitutions used in an SLD-refutation of* $\mathsf{P} \cup \{\mathsf{G}\}$ *of length* n*.*

Example 3. Consider the program of Example 1. We have:

$\mathtt{closure\ Q\ a\ b}$	$\theta_0 = \epsilon$
$(\lambda \mathtt{R}.\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{R\ X\ Y}))\ \mathtt{Q\ a\ b}$	$\theta_1 = \epsilon$
$\mathtt{Q\ a\ b}$	$\theta_2 = \{\mathtt{Q}/(\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{X} \approx \mathtt{X}_0) \wedge (\mathtt{Y} \approx \mathtt{Y}_0))\}$
$(\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{X} \approx \mathtt{X}_0) \wedge (\mathtt{Y} \approx \mathtt{Y}_0))\ \mathtt{a\ b}$	$\theta_3 = \epsilon$
$(\mathtt{a} \approx \mathtt{X}_0) \wedge (\mathtt{b} \approx \mathtt{Y}_0)$	$\theta_4 = \{\mathtt{X}_0/\mathtt{a}\}$
$\square \wedge (\mathtt{b} \approx \mathtt{Y}_0)$	$\theta_5 = \epsilon$
$(\mathtt{b} \approx \mathtt{Y}_0)$	$\theta_6 = \{\mathtt{Y}_0/\mathtt{b}\}$
$\square$	

where some simple steps involving lambda abstractions have been omitted. The composition of the above substitutions gives the substitution σ_1 below. Similarly we can get σ_2, etc.

$$\sigma_1 = \{\mathtt{Q}/\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{X} \approx \mathtt{a}) \wedge (\mathtt{Y} \approx \mathtt{b})\}$$
$$\sigma_2 = \{\mathtt{Q}/(\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{X} \approx \mathtt{a}) \wedge (\mathtt{Y} \approx \mathtt{Z})) \bigvee\nolimits_{\pi} \ (\lambda \mathtt{X}.\lambda \mathtt{Y}.(\mathtt{X} \approx \mathtt{Z}) \wedge (\mathtt{Y} \approx \mathtt{b}))\}$$
$$\ldots$$

Notice that σ_1 above corresponds to the set $\{(\mathtt{a}, \mathtt{b})\}$, σ_2 to the set $\{(\mathtt{a}, \mathtt{Z}), (\mathtt{Z}, \mathtt{b})\}$, for every $\mathtt{Z}$ in the Herbrand universe, and so on. $\square$

6.4 Soundness and Completeness of SLD-resolution

The proofs of soundness and completeness of the proposed proof procedure (which due to space limitations will appear in the full version of the paper) follow along similar lines as the corresponding results for the first-order case.

Theorem 3 (Soundness). *Let* P *be a program and* G *a goal. Then, every computed answer for* $\mathsf{P} \cup \{\mathsf{G}\}$ *is a correct answer for* $\mathsf{P} \cup \{\mathsf{G}\}$*.*

As in the first-order case, we have various forms of completeness. We start with the analogue of a theorem due to Apt and van Emden (see [Llo87]).

Theorem 4. *Let* P *be a program and* G *a goal and* $[\![\mathsf{G}]\!]_s(M_{\mathsf{P}}) = 1$ *for all states* s*. Then, there is a refutation for* $\mathsf{P} \cup \{\mathsf{G}\}$ *using the identity substitution.*

The following is a generalization of Hill's theorem (see [Llo87][Theorem 8.4]) for the higher-order case:

Theorem 5. *Let* P *be a program,* G *a goal and assume that* $\mathsf{P} \cup \{\mathsf{G}\}$ *is unsatisfiable (ie., it does not have any models). Then, there exists an SLD-refutation of* $\mathsf{P} \cup \{\mathsf{G}\}$.

Finally, the following is a generalization of Clark's theorem (see [Llo87][Theorem 8.6]) for the higher-order case:

Theorem 6 (Completeness). *Let* P *a program and* G *a goal. For every correct answer* θ *for* $\mathsf{P} \cup \{\mathsf{G}\}$*, there exists an SLD-refutation for* $\mathsf{P} \cup \{\mathsf{G}\}$ *using the computed answer* σ *and a basic substitution* γ *such that* $\theta = \sigma\gamma$.

7 Conclusions

An implementation of the proposed proof procedure has been performed in Haskell[1]. The main difference in comparison to a first order implementation, is that the proof procedure has to generate an infinite (yet enumerable) number of basic templates. This affects the search tree, making it in general infinite not only in depth but also in breadth. As a result, the naive depth first search strategy would in general be unfair with respect to the enumeration of the solutions. In our implementation we use the strategy for interleaving different solutions proposed in [KSFS05], which solves the search problem in a satisfactory way.

References

[AJ94] Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science III. Clarendon Press, Oxford (1994) (expanded version)

[Bez99] Bezem, M.: Extensionality of Simply Typed Logic Programs. In: International Conference on Logic Programming (ICLP), pp. 395–410 (1999)

[CKW93] Chen, W.C., Kifer, M., Warren, D.S.: HILOG: A Foundation for Higher-Order Logic Programming. J. of Logic Programming 15(3), 187–230 (1993)

[KSFS05] Kiselyov, O., Shan, C.C., Friedman, D.P., Sabry, A.: Backtracking, Interleaving, and Terminating Monad Transformers. In: International Conference on Functional Programming (ICFP), pp. 192–203 (2005)

[Llo87] Lloyd, J.: Foundations of Logic Programming. Springer, Heidelberg (1987)

[NM98] Nadathur, G., Miller, D.: Higher-Order Logic Programming. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.) Handbook of Logics for Artificial Intelligence and Logic Programming, pp. 499–590. Clarendon Press, Oxford (1998)

[Wad91] Wadge, W.W.: Higher-Order Horn Logic Programming. In: Proceedings of the International Symposium on Logic Programming, pp. 289–303 (1991)

[War82] Warren, D.H.D.: Higher-Order Extensions to Prolog: are they needed? Machine Intelligence 10, 441–454 (1982)

[1] The code can be retrieved from `http://code.haskell.org/hopes`

dl2asp: Implementing Default Logic via Answer Set Programming

Yin Chen[1], Hai Wan[2], Yan Zhang[3], and Yi Zhou[3]

[1] Department of Computer Science, South China Normal University, Guangzhou, China, 510631
[2] School of Software, Sun Yat-Sen University, Guangzhou, China, 510275
[3] School of Computing and Information Technology, University of Western Sydney, Penrith South DC, NSW 1797, Australia

Abstract. In this paper, we show that Reiter's default logic in the propositional case can be translated into answer set programming by identifying the internal relationships among formulas in a default theory. Based on this idea, we implement a new default logic solver - dl2asp. We report some experimental results, in particular the application of dl2asp for solving the fair division problem in social choice theory.

1 Introduction

As a predominant approach for nonmonotonic reasoning, Default Logic (DL) [21] has attracted many researchers in the last three decades. Default logic is theoretically significant not only because of its elegant syntax and semantics, but also its expressive power to capture other nonmonotonic reasoning approaches, such as autoepistemic logic, defeasible reasoning, and so on [1,11,8,22].

However, despite the remarkable success on theoretical aspects, default logic has encountered huge difficulties from a practical viewpoint. Although many endeavors have been done [3,18,17], the implementation of default logic still remains unsatisfactory. Consequently, the practical value of default logic has been severely restricted.

This paper intends to address this issue by translating default logic into Answer Set Programming (ASP) [7], a promising approach that has been successfully implemented by a number of sophisticated solvers [6,19,12,13]. It is well-known that ASP is a special case of default logic by restricting the formulas in default theories to atoms/literals [8,22]. An interesting question arises whether the converse can also be done to some extent. In other words, is it possible to translate default logic back into answer set programming?

We answer this question positively. We show that default logic in the propositional case can be translated to answer set programming by identifying the internal relationships among formulas in a default theory. By internal relationships, we mean those implication rules whose head is a formula, whose body is a set of formulas occurred in the default theory, and the body entails the head in propositional logic.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 104–116, 2010.
© Springer-Verlag Berlin Heidelberg 2010

This translation is not only theoretically interesting but also of practical relevance. Based on the translation, we implement a new solver for default logic, called dl2asp. We report some experimental results, which demonstrate that the performance of dl2asp is rather satisfactory.

The paper is organized as follows. Section 2 recalls some basic notions and definitions in default logic and answer set programming. Section 3 presents the translation from default logic to answer set programming, and discusses some related properties. Section 4 explains the implementation of dl2asp in detail. Section 5 reports some experiments, while Section 6 considers an application of dl2asp for solving the fair division problem in social choice theory. Finally, Section 7 concludes the paper.

2 Preliminaries

2.1 Reiter's Default Logic

We consider Reiter's default logic [21] in propositional case. A *default theory* Δ is a pair $\langle W, D\rangle$, where W is a set of propositional formulas and D is a set of *defaults* of the following form:

$$\alpha : \beta_1, \ldots, \beta_n / \gamma, \tag{1}$$

where $\alpha, \beta_1, \ldots, \beta_n, \gamma$ are propositional formulas. In addition, α is called the *prerequisite*, $\beta_1, \ldots, \beta_n$ the *justifications*, and γ the *conclusion* of the default. Let Δ be a default theory. We use P_Δ, J_Δ and C_Δ to denote the sets of prerequisites, justifications and conclusions occurred in the default theory respectively.

Definition 1 (Extension [21]). *Let $\Delta = \langle W, D\rangle$ be a default theory and T a theory. We say that T is an* extension *of Δ if $T = \Gamma(T)$, where for any theory S, $\Gamma(S)$ is the minimal set (in the sense of set inclusion) satisfying the following three conditions:*

1. *$W \subseteq \Gamma(S)$.*
2. *$\Gamma(S)$ is a theory.*
3. *For any default rule $\alpha : \beta_1, \ldots, \beta_n / \gamma \in D$, if $\alpha \in \Gamma(S)$ and $\neg\beta_i \notin S, (1 \leq i \leq n)$, then $\gamma \in \Gamma(S)$.*

Example 1. Consider the default theory $\Delta_1 = \langle W_1, D_1\rangle$, where $W_1 = \{\neg b \vee \neg c, c \vee d\}$ and D_1 contains the following four defaults:

$$: \neg b / a, \tag{2}$$
$$: \neg a, \neg c / b, \tag{3}$$
$$: a \wedge \neg b / \neg d, \tag{4}$$
$$\neg c : \neg a / \neg a. \tag{5}$$

It can be checked that Δ_1 has two extensions (under equivalence): $E_1 = Th(W_1 \cup \{a, \neg d\})$ and $E_2 = Th(W_1 \cup \{\neg a, b\})$.[1]

[1] We use $Th(S)$ to denote the deductive closure of a set S of formulas.

2.2 Answer Set Programming

An *answer set program* (*program* for short) is a set of *rules* of the following form:

$$a \leftarrow b_1, \ldots, b_m, \mathsf{not}\ b_{m+1}, \ldots, \mathsf{not}\ b_n, \tag{6}$$

where $0 \leq m \leq n$, a is either an atom or $\bot$, and $b_1, \ldots b_n$ are atoms. In addition, a is called the *head* of the rule and $\{b_1, \ldots, b_m, \mathsf{not}\ b_{m+1}, \ldots, \mathsf{not}\ b_n\}$ the *body* of the rule. More specifically, $\{b_1, \ldots, b_m\}$ is called the *positive body* of the rule, while $\{b_{m+1}, \ldots, b_n\}$ is called the *negative body* of the rule. We call a rule a *constraint* if a is $\bot$, *fact* if $n = 0$, and *positive* if $m = n$.

Let X be a set of atoms. We say that X *satisfies* a rule of form (6) if X satisfies its head (i.e. $a \in X$) whenever X satisfies its body (i.e. $\{b_1, \ldots, b_m\} \subseteq X$ and $\{b_{m+1}, \ldots, b_n\} \cap X = \emptyset$). Hence, X satisfies a constraint iff it does not satisfy its body.

Definition 2 (Answer set [7]). *Let Π be a program and X a set of atoms. We say that X is an* answer set *of Π if X is the minimal set (in the sense of set inclusion) that satisfies Π^X, where Π^X is obtained as follows:*

- *delete all rules whose bodies are not satisfied by X;*
- *delete* $\mathsf{not}\ b_i$ *in the bodies of the remaining rules.*

Gelfond and Lifschitz [8] showed that answer set programming is a special case of default logic by simply rewriting a rule of form (6) to the following default:

$$b_1 \wedge \ldots \wedge b_m : \neg b_{m+1}, \ldots, \neg b_n / a. \tag{7}$$

Let Π be a program. By $DL(\Pi)$, we denote the default theory obtained from Π as above (Note that W in $DL(\Pi)$ is obtained by facts in Π). Then, the answer sets of Π and the extensions of $DL(\Pi)$ are one-to-one corresponded.

Theorem 1 (From ASP to DL [8]). *Let Π be a program and X a set of atoms. Then, X is an answer set of Π iff $Th(X)$ is an extension of $DL(\Pi)$.*

An interesting question arises whether the converse of Theorem 1 holds as well. In other words, is it possible to translate default logic back into answer set programming? In the next section, we answer it positively.

3 From Default Logic to Answer Set Programming

This section translates Reiter's default logic in propositional case to answer set programming. To begin with, let us take a closer look at the translation from ASP to DL, which. In fact, this translation indicates that the rule of form (6) in ASP plays the same role of the default (7) in DL. This means that, conversely, a specific kind of default, namely of form (7) plays the same role to an ASP rule, namely of form (6). Analogously, its suggests that a default of form (1) in DL, i.e.

$$\alpha : \beta_1, \ldots, \beta_n / \gamma,$$

should play a similar role to

$$\gamma \leftarrow \alpha, \mathsf{not}\ \neg\beta_1, \ldots, \mathsf{not}\ \neg\beta_n.$$

However, this is not exactly an ASP rule. To fix this problem, we can simply introduce a new atom p_α for each formula α. Then, analogous to the translation from ASP to DL, a default of form (1) in DL should play a similar role to the following rule in ASP:

$$p_\gamma \leftarrow p_\alpha, \mathsf{not}\ p_{\neg\beta_1}, \ldots, \mathsf{not}\ p_{\neg\beta_n}. \tag{8}$$

Now, we have a naive translation from DL to ASP. Formally, let $\Delta = \langle W, D\rangle$ be a default theory. We introduce a set of new atoms p_α for each formula $\alpha \in W \cup P_\Delta \cup \neg J_\Delta \cup C_\Delta$.[2] Let r be a default of form (1), by $R(r)$, we denote the ASP rule of form (8). Let $\Delta = \langle W, D\rangle$ be a default theory, by $R(\Delta)$, we denote the program

$$\{p_\alpha \mid \alpha \in W\} \cup \{R(r) \mid r \in D\}.$$

Indeed, $R(\Delta)$ is the answer set program obtained from the default theory Δ by mapping each formula occurred in the default theory to a corresponding new atom.

Example 2. Recall Example 1. According to the above definition, $R(\Delta_1)$ contains the following rules:

$$\begin{array}{ll} p_{\neg b \vee \neg c} \leftarrow & p_b \leftarrow \mathsf{not}\ p_a, \mathsf{not}\ p_c \\ p_{c \vee d} \leftarrow & p_{\neg d} \leftarrow \mathsf{not}\ p_{\neg(a \wedge \neg b)} \\ p_a \leftarrow \mathsf{not}\ p_b & p_{\neg a} \leftarrow p_{\neg c}, \mathsf{not}\ p_a. \end{array}$$

It can be checked that $R(\Delta_1)$ has two answer sets: $M_1 = \{p_{\neg b \vee \neg c}, p_{c \vee d}, p_a, p_{\neg d}\}$ and $M_2 = \{p_{\neg b \vee \neg c}, p_{c \vee d}, p_b, p_{\neg d}\}$. Compared to the extensions of Δ_1, while M_1 exactly corresponds to E_1, M_2 and E_2 are not related.

Let us take a closer look at Examples 1 and 2. One may observe that there is no extension of Δ_1 containing b when $\neg d$ holds because $W_1 \cup \{\neg d\} \models c$, thus default (3) in Δ_1 will never be triggered. This is the reason why M_2 fails to correspond to any extension of Δ_1. Also, suppose that b is in an extension of Δ_1. Then, $\neg c$ must be in the extension as well because $W \cup \{b\} \models \neg c$. As such, default (5) in Δ_1 could be triggered so that $\neg a$ is also in the extension. This explains why there is no answer sets of $R(\Delta_1)$ corresponding to E_2.

In general, we can conclude that what is missing in $R(\Delta)$ are the internal relationships among formulas in Δ. By simply translating Δ to $R(\Delta)$, some internal relationships among formulas are lost, which might be crucial for computing a default theory's extensions, for instance, the entailment relationship $W_1 \cup \{\neg d\} \models c$ as discussed above.

Our main theoretical result in this paper is: together with the internal relationships, $R(\Delta)$ exactly captures the extensions of Δ. Formally, let Δ be a

[2] We use $\neg J_\Delta$ to denote the set of formulas $\{\neg\phi \mid \phi \in J_\Delta\}$.

default theory and $F_\Delta = W \cup P_\Delta \cup \neg J_\Delta \cup C_\Delta$. The set of *implication rules* of Δ, denoted by $I(\Delta)$, is the set of rules of the form

$$p_\phi \leftarrow p_{\phi_1}, \ldots, p_{\phi_n} \tag{9}$$

where $\{\phi, \phi_1, \ldots \phi_n\} \subseteq F_\Delta$, $\phi \notin \{\phi_1, \ldots, \phi_n\}$ and $\{\phi_1, \ldots, \phi_n\} \models \phi$.

Finally, let $AS(\Delta) = R(\Delta) \cup I(\Delta)$. The following theorem shows that $AS(\Delta)$ exactly captures the extensions of Δ. That is, the answer sets of $AS(\Delta)$ is one-to-one corresponding to the extensions of Δ.

Theorem 2 (From DL to ASP). *Let Δ be a default theory and T a consistent theory.*[3] *Then, T is an extension of Δ iff p_T is an answer set of $AS(\Delta)$, where $p_T = \{p_\alpha \mid \alpha \in F_\Delta, T \models \alpha\}$.*

Proof (sketch). [4] Firstly, it is observed that if T is an extension of Δ, then there exists $F \subseteq F_\Delta$ such that T is the deductive closure of $W \cup F$.

Now, suppose that T is an extension of Δ. Then, p_T satisfies all rules in $AS(\Delta)$ according to the definition. Thus, p_T satisfies $AS(\Delta)^{p_T}$. Assume that p_T is not an answer set of $AS(\Delta)$. Then, there exists $X \subset p_T$ such that X satisfies $AS(\Delta)^{p_T}$ as well. Let T' be the deductive closure of $\{\alpha \mid p_\alpha \in X\}$. Then, it can be checked that T' satisfies the conditions of the operator Γ with respect to T (note that X satisfies $I(\Delta)$). Hence, $\Gamma(T) \subseteq T'$. In addition, $T' \subset T$ since $X \subset p_T$. This shows that $\Gamma(T) \subseteq T' \subset T$, a contradiction.

On the other hand, suppose that p_T is an answer set of $AS(\Delta)$. Then, p_T satisfies each rule in $R(\Delta)$. Therefore, T satisfies the conditions of the operator Γ with respect to T itself. Hence, $\Gamma(T) \subseteq T$. Assume that $\Gamma(T) \subset T$. Then, $p_{\Gamma(T)}$ satisfies $R(\Delta)^{p_T}$ according to the definition of $R(\Delta)$. Also, $p_{\Gamma(T)}$ satisfies $I(\Delta)^{p_T}$ according to the definitions of $I(\Delta)$ and $p_{\Gamma(T)}$. Hence, $p_{\Gamma(T)}$ satisfies $AS(\Delta)^{p_T}$. In addition, $p_{\Gamma(T)} \subset p_T$ since $\Gamma(T) \subset T$. This shows that p_T is not an answer set of $AS(\Delta)$, a contradiction.

Although $AS(\Delta)$ exactly captures the extensions of Δ, the implication rules in $I(\Delta)$ could be a lot. Next, we propose several techniques to reduce the number of implication rules by the following observations:

- In ASP, suppose that there are two rules sharing the same head, the same negative body but one rule's positive body is a subset of another's. Then, the latter rule is "dummy" because these two rules are strongly equivalent to the former one [15]. Hence, we can only consider those "minimal" implication rules for a default theory Δ.
- In DL, given a default theory $\Delta = \langle W, D \rangle$, all the extensions must contain W. Hence, we can fix W for the implication rules. That is, we can only consider those internal relationships (i.e. implication rules) generated from the set $P_\Delta \cup \neg J_\Delta \cup C_\Delta$ under the context of W.

[3] Here, we only consider consistent extensions. Inconsistency can be easily checked in default logic [22].

[4] Due to a space limit, proofs in this paper, if given, are sketched.

- Suppose that there exists an inconsistent (unsatisfiable) set of formulas. Then, for any other ϕ, it generates an implication rule. Of course, this is not necessary. All we need is a constraint stating that these formulas (their corresponding atoms) cannot appear at the same time.
- Observed from Theorem 2.5 [21], all the extensions of a default theory Δ can be rewritten as $Th(W \cup C)$, where C is a subset of C_Δ. Hence, we only need to consider the implication rules whose heads are only from $P_\Delta \cup \neg J_\Delta$ and bodies are only from C_Δ.

Based on the above observations and discussions, we can simplify the implication rules as follows. Let $\Delta = \langle W, D\rangle$ be a default theory. The set of *modified implication rules* of Δ, denoted by $I^*(\Delta)$, is the set of rules of form (9), where

- either ϕ is $\bot$[5] $\phi_i \in C_\Delta$, $W \cup \{\phi_1, \ldots, \phi_n\}$ is unsatisfiable, and $\{\phi_1, \ldots, \phi_n\}$ is the minimal set satisfying the above conditions,
- or $\phi \in P_\Delta \cup \neg J_\Delta$, $\phi_i \in C_\Delta \setminus \{\phi\}$, $W \cup \{\phi_1, \ldots, \phi_n\}$ is satisfiable, $W \cup \{\phi_1, \ldots, \phi_n\} \models \phi$, and $\{\phi_1, \ldots, \phi_n\}$ is the minimal set satisfying the above conditions.

Example 3. Recall Δ_1 discussed in Examples 1 and 2 again. According to the definition, $I^*(\Delta)$ is the set of the following rules:

$$\bot \leftarrow p_a, p_{\neg a} \qquad p_c \leftarrow p_{\neg d}$$
$$\bot \leftarrow p_b, p_{\neg d} \qquad p_{\neg(a \wedge \neg b)} \leftarrow p_b$$
$$p_{\neg c} \leftarrow p_b \qquad p_{\neg(a \wedge \neg b)} \leftarrow p_{\neg a}$$

It can be checked that $R(\Delta) \cup I^*(\Delta)$ has two answer sets: $\{p_{\neg b \vee \neg c}, p_{c \vee d}, p_a, p_{\neg d}\}$ and $\{p_{\neg b \vee \neg c}, p_{c \vee d}, p_{\neg a}, p_b\}$, which are exactly corresponding to the extensions E_1 and E_2 of Δ_1 respectively.

In general, let $AS^*(\Delta) = R(\Delta) \cup I^*(\Delta)$. The following theorem shows that $AS^*(\Delta)$ is enough to capture the extensions of Δ.

Theorem 3. *Let Δ be a default theory and T a consistent theory. Then, T is an extension of Δ iff p^*_T is an answer set of $AS^*(\Delta)$, where $p^*_T = \{p_\alpha \mid \alpha \in W \cup C_\Delta, T \models \alpha\}$.*

Clearly, the atoms used in $AS^*(\Delta)$ is linear with respect to the size of Δ. However, although the number of implication rules in $I^*(\Delta)$ is significantly reduced compared to $I(\Delta)$, there might be exponential number of such rules. It is well-known that checking whether a default theory in the propositional case has an extension is Σ^P_2 complete [10], while checking whether a normal logic program has an answer set is NP complete [4]. Hence, the exponential size seems inevitable, providing some general assumptions in the complexity theory.

However, as we will show later in the experiments, the number of implication rules is not really explosive. This is also partially evidenced by the research of

[5] In this case, we define $p_\bot$ as $\bot$ for convenience.

minimal unsatisfiable subset. For instance, the experiments in [14] showed that the number of MUS (corresponding to minimal implication rules in $I^*(\Delta)$) is not big for many cases. Also, although simple, it is worth mentioning that if the default theory Δ itself is ASP-like (i.e. $W \cup C_\Delta \cup \neg J_\Delta \cup P_\Delta$ only contains atoms), then $I^*(\Delta)$ is empty.

A closely related work is due to Dao-Tran et al. [20] for translating default logic to so-called description logic knowledge bases, which is an extension of ASP with description logics. Restricted in the propositional case, while this work is to translate DL to an extension of ASP, ours directly translates DL to ASP itself.

4 Implementation

Based on Theorems 2 and 3, we have implemented a new solver for default logic, called dl2asp, which computes all extensions of a given default theory. More precisely, the key idea of dl2asp is to translate a given default theory Δ to $AS^*(\Delta)$. Then, by Theorem 3, the task of computing all the extensions of Δ turns into computing all the answer sets of $AS^*(\Delta)$.

A default theory is encoded in an ASCII file in dl2asp. For example, the default theory Δ_1 in Example (1) is encoded as follows:

```
-b | -c                          c | d
: -b / a                         : -a, -c / b
: a & -b / -d                    -c : -a / -a
```

Here, "-", "|" and "&" stand for the connectives $\neg$, $\vee$ and $\wedge$ respectively.

Figure 1 illustrates how dl2asp works. An input default theory is firstly translated to an answer set program by the *translator* in dl2asp. Then, an *ASP solver* is called to compute the answer sets of the program. Finally, the answer sets will be interpreted back to extensions of the original default theory by a *convertor*.

For the *ASP solver* module in dl2asp, we just use *clasp*[6]. The *convertor* in dl2asp is trivial. As demonstrated in Theorem 3, one can simply interpret each atom in the answer sets to their corresponding formulas. Hence, the main issue in dl2asp is the *translator*.

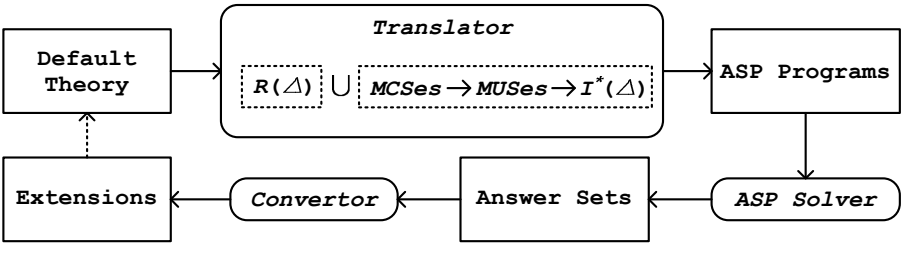


Fig. 1. Outline of dl2asp

[6] http://www.cs.uni-potsdam.de/clasp/

Let $\Delta = \langle W, D\rangle$ be a default theory. According to the construction, $AS^*(\Delta)$ contains two parts, namely $R(\Delta)$ and $I^*(\Delta)$. For $R(\Delta)$, we can first introduce a new atom p_α for each formula $\alpha \in F_\Delta$, then get a new fact p_α for each formula $\alpha \in W$ and get a new rule of form (8) for each default in D of form (1). However, $I^*(\Delta)$ is relatively difficult. The most technical part of dl2asp is to compute $I^*(\Delta)$ - the set of modified implication rules.

For this purpose, we borrow some ideas and techniques of computing minimal unsatisfiable subsets from [14]. Let S be a set of formulas. A subset S' of S is a *Minimal Unsatisfiable Subset (MUS)* if S' is unsatisfiable and for any $S'' \subset S'$, S'' is satisfiable. Liffiton and Sakallah [14] developed a sound and complete algorithm, called CAMUS[7], for computing all MUSes of a given set of clauses. However, instead of computing MUSes directly, they computed so-called MCSes first. Here, a subset S' of S is a *Maximal Correction Subset (MCS)* if $S \backslash S'$ is satisfiable and for any $S'' \subset S'$, $S \backslash S''$ is unsatisfiable. MCS and MUS are closely related. In fact, all the MUSes are exactly all the minimal hitting sets of the collection of all MCSes. Here, given a collection of sets Ω, a set H is a *minimal hitting set* of Ω iff for all $H_0 \in \Omega$, $H \cap H_0 \neq \emptyset$ and there is no $H' \subset H$ satisfying the above condition.

We now apply this method to compute all the modified implication rules (i.e. $I^*(\Delta)$) in our translation. Let Δ be a default theory. According to the construction, $I^*(\Delta)$ contains rules of the form $p_\phi \leftarrow p_{\phi_1}, \ldots, p_{\phi_n}$, where $\{p_{\phi_1}, \ldots, p_{\phi_n}\}$ is a minimal set such that $W \cup \{\phi_1, \ldots, \phi_n\} \models \phi$. That is, it is a minimal set inconsistent with $W \cup \{\neg\phi\}$. In other words, it is an MUS of the set $W \cup \{\neg\phi\} \cup C_\Delta$ by fixing W and $\{\neg\phi\}$. Hence, we can compute all the implication rules in $I^*(\Delta)$ as follows:

constraints compute all the MUSes of $W \cup C_\Delta$ by fixing W;
other implication rules for all $\phi_i \in \neg J_\Delta \cup P_\Delta$, compute all the MUSes of $W \cup \{\neg\phi_i\} \cup C_\Delta$ by fixing W and $\{\neg\phi_i\}$.

There are two major differences between this task and the one in [14] for computing all MUSes. Firstly, we need to fix some formulas, e.g. formulas in W. Secondly, we need to compute the MUSes for every ϕ in $P_\Delta \cup \neg J_\Delta$. Clearly, the task of computing constraints can be considered as a special case of the second case. In dl2asp, we first compute constraints, then the other implication rules, where the latter is implemented by Algorithm 1.

Let use take a closer look at Algorithm 1. Step 1 fixes W as a background formula set. Steps 2-4 (Steps 5-6) add a selector variable x_i (y_j) to the formula $\neg\psi_i$ (ϕ_j resp.). The *AtMost* constraint in Step 7 is provided by MiniSat [5] to restrict that at most one of $\{x_1, \ldots, x_m\}$ holds, and together with Step 8, exactly one of $\{\neg\psi_1, \ldots, \neg\psi_m\}$ holds. In fact, x_i is corresponding to those implication rules whose head is p_{ψ_i}. Steps 9-16 compute all the MCSes in a similar way to Figure 2 in [14] except that W and at most one of ψ_i are fixed. *SAT* in Step 10 and *IncrementalSAT* in Step 12 uses MiniSat's incremental solving ability to find a model of a formula. *BlockingClause* in Steps 14 and 15 is pro-

[7] http://www.eecs.umich.edu/~liffiton/camus/

vided by CAMUS to block the MCS that obtained before. Steps 17-20 compute all the implication rules in the same way to computing MUSes in [14], where $ConstructMUS$ is provided by CAMUS for computing all minimal hitting sets, and *Rewrite* is the function to rewrite an MUS to an implication rule of form (9), and delete those self implication rules like $p_\phi \leftarrow p_\phi$ and those implication rules like $p_\phi \leftarrow p_{\phi_1}, \ldots, p_{\phi_n}$ when there is already a constraint $\bot \leftarrow p_{\phi_1}, \ldots, p_{\phi_n}$.

Algorithm 1. Computing Implication Rules

input : A default theory $\Delta = \langle W, D \rangle$, where $C_\Delta = \{\phi_1, \ldots, \phi_n\}$ and $P_\Delta \cup \neg J_\Delta = \{\psi_1, \ldots, \psi_m\}$
output: the set of non-constraint implication rules of Δ

1 $\Phi \leftarrow W$ // set W as background theory
2 **forall** $1 \leq i \leq m$ **do** // add selector variables to $\neg\psi$
3 $MCS(\neg\psi_i) \leftarrow \emptyset$
4 $\Phi \leftarrow \Phi \cup \{\neg\psi_i \vee \neg x_i\}$
5 **forall** ϕ_j *such that* $1 \leq j \leq n$ **do** // add selector variables to ϕ
6 $\Phi \leftarrow \Phi \cup \{\phi_j \vee \neg y_j\}$
7 $\Phi \leftarrow \Phi \cup AtMost(\{x_1, \ldots, x_m\}, 1)$ // at most one of x_i holds
8 $\Phi \leftarrow \Phi \cup \{x_1 \vee \cdots \vee x_m\}$ // at least one of x_i holds
9 $k \leftarrow 1$
10 **while** $(SAT(\Phi))$ **do** // computing $MCSes$
11 $\Phi_k \leftarrow \Phi \cup AtMost(\{\neg y_1, \ldots, \neg y_n\}, k)$
12 **while** $(M = IncrementalSAT(\Phi_k))$ **do**
13 $MCS(\neg\psi_i) \leftarrow MCS(\psi_i) \cup \{\phi_j \mid y_j \in M\}$
14 $\Phi_k \leftarrow \Phi_k \cup BlockingClause(M)$
15 $\Phi \leftarrow \Phi \cup BlockingClause(M)$
16 $k \leftarrow k + 1$
17 $I^*(\Delta) \leftarrow \emptyset$
18 **forall** $1 \leq i \leq m$ **do** // computing implication rules
19 $I^*(\Delta) \leftarrow I^*(\Delta) \cup Rewrite(ConstructMUS(\psi_i))$
20 return $I^*(\Delta)$

Example 4. Again, recall Δ_1 discussed in Examples 1 and 2, and consider the formula $\neg(a \wedge \neg b) \in \neg J_\Delta$. According to Algorithm 1, we have $MCS(\neg(a \wedge \neg b)) = \{\{b, \neg a\}\}$ and $MUS(\neg(a \wedge \neg b)) = \{\{b\}, \{\neg a\}\}$. So, we have two rules $p_{\neg(a \wedge \neg b)} \leftarrow p_b$ and $p_{\neg(a \wedge \neg b)} \leftarrow p_{\neg a}$ in $I^*(\Delta_1)$ corresponding to the result above.

Finally, we end up this section by showing how the rest parts of dl2asp work for the example. By calling *clasp*, $R(\Delta_1) \cup I^*(\Delta_1)$ has two answer sets: $\{p_{\neg b \vee \neg c}, p_{c \vee d}, p_a, p_{\neg d}\}$ and $\{p_{\neg b \vee \neg c}, p_{c \vee d}, p_{\neg a}, p_b\}$. Then, the *convertor* just rewrite them as: $\{\neg b \vee \neg c, c \vee d, a, \neg d\}$ and $\{\neg b \vee \neg c, c \vee d, \neg a, b\}$. In contrast with Example 1, these two formula sets are exactly the two extensions of Δ_1.

5 Experimental Results

In this section, we report some experimental results of dl2asp. First, we briefly review some existing solvers for default logic, including DeReS, XRay, and GADEL. Unfortunately, we are unable to run them properly because these solvers are out of date, and the versions of softwares they used are too old to be compatible with current versions.

Nevertheless, it is still necessary to review these approaches, particularly their test data. The task of all these approaches is to compute all extensions of a given default theory. For this purpose, DeReS [3] directly uses search algorithms to compute the extensions. In [3], some benchmarks in graph theory, e.g. finding Hamiltonian circuit, are tested. Instead, XRay [18] is implemented by using Prolog, and supporting local proof procedures. XRay tested the Hamiltonian circuit problem and some contextual default theories randomly generated. GADEL [16] applies genetic algorithms to compute the extensions. GADEL tested the Hamiltonian circuit problem as well as a handed-coded default theory about relationships among people (Example 4.1, [16]).

Similarly, dl2asp intends to find all extensions of a given default theory as well. First of all, it is observed that if the default theory is ASP-like (i.e. $W \cup C_\Delta \cup \neg J_\Delta \cup P_\Delta$ only contains atoms), then the set of implication rules is empty. Furthermore, if the default theory is disjunction-free (i.e. all the formulas occurred in the default theory are literals), then all the implication rules are constraints, and the total number is linear. This is because the implication rules can only be of the form $\bot \leftarrow p_l, \mathsf{not}\ p_{\neg l}$, where l is a literal occurred in the default theory and $\neg l$ is the complementary literal of l. In both cases, Algorithm 1 can find out all the implication rules immediately. Hence, it is not interesting to test such default theories for dl2asp.

However, most of the test benchmarks, e.g. the Hamiltonian circuit problem, belong to the above two categories. Therefore, in this section, we only report our experimental results for solving the people's relationship problem[8] (Example 4.1 in [16]), which is claimed difficult to be solved in default logic.

The system dl2asp is written in C++. The program is running on a machine with 4 processors($AMD\ Athlon^{tm}\ II\ X4\ 620$) under Ubuntu 9.10 Linux operating system. We record two series of time in seconds, $time_t$ - the time for computing all implication rules and $time_{all}$ - the overall time, taking the average of 3 runs. num_{I^*} is the number of the rules in $I^*(\Delta_\mathcal{P})$. The experimental results of the people's relationship problem are summarized in Table 1.

Although we do not intend to compare dl2asp with other solvers on this particular instance because of fairness reasons, we can see that dl2asp performs rather satisfactory on this benchmark. As an example, for $woman \wedge student$, while dl2asp only takes 0.3 seconds, GADEL and DeRes takes 1202 seconds and more than 7200 seconds respectively under their test environments (see Table 2 in [16]).

[8] For more details about this particular default theory, please refer to [16].

Table 1. Experimental results of the people's relationship problem

	boy	*girl*	*man*	*woman*	*man* ∧ *student*	*woman* ∧ *student*
num_{I^*}	10	10	11	12	10	11
$time_t$	0.5134	0.5134	0.6427	0.7001	0.2453	0.2894
$time_{all}$	0.5227	0.5228	0.6534	0.7120	0.2520	0.2987

6 Application to Fair Division Problem

In this section, we apply dl2asp for solving the fair division problem, which is one of the central problems in social choice theory. The problem of fair division is: given a set of agents, a set of goods and the preference among goods for each agent, to obtain a "fair" solution for allocating the goods to the agents [2].

Formally, a *fair division problem* is a tuple $\mathcal{P} = \langle I, X, \mathcal{R}\rangle$, where $I = \{1, \ldots, N\}$ is a set of agents, $X = \{x_1, \ldots, x_p\}$ is a set of indivisible goods, and $\mathcal{R} = \{R_1, \ldots, R_N\}$ is a preference profile, where each R_i is a reflexive, transitive and complete relation on 2^X. An *(complete) allocation* for $\mathcal{P} = \langle I, X, \mathcal{R}\rangle$ is a mapping $\pi : I \to 2^X$ such that for all i and $j \neq i$, $\pi(i) \cap \pi(j) = \emptyset$, and for every $x \in X$ there exists an i such that $x \in \pi(i)$. An allocation π is *(Pareto-) efficient* iff there is no π' such that π' dominates π, where for two allocations π and π', π dominates π' iff for all i, $(\pi(i), \pi'(i)) \in R_i$, and there exists an i such that $(\pi'(i), \pi(i)) \notin R_i$. An allocation π is *envy-free* iff $(\pi(i), \pi(j)) \in R_i$ holds for all i and all $j \neq i$.

A preference R_i is *dichotomous* iff there exists a subset $Good_i$ of 2^X such that for all $A, B \subseteq X$, $(A, B) \in R_i$ iff $A \in Good_i$ or $B \notin Good_i$. A dichotomous preference can be naturally represented by a single propositional formula, where variables correspond to goods. Given a dichotomous preference R_i, we can always use formula $\phi_i = \bigvee_{A \in Good_i} (\bigwedge_{x \in A} x \wedge \bigwedge_{x \notin A} \neg x)$ to represent R_i. Thus, a fair division problem with dichotomous preference $\mathcal{P} = \langle I, X, \mathcal{R}\rangle$ can always be represented by $\langle \phi_1, \ldots, \phi_N\rangle$, and I, X and $\mathcal{R}$ are obviously determined from $\langle \phi_1, \ldots, \phi_N\rangle$. The following proposition shows that for fair division problem with dichotomous preference, it can be translated into default logic.

Proposition 1 (Proposition 3, [2]). *Let $\mathcal{P} = \langle \phi_1, \ldots, \phi_N\rangle$ be a fair division problem. Let $\Delta_\mathcal{P}$ be the default theory $\langle \Gamma_\mathcal{P}, \Phi_\mathcal{P} \cup \{\neg \Lambda_\mathcal{P} : /\bot\}\rangle$, where*

- $\Gamma_\mathcal{P} = \bigwedge_{x \in X} \bigwedge_{i \neq j} \neg(x_i \wedge x_j)$,
- $\Lambda_\mathcal{P} = \bigwedge_{i=1,\ldots,N} \left[\phi_i^* \vee \left(\bigwedge_{j \neq i} \neg \phi^*_{j|i}\right)\right]$, *and*
- $\Phi_\mathcal{P}$ *is the set of defaults of the form* $: \phi_i^*/\phi_i^*$, $i = 1, \ldots, N$,

where ϕ_i^* is obtained from ϕ_i by replace every variable x by a new symbol x_i, and $\phi^*_{j|i}$ is obtained from ϕ_i^* by replace every symbol x_i by x_j. Then, each extension of $\Delta_\mathcal{P}$ is corresponding to an efficient and envy-free allocation of $\mathcal{P}$.

Based on this result, we are able to use dl2asp to compute all efficient and envy-free allocations for a given fair division problem. Given the numbers of goods and agents, we randomly generate a fair division problem instance, then

Table 2. Experimental results of Fair Division Problem

$(goods, agents)$	$(3,4)$	$(4,4)$	$(4,6)$	$(4,8)$	$(4,10)$	$(4,15)$	$(4,20)$	$(4,25)$
num_{I^*}	1	1	1	1	14	21	410	460
$time_t$	0.0213	0.0267	0.0653	0.2213	0.4427	1.7534	21.8267	33.2407
$time_{all}$	0.0333	0.0547	0.1000	0.2787	0.5320	1.9148	22.2241	33.8248
EXT	0	0	0	0	5	13	170	177

compute all the extensions of the default theory according to Proposition 1. Our experimental results are shown in Table 2, where EXT is the number of extensions returned by dl2asp and num_{I^*}, $time_t$ and $time_{all}$ are the same as those in Table 1.

7 Conclusions and Future Work

This paper contributes the study of default logic both from a theoretical and a practical point of view. Theoretically, we showed that default logic can be translated into answer set programming by identifying the internal relationships (i.e. implication rules) among formulas (Theorem 2), and indeed, the number of such implication rules can be largely reduced (Theorem 3). Practically, based on the above translation, we developed a new solver - dl2asp - for implementing default logic via answer set programming. Our experimental results (Table 1) illustrated that the performance of dl2asp is rather satisfactory, which is further confirmed by applying dl2asp to solving the fair division problem (Table 2).

One of our future work is to consider the relationships between default logic and disjunctive answer set programming as they are on the same complexity level. For instance, an open problem is whether DL can be naturally translated to disjunctive ASP, or the other way around. If not, then an interesting question arises, for a particular application on the Σ_2^P level, whether it is better to be encoded in DL or in disjunctive ASP.

For other future directions, one important task is to develop more techniques for improving dl2asp, especially for computing the implication rules. Another work worth pursuing is to extend dl2asp for more expressive default logics, such as disjunctive default logic [9] and general default logic [22]. Last but not least, it is interesting to explore more applications of default logic by using dl2asp.

Acknowledgments

We would like to thank Jérôme Lang for his generous help on the fair division problem, and the anonymous reviewers for their valuable comments. This research is supported in part by an Australian Research Council Linkage Project grant LP0883646, and the first author is also partially supported by Natural Science Foundation of China under grant NSFC60705095 and Natural Science Foundation of GuangDong Province, China under grant GDSF07300237.

References

1. Bochman, A.: Default logic generalized and simplified. Ann. Math. Artif. Intell. 53(1-4), 21–49 (2008)
2. Bouveret, S., Lang, J.: Efficiency and envy-freeness in fair division of indivisible goods: Logical representation and complexity. JAIR 32, 525–564 (2008)
3. Cholewinski, P., Marek, V.W., Truszczynski, M., Mikitiuk, A.: Computing with default logic. Artificial Intelligence (AIJ) 112(1-2), 105–146 (1999)
4. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3), 374–425 (2001)
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
6. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI, pp. 386–392 (2007)
7. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080 (1988)
8. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
9. Gelfond, M., Przymusinska, H., Lifschitz, V., Truszczynski, M.: Disjunctive defaults. In: KR, pp. 230–237 (1991)
10. Gottlob, G.: Complexity results for nonmonotonic logics. Journal of Logic and Computation 2, 397–425 (1992)
11. Janhunen, T.: On the intertranslatability of autoepistemic, default and priority logics, and parallel circumscription. In: Dix, J., Fariñas del Cerro, L., Furbach, U. (eds.) JELIA 1998. LNCS (LNAI), vol. 1489, pp. 216–232. Springer, Heidelberg (1998)
12. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Log. 7(3), 499–562 (2006)
13. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 346–350. Springer, Heidelberg (2003)
14. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning (JAR) 40(1), 1–33 (2008)
15. Lin, F., Chen, Y.: Discovering classes of strongly equivalent logic programs. J. Artif. Intell. Res. (JAIR) 28, 431–451 (2007)
16. Nicolas, P., Saubion, F., Stéphan, I.: Gadel: a genetic algorithm to compute default logic extensions. In: ECAI, pp. 484–490 (2000)
17. Nicolas, P., Saubion, F., Stéphan, I.: Heuristics for a default logic reasoning system. IJAIT 10(4), 503–523 (2001)
18. Nicolas, P., Schaub, T.: The xray system: an implementation platform for local query-answering in default logics. In: Hunter, A., Parsons, S. (eds.) Applications of Uncertainty Formalisms. LNCS (LNAI), vol. 1455, pp. 354–378. Springer, Heidelberg (1998)
19. Niemelä, I., Simons, P.: Smodels - an implementation of the stable model and well-founded semantics for normal lp. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 421–430. Springer, Heidelberg (1997)
20. Dao-Tran, M., Eiter, T., Krennwallner, T.: Realizing Default Logic over Description Logic Knowledge Bases. In: Sossai, C., Chemello, G. (eds.) ECSQARU 2009. LNCS, vol. 5590, pp. 602–613. Springer, Heidelberg (2009)
21. Reiter, R.: A logic for default reasoning. Artificial Intelligence (AIJ) 13(1-2), 81–132 (1980)
22. Zhou, Y., Lin, F., Zhang, Y.: General default logic. Annals of Mathematics and Artificial Intelligence (2010) (to appear)

Sets of Boolean Connectives That Make Argumentation Easier*

Nadia Creignou[1], Johannes Schmidt[1], Michael Thomas[2], and Stefan Woltran[3]

[1] LIF, UMR CNRS 6166, Aix-Marseille Université
163, Avenue de Luminy, 13288 Marseille Cedex 9, France
creignou@lif.univ-mrs.fr, johannes.schmidt@lif.univ-mrs.fr
[2] Institut für Theoretische Informatik, Gottfried Wilhelm Leibniz Universität
Appelstr. 4, 30167 Hannover, Germany
thomas@thi.uni-hannover.de
[3] Institut für Informationssysteme E184/2, Technische Universität Wien
Favoritenstr. 9–11, 1040 Wien, Austria
woltran@dbai.tuwien.ac.at

Abstract. Many proposals for logic-based formalizations of argumentation consider an argument as a pair (Φ, α), where the support Φ is understood as a minimal consistent subset of a given knowledge base which has to entail the claim α. In most scenarios, arguments are given in the full language of classical propositional logic which makes reasoning in such frameworks a computationally costly task. For instance, the problem of deciding whether there exists a support for a given claim has been shown to be Σ_2^{p}-complete. In order to better understand the sources of complexity (and to identify tractable fragments), we focus on arguments given over formulae in which the allowed connectives are taken from certain sets of Boolean functions. We provide a complexity classification for four different decision problems (existence of a support, checking the validity of an argument, relevance and dispensability) with respect to all possible sets of Boolean functions.

1 Introduction

Argumentation is nowadays a main research topic within the area of Artificial Intelligence ([BD07, BH08, RS09]) aiming to formally analyze pros and cons of statements within a certain scenario in order to, for instance, support decision making. There are (among others) two important lines of research: abstract argumentation [Dun95] focuses on the relation between arguments without taking their internal structure into account; deductive (or logic-based) argumentation [CML00, PV02, BH08] starts from a concrete formal representation of an argument and then defines on top of this concept notions such as counterarguments, rebuttals and more complex structures like argument trees [BH01].

* Supported by ANR *Algorithms and complexity* 07-BLAN-0327-04, WWTF grant ICT 08-028, and DFG grant VO 630/6-2.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 117–129, 2010.
© Springer-Verlag Berlin Heidelberg 2010

In logic-based argumentation, most proposals consider an argument as a pair (Φ, α), where the support Φ is a consistent set (or a minimal consistent set) of formulae from a given knowledge base that entails the claim α which is a formula (see, for example, [BH01, AC02, GS04, DKT06]). Different logical formalisms provide different definitions for consistency and entailment and hence give different options for defining the notion of an argument. One natural candidate for formalizing arguments is the full language of classical propositional logic. However, it is computationally challenging to generate arguments from a knowledge base Δ using classical logic; in fact, the problem of deciding whether there exists a support $\Phi \subseteq \Delta$ for a given claim α has been shown to be Σ_2^{p}-complete [PWA03].

Computing the support for an argument underlies many reasoning problems in logic-based argumentation, for instance, the computation of argument trees as proposed by Besnard and Hunter [BH01]. Since the basic task of finding a support is already computationally involved, it is indispensable to understand its sources of complexity and to identify fragments for which that problem becomes tractable. In this paper, we contribute to this line of research by restricting the formulae involved (*i.e.*, formulae in the knowledge base and thus in the support, as well as the formula used as the claim). In fact, we restrict formulae to connectives from a given set taken from certain sets of Boolean functions and study the decision problems of existence, validity, relevance, and respectively, dispensability, which are defined as follows: Arg (given Δ, α, does there exist a support $\Phi \subseteq \Delta$ of α), Arg-Check (given a pair (Φ, α), is it an argument), Arg-Rel (given Δ, α, φ, is there an argument (Φ, α) such that $\varphi \in \Phi \subseteq \Delta$), and Arg-Disp (given Δ, α, φ, is there an argument (Φ, α) such that $\varphi \notin \Phi \subseteq \Delta$). We understand here as arguments pairs (Φ, α) with *minimal* support, *i.e.*, (Φ, α) is an argument if Φ is consistent, entails α, and no $\Phi' \subsetneq \Phi$ entails α. It can be seen that the minimality condition is only important for the problems Arg-Check and Arg-Rel (for instance, in case of Arg-Disp, there exists a support Φ without φ for α exactly if there exists a minimal such support); we will make this more precise in Section 3. We also mention that the problem of Arg-Rel is of particular importance, since it allows to determine (in terms of decision problems) the actual form of a potential support, an important core problem in constructing argument trees.

The main contribution of this paper is a systematic complexity classification for these four problems in terms of all possible sets of Boolean connectives. We show that, depending on the chosen set of connectives, the problems range from inside P up to the second level of the polynomial hierarchy, and we identify those fragments complete for NP, coNP, and also for DP, the class of differences of problems in NP. These fragments highlight the sources of complexity of the problems. We also show that unless the polynomial hierarchy collapses there exist particular sets of Boolean connectives such that: (i) deciding the existence of an argument is easier than verifying a given one; (ii) deciding the dispensability of a formula for some argument is easier than deciding its relevance.

The paper is structured as follows. Section 2 contains preliminaries. We define the studied framework of argumentation and relevant decision problems in

Section 3. The complexity of these problems is subsequently classified in the Sections 4 to 6. Section 7 concludes with a discussion of related work and provides an overview of the achieved results as well as future research directions.

2 Preliminaries

We require standard notions of complexity theory. For the decision problems the arising complexity degrees encompass the classes LOGSPACE, P, NP, coNP, DP and Σ_2^{p}, where DP is defined as the set of languages recognizable by the difference of two languages in NP, *i.e.*, $\mathrm{DP} := \{L_1 \setminus L_2 \mid L_1, L_2 \in \mathrm{NP}\} = \{L_1 \cap L_2 \mid L_1 \in \mathrm{NP}, L_2 \in \mathrm{coNP}\}$, and Σ_2^{p} is the set of languages recognizable by nondeterministic polynomial-time Turing machines with an NP oracle. A complete problem for DP is CRITICAL-SAT, the problem to decide whether a given formula in 3CNF is unsatisfiable but removing any of its clauses makes it satisfiable [PW88]. For our hardness results we employ *logspace many-one reductions*, defined as follows: a language A is logspace many-one reducible to some language B (written $A \leq_{\mathrm{m}}^{\log} B$) if there exists a logspace-computable function f such that $x \in A$ if and only if $f(x) \in B$. For more background information on complexity theory, the reader is referred to [Pap94].

We assume familiarity with propositional logic. The set of all propositional formulae is denoted by $\mathcal{L}$. We use $\alpha, \varphi, \psi \ldots$ to denote formulae, and $\Delta, \Phi, \Psi, \ldots$ to denote sets of formulae. A *model* for a formula φ is a truth assignment to the set of its variables that satisfies φ. Further we denote by $\varphi[x/u]$ the formula obtained from φ by replacing all occurrences of x with u. For any formula $\varphi \in \mathcal{L}$, $\mathrm{Vars}(\varphi)$ denotes the set of variables occurring in φ (for $\Gamma \subseteq \mathcal{L}$, we use $\mathrm{Vars}(\Gamma) := \bigcup_{\gamma \in \Gamma} \mathrm{Vars}(\gamma)$), and we write $\Phi \models \varphi$ if Φ entails φ, *i.e.*, if every model of Φ also satisfies φ.

Throughout all the paper Δ is assumed to be a given finite set of formulae (the knowledge base) representing a large depositary of information, from which arguments can be constructed for arbitrary claims.

A *clone* is a set of Boolean functions that is closed under superposition, *i.e.*, it contains all projections (the functions $f(a_1, \ldots, a_n) = a_k$ for $1 \leq k \leq n$) and is closed under arbitrary composition. Let B be a finite set of Boolean functions. We denote by $[B]$ the smallest clone containing B and call B a *base* for $[B]$. The set of all clones was identified by Post [Pos41]. He gave a finite base for each clone and showed that they form a lattice with respect to subset inclusion, union and intersection; hence the name of *Post's lattice* (see Figure 1). In order to define the clones, we require the following notions, where f is an n-ary Boolean function:

- f is *c-reproducing* if $f(c, \ldots, c) = c$, $c \in \{0, 1\}$.
- f is *monotonic* if $a_1 \leq b_1, \ldots, a_n \leq b_n$ implies $f(a_1, \ldots, a_n) \leq f(b_1, \ldots, b_n)$.
- f is *c-separating of degree* k if for all $A \subseteq f^{-1}(c)$ of size $|A| = k$ there exist $i \in \{1, \ldots, n\}$ and $c \in \{0, 1\}$ such that $(a_1, \ldots, a_n) \in A$ implies $a_i = c$.
- f is *c-separating* if f is c-separating of degree $|f^{-1}(c)|$.
- f is *self-dual* if $f \equiv \neg f(\neg x_1, \ldots, \neg x_n)$.
- f is *affine* if $f \equiv x_1 \oplus \cdots \oplus x_n \oplus c$ with $c \in \{0, 1\}$.

A list of the relevant clones with definitions and finite bases is given in Table 1 on page 128, see [BCRV03] for a complete list. A propositional formula using only functions from B as connectives is called a *B-formula*. The set of all B-formulae is denoted by $\mathcal{L}(B)$. Let f be an n-ary Boolean function. A B-formula φ such that $\mathrm{Vars}(\varphi) = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ is a *B-representation* of f if for all $a_1, \ldots, a_n, b_1, \ldots, b_m \in \{0,1\}$ it holds that $f(a_1, \ldots, a_n) = 1$ if and only if every $\sigma\colon \mathrm{Vars}(\varphi) \to \{0,1\}$ with $\sigma(x_i) = a_i$ and $\sigma(y_i) = b_i$ for all relevant i, satisfies φ.

3 Argumentation

Definition 3.1. [BH01] *An* argument *is a pair (Φ, α), where Φ is a set of formulae and α is a formula such that*

1. *Φ is consistent,*
2. *$\Phi \models \alpha$,*
3. *Φ is minimal with this last property,* i.e., *no proper subset of Φ entails α.*

We say that (Φ, α) is an argument *for α. If $\Phi \subseteq \Delta$ then it is said to be an* argument in Δ. *We call α the* consequent *and Φ the* support *of the argument.*

Let B be a finite set of Boolean functions. Then the *argument existence problem for B-formulae* is defined as

Problem:	Arg(B).
Instance:	$\mathcal{A} = (\Delta, \alpha)$, where $\Delta \subseteq \mathcal{L}(B)$ and $\alpha \in \mathcal{L}(B)$.
Question:	Does there exist Φ such that (Φ, α) is an argument in Δ?

Besides the decision problem for the existence of an argument we are interested in the decision problems for B-formulae for validity, relevance and dispensability. They are defined as follows and deal with formulae in $\mathcal{L}(B)$ only.

Arg-Check(B): given a pair (Φ, α), is it an argument; Arg-Rel(B): given Δ, α, φ, is there an argument (Φ, α) such that $\varphi \in \Phi \subseteq \Delta$; and Arg-Disp($B$): given Δ, α, φ, is there an argument (Φ, α) such that $\varphi \notin \Phi \subseteq \Delta$.

Observe that the minimality of the support is only relevant for the problems Arg-Check and Arg-Rel. For Arg and Arg-Disp, the existence of a consistent subset Φ of the knowledge base Δ that entails the claim α (and does not contain some formula φ) implies a consistent $\Phi' \subseteq \Phi$ such that $\Phi' \models \alpha$ and $\Phi' \setminus \{\psi\} \not\models \alpha$ for all $\psi \in \Phi'$. To decide the existence of an argument, it therefore suffices to find any consistent subset of Δ that entails α. For Arg-Rel, on the other hand, we have to decide whether there exists an argument for α that contains the formula φ. The existence of some consistent set $\Phi \subseteq \Delta$ with $\varphi \in \Phi$ and $\Phi \models \alpha$ does not help here, because φ might be excluded from the minimal subset $\Phi' \subseteq \Phi$ yielding an argument for α. Consequently, unlike in other nonmonotonic reasoning formalisms, the complexity of deciding relevance and dispensability of a formula for some argument may differ. Indeed, we will show that there exist sets B such that Arg-Rel(B) is harder to decide than Arg-Disp(B) unless

the polynomial hierarchy collapses. Similarly, for Arg-Check, we have to verify that the set Φ in the given pair (Φ, α) is indeed minimal with respect to consistency and entailment of α. While this is supposedly easier to decide than Arg, we will see that owing to the verification of minimality there exist sets B such that Arg-Check(B) is harder to decide than Arg(B) unless the polynomial hierarchy collapses.

We conclude this section with two lemmas that make clear the role of the constant 1 in our study. They will be of use later on to establish our complexity classifications. Recall that the clone E_2 is defined in Table 1 on page 128.

Lemma 3.2. *Let* Arg-$\mathfrak{P}$ *denote any of the problems* Arg, Arg-Check *or* Arg-Rel. *Let B be a finite set of Boolean functions such that $\wedge \in [B]$,* i.e., *$\mathsf{E}_2 \subseteq [B]$. Then* Arg-$\mathfrak{P}(B \cup \{1\}) \leq_{\mathrm{m}}^{\log}$ Arg-$\mathfrak{P}(B)$.

Proof. Let $\mathcal{I}$ be the given instance. We map $\mathcal{I}$ to the instance $\mathcal{I}'$ obtained by replacing each formula ψ occurring in $\mathcal{I}$ by $\psi[1/t] \wedge t$. □

In addition on this, one can also eliminate the constant 1 for the problems Arg(B) and Arg-Rel(B) when $\mathsf{D}_2 \subseteq [B]$.

Lemma 3.3. *Let B be a finite set of Boolean functions such that $\mathsf{D}_2 \subseteq [B]$. Then* Arg$(B \cup \{1\}) \leq_{\mathrm{m}}^{\log}$ Arg(B) *and* Arg-Rel$(B \cup \{1\}) \leq_{\mathrm{m}}^{\log}$ Arg-Rel(B).

Proof. Let $g(x, y, z) := (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$. The function g is a base of D_2 and evaluates to true if and only if at least two of the variables are set to true. Given an instance (Δ, α) of Arg$(B \cup \{1\})$, we define an instance (Δ', α') of Arg(B) by $\Delta' := \{\psi[1/t] \mid \psi \in \Delta\} \cup \{t\}$ and $\alpha' = g(\alpha[1/t], t, q)$,where t and q are fresh variables. We claim that there is an argument for α in Δ if and only if there is an argument for α' in Δ'.

Let Φ be an argument for α in Δ. Consider $\Phi' := \{\psi[1/t] \mid \psi \in \Phi\} \cup \{t\}$. Observe that $\Phi' \equiv \Phi$. Thus Φ' is satisfiable and $\Phi' \models \alpha$, hence $\Phi' \models \alpha[1/t] \wedge t$, as t does not occur in α. Therefore, we obtain $\Phi' \models g(\alpha[1/t], t, q)$. Moreover, either Φ' or $\Phi' \setminus \{t\}$ is minimal with this property. Indeed, suppose that there exists a $\psi' \in \Phi'$ with $\psi' = \psi[1/t]$ for some $\psi \in \Phi$ such that $\Phi' \setminus \{\psi'\} \models g(\alpha[1/t], t, q)$. Then $\Phi' \setminus \{\psi'\} \models \alpha[1/t] \wedge t$ as q does not occur in Φ', and hence $\Phi \setminus \{\psi\} \models \alpha$, contradictory to the minimality of Φ.

Conversely, with similar arguments it is easy to see that if Φ' is an argument for α' in Δ', then $\Phi := \{\psi[t/1] \mid \psi \in \Phi', \psi \neq t\}$ is an argument for α in Δ: as q does not occur in Φ', $\Phi' \models \alpha'$ implies that $\Phi' \models \alpha[1/t] \wedge t$.

This proves correctness of the reduction from Arg$(B \cup \{1\})$ to Arg(B). The analogous result for Arg-Rel follows from the same arguments as above, mapping the additional component φ to $\varphi' := \varphi[1/t]$. □

Remark 3.4. Observe that this reduction does not work for Arg-Check: one would have to decide whether to map Φ to Φ' or to $\Phi' \setminus \{t\}$ to ensure minimality, which requires the ability to decide whether $\Phi' \setminus \{t\} \models t$ in Logspace.

4 The Complexity of Verification

We commence our study of the introduced argumentation problems with the argument verification problem. This problem is in DP. Indeed it is readily observed, as there are languages A, B with $A \in \mathrm{NP}$ and $B \in \mathrm{coNP}$ such that $\textsc{Arg-Check} = A \cap B$, with

$$A = \{(\Delta, \Phi, \alpha) \mid \Phi \text{ is satisfiable}, \forall \varphi \in \Phi : \Phi \setminus \{\varphi\} \not\models \alpha\};$$
$$B = \{(\Delta, \Phi, \alpha) \mid \Phi \models \alpha\}.$$

Proposition 4.1. *Let* $\mathsf{S}_{00} \subseteq [B]$. *Then* $\textsc{Arg-Check}(B)$ *is* DP*-complete.*

Proof. To prove DP-hardness we establish a reduction from Critical-Sat. Let $\psi = \bigwedge_{j=1}^{m} C_j$ be an instance of Critical-Sat, and $\mathrm{Vars}(\psi) = \{x_1, \ldots, x_n\}$. Let $u, x'_1, \ldots, x'_n$ be fresh, pairwise distinct variables. We may suppose without loss of generality that each x_i appears in ψ both as positive and as negative literal. Let further $C'_j := C_j[\neg x_i / x'_i \mid 1 \leq i \leq n]$ for $1 \leq j \leq m$ and $\psi' := \bigwedge_{j=1}^{m} C'_j$. We map ψ to (Φ, α), where we define

$$\Phi = \{C'_j \mid j \in \{1, \ldots, m\}\}, \text{ and } \alpha = \bigvee_{i=1}^{n} u \vee (x_i \wedge x'_i).$$

Since $x \vee y$ and $x \vee (y \wedge z)$ are functions of S_{00}, α and all C'_j's are S_{00}-formulae. These are by definition 1-reproducing. Therefore, Φ and α are satisfiable. For $1 \leq k \leq m$, let Φ_k, ψ_k, ψ'_k denote the respective set of clauses where we deleted the k^{th} clause. Note that always $\Phi \equiv \psi'$ and $\Phi_k \equiv \psi'_k$.

Suppose now that $\psi \in \textsc{Critical-Sat}$, *i.e.*, ψ is unsatisfiable and ψ_k is satisfiable for all $k \in \{1, \ldots, m\}$. We show that Φ entails α. Since $\psi \equiv \psi' \wedge \bigwedge_{i=1}^{n}(x_i \oplus x'_i)$ is unsatisfiable, and $\psi' \equiv \Phi$ is monotonic, all models of Φ have to set both x_i and x'_i to 1 for at least one $i \in \{1, \ldots, n\}$. Since $\alpha[u/0] \equiv \bigvee_{i=1}^{n}(x_i \wedge x'_i)$, we therefore have $\Phi \models \alpha[u/0]$. Obviously $\Phi \models \alpha[u/1]$.

It remains to prove that Φ is minimal. Since for each $k \in \{1, \ldots, m\}$ $\psi_k \equiv \psi'_k \wedge \bigwedge_{i=1}^{n}(x_i \oplus x'_i)$ is satisfiable, no $\psi'_k \equiv \Phi_k$ entails $\alpha[u/0] \equiv \bigvee_{i=1}^{n}(x_i \wedge x'_i)$. *A fortiori* no Φ_k entails α.

Conversely suppose that $(\Phi, \alpha) \in \textsc{Arg-Check}$. Then, in particular, Φ entails $\alpha[u/0]$. Thus we have $\psi' \models \bigvee_{i=1}^{n}(x_i \wedge x'_i)$, which implies that ψ is unsatisfiable. By the minimality of Φ we know that no Φ_k entails α. Since $\Phi_k \models \alpha[u/1]$, we conclude that $\Phi_k \not\models \alpha[u/0]$, which implies that $\psi'_k \wedge \bigwedge_{i=1}^{n}(\neg x_i \vee \neg x'_i)$ is satisfiable. As ψ'_k is monotonic, we obtain that also $\psi'_k \wedge \bigwedge_{i=1}^{n}(x_i \oplus x'_i)$ and hence ψ_k itself is satisfiable.

We finally transform (Φ, α) into a B-instance for all B such that $\mathsf{S}_{00} \subseteq [B]$ by replacing every connective by its B-representation. This transformation works in logarithmic space if we construct α as an $\vee$-tree of depth logarithmic in n. □

Proposition 4.2. *Let* B *be a finite set of Boolean functions such that* $\mathsf{D}_2 \subseteq [B]$. *Then* $\textsc{Arg-Check}(B)$ *is* DP*-complete.*

Proof. We give a reduction from CRITICAL-SAT similar to Proposition 4.1. For $k \in \mathbb{N}$, we define g_k as a $(k+1)$-ary function satisfying $g_k(z_1, \ldots, z_k, 0) \equiv \bigwedge_{i=1}^{k} z_i$ and $g_k(z_1, \ldots, z_k, 1) \equiv \bigvee_{i=1}^{k} z_i$. Note that for every $k \in \mathbb{N}$, g_k is monotonic and self-dual, and thus contained in D_2. By abuse of notation, given a clause $C = (l^1 \vee l^2 \vee l^3)$ and a variable x, $g_3(C, x)$ stands for $g_3(l^1, l^2, l^3, x)$. Let $\psi = \bigwedge_{j=1}^{m} C_j$ be an instance of CRITICAL-SAT with $C_j = (l_j^1 \vee l_j^2 \vee l_j^3)$ and $\mathrm{Vars}(\psi) = \{x_1, \ldots, x_n\}$. Let further $u, v, x_1', \ldots, x_n'$ be fresh, pairwise distinct variables and $C_j' := C_j[\neg x_i / x_i' \mid 1 \leq i \leq n]$ for $1 \leq j \leq m$. We may suppose without loss of generality that each x_i appears in ψ both as a positive and as a negative literal.

We map ψ to (Φ, α), where

$$\Phi := \{g_3(C_j', u) \mid 1 \leq j \leq m\}, \text{ and } \alpha := g_n\big((g_2(x_i, x_i', v))_{1 \leq i \leq n}, u\big).$$

Obviously α and the formulae in Φ are D_2-formulae and thus satisfiable. As in the proof of the previous proposition a careful examination allows to prove that $\psi \in$ CRITICAL-SAT if and only if $(\Phi, \alpha) \in$ ARG-CHECK.

Finally, we transform (Φ, α) into a B-instance for all B such that $\mathsf{D}_2 \subseteq [B]$ in replacing all occurrences of g_k by its B-representation. This transformation works in logarithmic space, because we may assume the function g_n to be a g_2-tree of depth logarithmic in n. □

Theorem 4.3. *Let B be a finite set of Boolean functions. Then the argument validity problem for propositional B-formulae,* ARG-CHECK(B)*, is*

1. DP*-complete if* $\mathsf{S}_{00} \subseteq [B]$ *or* $\mathsf{S}_{10} \subseteq [B]$ *or* $\mathsf{D}_2 \subseteq [B]$,
2. *in* P *if* $\mathsf{L}_2 \subseteq [B] \subseteq \mathsf{L}$,
3. *in* LOGSPACE *if* $[B] \subseteq \mathsf{V}$ *or* $[B] \subseteq \mathsf{E}$ *or* $[B] \subseteq \mathsf{N}$.

Proof. For DP-completeness, according to Propositions 4.1 and 4.2 it remains only to deal with the case $\mathsf{S}_{10} \subseteq [B]$. Since $\mathsf{D}_2 \subseteq \mathsf{M}_1 = [\mathsf{S}_{10} \cup \{1\}] \subseteq [B \cup \{1\}]$, we obtain that ARG-CHECK$(B \cup \{1\})$ is DP-hard by Proposition 4.2. As $\wedge \in [B]$, we may apply Lemma 3.2 and obtain the DP-hardness of ARG-CHECK(B).

For testing whether (Φ, α) is an argument we need to check the following three conditions:

(1) Φ is satisfiable,
(2) $\Phi \wedge \neg\alpha$ is unsatisfiable (*i.e.*, $\Phi \models \alpha$), and
(3) for all $\varphi \in \Phi$, $(\Phi \setminus \{\varphi\}) \cup \{\neg\alpha\}$ is satisfiable (*i.e.*, Φ is minimal).

In the case $\mathsf{L}_2 \subseteq [B] \subseteq \mathsf{L}$ the sets Φ, $\Phi \cup \{\neg\alpha\}$, and $(\Phi \setminus \{\varphi\}) \cup \{\neg\alpha\}$ for all $\varphi \in \Phi$ can be easily transformed into systems of linear equations. Thus checking the three conditions comes down to solving a polynomial number of systems of linear equations. This can be done in polynomial time using Gaussian elimination. For $[B] \subseteq \mathsf{V}$, for $[B] \subseteq \mathsf{E}$, and for $[B] \subseteq \mathsf{N}$ this check can be done in logarithmic space, as in this case the satisfiability of sets of B-formulae can be determined in logarithmic space. □

5 The Complexity of Existence and Dispensability

Theorem 5.1. *Let B be a finite set of Boolean functions. Then the argument existence problem for propositional B-formulae,* Arg(B)*, is*

1. Σ_2^{p}*-complete if* $\mathsf{D} \subseteq [B]$ *or* $\mathsf{S}_1 \subseteq [B]$,
2. coNP*-complete if* $\mathcal{X} \subseteq [B] \subseteq \mathcal{Y}$ *with* $\mathcal{X} \in \{\mathsf{S}_{00}, \mathsf{S}_{10}, \mathsf{D}_2\}$ *and* $\mathcal{Y} \in \{\mathsf{M}, \mathsf{R}_1\}$,
3. *in* NP *if* $[B] \in \{\mathsf{L}, \mathsf{L}_0, \mathsf{L}_3\}$,
4. *in* P *if* $[B] \in \{\mathsf{L}_1, \mathsf{L}_2\}$, *and*
5. *in* Logspace *if* $[B] \subseteq \mathsf{V}$ *or* $[B] \subseteq \mathsf{E}$ *or* $[B] \subseteq \mathsf{N}$.

The same classification holds for Arg-Disp(B).

Proof. The general argumentation problem has been shown to be Σ_2^{p}-complete in [PWA03] via a reduction from $\mathrm{QSAT}_{2,\exists}$. Starting from formulae in 3DNF, we can use the reduction from [PWA03] and insert parentheses to obtain formulae of logarithmic depth only. We can now substitute the connectives $\wedge$, $\vee$, $\neg$ with their B-representations to obtain Σ_2^{p}-completeness for Arg(B) if $[B] = \mathsf{BF}$.

As $\mathsf{E}_2 \subseteq \mathsf{S}_1$ and $[\mathsf{S}_1 \cup \{1\}] = \mathsf{BF}$, we obtain Σ_2^{p}-completeness for the case $\mathsf{S}_1 \subseteq [B]$ according to Lemma 3.2. For the case $\mathsf{D} \subseteq [B]$, we obtain Σ_2^{p}-completeness by Lemma 3.3, since $\mathsf{D}_2 \subseteq \mathsf{D}$ and $[\mathsf{D} \cup \{1\}] = \mathsf{BF}$.

For $\mathcal{X} \subseteq [B] \subseteq \mathcal{Y}$ with $\mathcal{X} \in \{\mathsf{S}_{00}, \mathsf{S}_{10}, \mathsf{D}_2\}$ and $\mathcal{Y} \in \{\mathsf{M}, \mathsf{R}_1\}$, membership in coNP follows from the facts that satisfiability is in Logspace [Lew79], while entailment is in coNP [BMTV09]. To prove the coNP-hardness of Arg(B), we give a reduction from the implication problem for B-formulae, which is coNP-hard if $[B]$ contains one of the clones S_{00}, S_{10}, D_2. Let (ψ, α) be a pair of B-formulae. We map this instance to $(\{\psi\}, \alpha)$ if ψ is satisfiable and to a trivial positive instance otherwise.

For $[B] \in \{\mathsf{L}, \mathsf{L}_0, \mathsf{L}_3\}$, membership in NP follows from the fact that in this case Arg-Check is in P. Due to the trivial satisfiability of B-formulae for $[B] \in \{\mathsf{L}_1, \mathsf{L}_2\}$, we can improve the upper bound for Arg(B) with $[B] \in \{\mathsf{L}_1, \mathsf{L}_2\}$ to membership in P.

In all other cases, Logspace-membership follows from the fact that the satisfiability and entailment problem for B-formulae are contained in Logspace (see [BMTV09]).

Finally, observe that we have Arg-Disp(B) $\equiv_{\mathrm{m}}^{\log}$ Arg(B). To prove that Arg(B) $\leq_{\mathrm{m}}^{\log}$ Arg-Disp(B), map $\mathcal{A} = (\Delta, \alpha)$ to $\mathcal{D} := (\Delta \cup \{t\}, \alpha, t)$. For the converse direction, map $\mathcal{D} = (\Delta, \alpha, \varphi)$ to $\mathcal{A} := (\Delta \setminus \{\varphi\}, \alpha)$. □

6 The Complexity of Relevance

Proposition 6.1. *Let B be a finite set of Boolean functions such that* $\mathsf{S}_{00} \subseteq [B]$. *Then* Arg-Rel($B$) *is* Σ_2^{p}*-complete.*

Proof. To see that Arg-Rel(B) is contained in Σ_2^{p}, observe that, given an instance $(\Delta, \alpha, \varphi)$, we can guess a set $\Phi \subseteq \Delta$ such that $\varphi \in \Phi$ and verify conditions (1)–(3) as given in the proof of Theorem 4.3 in polynomial time using an NP-oracle.

To prove Σ_2^{p}-hardness, we provide a reduction from the problem $\mathrm{QSAT}_{2,\exists}$. An instance of this problem is a quantified formula $\exists X \forall Y \beta$ where $\beta = \bigvee_{j=1}^{p} t_j$ with exactly three literals by term. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. We transform $\exists X \forall Y \beta$ to $(\Delta, \alpha, \varphi)$, where

$$\Delta := \{x_i, x_i' \mid 1 \leq i \leq n\} \cup \{v \wedge \textstyle\bigwedge_{i=1}^{m}(y_i \vee y_i')\} \cup \{u\},$$
$$\alpha := \beta' \wedge v \wedge \big(\textstyle\bigvee_{i=1}^{n}(x_i \wedge x_i') \vee u\big), \text{ and } \varphi := u,$$

and where $\beta' = \bigvee_{j=1}^{p} t_j'$, $t_j' := t_j[\neg x_1/x_1', \ldots, \neg x_n/x_n', \neg y_1/y_1', \ldots, \neg y_m/y_m']$ for all $1 \leq j \leq p$, and u, v are fresh variables.

We show that $\exists X \forall Y \beta$ is valid if and only if $(\Delta, \alpha, \varphi) \in \textsc{Arg-Rel}(\{\wedge, \vee\})$. If $\exists X \forall Y \beta$ is valid, then there exists an assignment $\sigma\colon X \to \{0,1\}$ such that $\sigma \models \beta$. Consequently, for $\Phi := \{x_i \mid \sigma(x_i) = 1\} \cup \{x_i' \mid \sigma(x_i) = 0\} \cup \{u, v \wedge \bigwedge_{i=1}^{m}(y_i \vee y_i')\}$, we obtain $\Phi \models \beta$. As Φ is consistent, it thus remains to show that u is relevant, *i.e.*, that $\Phi \setminus \{u\} \not\models \alpha$. This follows from the fact that $\Phi \setminus \{\varphi\}$ is satisfied by the assignment σ' obtained from σ by setting $\sigma'(u) := 0$, while $\sigma' \not\models \bigvee_{i=1}^{n}(x_i \wedge x_i') \vee u$ and hence $\sigma' \not\models \alpha$.

For the converse direction, let Φ be a support for α such that $u \in \Phi$. Since $\Phi \models \alpha$ we conclude that $v \wedge \bigwedge_{i=1}^{m}(y_i \vee y_i') \in \Phi$ and hence $\Phi = \mathcal{X} \cup \{v \wedge \bigwedge_{i=1}^{m}(y_i \vee y_i')\} \cup \{u\}$, for some $\mathcal{X} \subseteq \{x_i, x_i' \mid 1 \leq i \leq n\}$. From $\Phi \models \alpha$ also follows that $\Phi \models \beta'$. From the minimality of Φ we conclude that in particular $\Phi \setminus \{u\} \not\models \alpha$. And therefore $\Phi \not\models \bigvee_{i=1}^{n}(x_i \wedge x_i')$. That is $\Phi \wedge \bigwedge_{i=1}^{n}(\neg x_i \vee \neg x_i')$ is satisfiable and since Φ is monotonic, consequently also $\Phi \wedge \bigwedge_{i=1}^{n}(x_i \oplus x_i')$ is satisfiable. Summed up, we know that $\gamma := \mathcal{X} \wedge \bigwedge_{i=1}^{n}(x_i \oplus x_i') \wedge \bigwedge_{i=1}^{m}(y_i \vee y_i')$ is satisfiable and $\gamma \models \beta'$. Hence, *a fortiori*, $\gamma' := \mathcal{X} \wedge \bigwedge_{i=1}^{n}(x_i \oplus x_i') \wedge \bigwedge_{i=1}^{m}(y_i \oplus y_i')$ is satisfiable and $\gamma' \models \beta'$. Define now $\sigma_X(x_i) = 1$ if $x_i \in \mathcal{X}$, $\sigma_X(x_i) = 0$ otherwise. Obviously any extension of σ_X to Y satisfies β and therefore $\exists X \forall Y \beta$ is valid.

It remains to transform $(\Delta, \alpha, \varphi)$ into an $\textsc{Arg-Rel}(B)$-instance for all B such that $\mathsf{S}_{00} \subseteq [B]$. As both $\wedge$ and $\vee$ are associative, we can insert parentheses into $(\Delta, \alpha, \varphi)$ such that we can represent each formula as binary $\{\wedge, \vee\}$-tree of logarithmic depth. Let f be a fresh variable and let h be the boolean function in S_{00} defined by $h(f, x, y) \equiv f \vee (x \wedge y)$. We further transform our instance into $(\Delta', \alpha' \vee f, \varphi')$, where $\Delta', \alpha', \varphi'$ are obtained by replacing each occurrence of $x \wedge y$ by $h(f, x, y)$. One easily verifies that $(\Delta', \alpha' \vee f, \varphi')$ is in $\textsc{Arg-Rel}(\{\vee, h\})$ if and only if $(\Delta, \alpha, \varphi) \in \textsc{Arg-Rel}(\{\wedge, \vee\})$. We finally replace $\vee$ and h by their B-representation. □

Proposition 6.2. *Let B be a finite set of Boolean functions such that $[B] \subseteq \mathsf{V}$ or $[B] \subseteq \mathsf{E}$ or $[B] \subseteq \mathsf{N}$. Then $\textsc{Arg-Rel}(B)$ is in* Logspace.

Proof. We assume the representation of V-, E-, or N-formulae as respectively positive clauses, positive terms, or literals. Let us first consider $\textsc{Arg-Rel}(B)$ for $[B] \subseteq \mathsf{E}$. It is easy to observe that a set of positive terms Δ entails a positive term α if and only if $\mathrm{Vars}(\alpha) \subseteq \mathrm{Vars}(\Delta)$. We claim that Algorithm 1 decides $\textsc{Arg-Rel}(B)$.

Algorithm 1 can be implemented using only a logarithmic amount of space if we do not construct Δ_x entirely but rather check the condition in line 3 directly: $\Delta_x \models \alpha$ holds if and only if $\mathrm{Vars}(\alpha) \subseteq \mathrm{Vars}(\varphi) \cup \mathrm{Vars}(\{\tau \in \Delta \mid x \notin \mathrm{Vars}(\tau)\})$.

Algorithm 1. Algorithm for Arg-Rel(B) with $[B] \subseteq \mathsf{E}$

Require: a set Δ of positive terms and positive terms α, φ with $\varphi \in \Delta$.
1: **for all** $x \in \mathrm{Vars}(\varphi)$ **do**
2: $\quad \Delta_x := \{\varphi\} \cup \{\tau \in \Delta \mid x \notin \mathrm{Vars}(\tau)\}$
3: $\quad$ **if** $\Delta_x \models \alpha$ **then**
4: $\quad\quad$ accept
5: $\quad$ **end if**
6: **end for**
7: reject

To prove correctness, notice that Algorithm 1 accepts only if there exists a $\Delta_x \subseteq \Delta$ such that $\Delta_x \models \alpha$ and $\Delta_x \setminus \{\varphi\} \not\models \alpha$. Thus Δ_x contains a support Φ such that $\varphi \in \Phi$. Conversely, let Φ be a support such that $\varphi \in \Phi$. Since $\Phi \models \alpha$ and $\Phi \setminus \{\varphi\} \not\models \alpha$, there is at least one $x_i \in (\mathrm{Vars}(\varphi) \cap \mathrm{Vars}(\alpha)) \setminus \mathrm{Vars}(\Phi)$. For this x_i the algorithm constructs $\Delta_{x_i} := \{\varphi\} \cup \{\tau \in \Delta \mid x_i \notin \mathrm{Vars}(\tau)\}$. Obviously $\Phi \subseteq \Delta_{x_i}$ and therefore $\Delta_{x_i} \models \alpha$ which causes the algorithm to accept.

Next, consider Arg-Rel(B) for $[B] \subseteq \mathsf{V}$. Observe that a set of positive clauses C entails a positive clause α if and only if there is a clause $c \in C$ such that $\mathrm{Vars}(c) \subseteq \mathrm{Vars}(\alpha)$. Thus if there is a support Φ with $\varphi \in \Phi$ then it is the singleton $\{\varphi\}$. Given $(\Delta, \alpha, \varphi)$ as an instance of Arg-Rel(V), it hence suffices to check whether $\mathrm{Vars}(\varphi) \subseteq \mathrm{Vars}(\alpha)$, which can be done in Logspace.

Finally Arg-Rel(B) for $[B] \subseteq \mathsf{N}$ is in Logspace, since each B-formula can be transformed into a single literal. □

From the two propositions above, Lemma 3.2 and Lemma 3.3 we obtain the following complexity classification for Arg-Rel.

Theorem 6.3. *Let B be a finite set of Boolean functions. Then the argument relevance problem for propositional B-formulae,* Arg-Rel(B)*, is*

1. Σ_2^{p}*-complete if* $\mathsf{S}_{00} \subseteq [B]$ *or* $\mathsf{D}_2 \subseteq [B]$ *or* $\mathsf{S}_{10} \subseteq [B]$,
2. *in* NP *if* $\mathsf{L}_2 \subseteq [B] \subseteq \mathsf{L}$,
3. *in* Logspace *if* $[B] \subseteq \mathsf{V}$ *or* $[B] \subseteq \mathsf{E}$ *or* $[B] \subseteq \mathsf{N}$.

7 Discussion and Conclusion

Complexity classifications along the lines of Boolean clones have already been carried out for AI formalisms as circumscription [Tho09] and abduction [CST10]. In particular, the latter work is closely related to the contents of this paper. To make this more precise, let us consider the positive abduction problem P-Abd(B) which takes as an instance a triple (Γ, H, m), where $\Gamma \subseteq \mathcal{L}(B)$, $m \in \mathcal{L}(B)$, H is a set of variables, and asks whether there exists an explanation $E \subseteq H$ such that $\Gamma \wedge E$ is satisfiable and $\Gamma \wedge E \models m$. Hence, the main difference to argumentation is the presence of the knowledge base Γ in the tests for consistency and entailment. Nonetheless, the following relations hold: (1) if $\wedge \in [B]$, *i.e.*, if $\mathsf{E}_2 \subseteq [B]$, then P-Abd($B$) $\leq_{\mathrm{m}}^{\log}$ Arg(B); (2) if $\rightarrow \in [B]$, *i.e.*, if $\mathsf{S}_0 \subseteq [B]$, then Arg($B$) $\leq_{\mathrm{m}}^{\log}$ P-Abd(B).

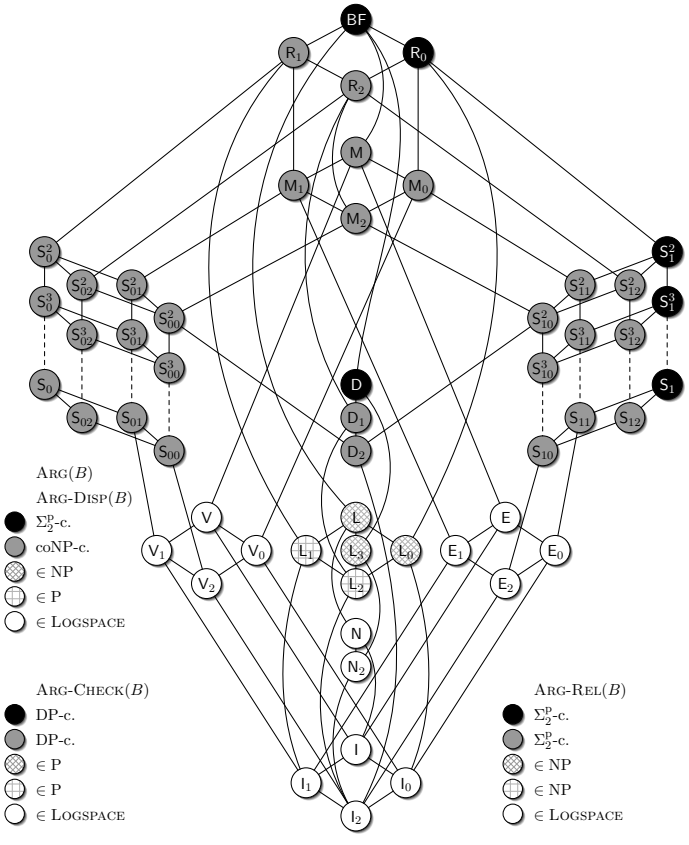


Fig. 1. Post's lattice showing the complexity of the argumentation problems studied herein

In fact, it turns out that ARG and ARG-DISP have the same complexity classification as positive abduction. This is due to the fact that minimality of the argument plays no role in ARG and ARG-DISP. However, for ARG-REL the situation is different but we expect similarly harder complexity for the relevance problem in abduction with respect to subset-minimal explanations (see, e.g., [EG95] for the definitions) which has not been analyzed in [CST10]. In other words, the results provided in the present paper can be used to obtain novel results for certain variants of abduction, which have not been classified yet.

To summarize, we took in this paper first steps to understanding the complexity of logic-based argumentation by providing a classification of the complexity of four important tasks for all possible restrictions on the set of allowed connectives. The results are collected in Figure 1. Notably are the sets B of Boolean connectives where $\mathcal{X} \subseteq [B] \subseteq \mathcal{Y}$ with $\mathcal{X} \in \{\mathsf{S}_{00}, \mathsf{S}_{10}, \mathsf{D}_2\}$ and $\mathcal{Y} \in \{\mathsf{M}, \mathsf{R}_1\}$ which give coNP-completeness for ARG(B), while ARG-REL(B) remains complete for Σ_2^{p} (typically this applies to monotonic formulae in which no negation is involved). As well, ARG(B) with $\mathsf{L}_2 \subseteq [B] \subseteq \mathsf{L}_1$ is in P, while for the

Table 1. List of some Boolean clones with definitions and bases.

Name	Definition	Base
BF	All Boolean functions	$\{x \wedge y, \neg x\}$
R_0	$\{f \mid f$ is 0-reproducing$\}$	$\{x \wedge y, x \oplus y\}$
R_1	$\{f \mid f$ is 1-reproducing$\}$	$\{x \vee y, x \leftrightarrow y\}$
R_2	$R_0 \cap R_1$	$\{\vee, x \wedge (y \leftrightarrow z)\}$
M	$\{f \mid f$ is monotonic$\}$	$\{x \vee y, x \wedge y, 0, 1\}$
S_0	$\{f \mid f$ is 0-separating$\}$	$\{x \rightarrow y\}$
S_1	$\{f \mid f$ is 1-separating$\}$	$\{x \wedge \neg y\}$
S_{00}	$S_0 \cap R_2 \cap M$	$\{x \vee (y \wedge z)\}$
S_{10}	$S_1 \cap R_2 \cap M$	$\{x \wedge (y \vee z)\}$
D	$\{f \mid f$ is self-dual$\}$	$\{(x \wedge \neg y) \vee (x \wedge \neg z) \vee (\neg y \wedge \neg z)\}$
D_2	$D \cap M$	$\{(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)\}$
L	$\{f \mid f$ is affine$\}$	$\{x \oplus y, 1\}$
L_0	$L \cap R_0$	$\{x \oplus y\}$
L_1	$L \cap R_1$	$\{x \leftrightarrow y\}$
L_2	$L \cap R_2$	$\{x \oplus y \oplus z\}$
L_3	$L \cap D$	$\{x \oplus y \oplus z \oplus 1\}$
V	$\{f \mid f$ is a disjunction of variables or constant$\}$	$\{x \vee y, 0, 1\}$
V_2	$V \cap R_2$	$\{x \vee y\}$
E	$\{f \mid f$ is a conjunction of variables or constant$\}$	$\{x \wedge y, 0, 1\}$
E_2	$E \cap R_2$	$\{x \wedge y\}$
N	$\{f \mid f$ depends on at most one variable$\}$	$\{\neg x, 0, 1\}$
I	$\{f \mid f$ is a projection or a constant$\}$	$\{\mathrm{id}, 0, 1\}$
I_2	$I \cap R_2$	$\{\mathrm{id}\}$

corresponding problems Arg-Rel(B), we only have an NP upper-bound, so far. In fact, the exact classification of the problems into tractable and intractable cases remains open for affine sets of Boolean connectives in the following cases: Arg(B) with $[B] \in \{L, L_0, L_3\}$ and Arg-Rel(B) with $L_2 \subseteq [B] \subseteq L$.[1]

The complexity of Arg-Rel is a computational core for evaluating more complex argumentation problems, for instance, the warranted formula problem (WFP) on argument trees, which has recently been shown to be PSPACE-complete [HG10]. We expect that fragments studied here also lower the complexity of WFP, but leave details for future work.

Further future work concerns studying the complexity of all these problems in the popular Schaefer's framework (in which formulas are in generalized conjunctive normal form), as well as addressing more advanced problems of logic-based argumentation which are defined, *e.g.*, over argument-trees.

References

[AC02] Amgoud, L., Cayrol, C.: A model of reasoning based on the production of acceptable arguments. Ann. Math. Artif. Intell. 34, 197–216 (2002)

[BCRV03] Böhler, E., Creignou, N., Reith, S., Vollmer, H.: Playing with Boolean blocks I: Post's lattice with applications to complexity theory. SIGACT News 34(4), 38–52 (2003)

[1] We note that the complexity of the corresponding fragments remained unclassified also for circumscription and positive abduction.

[BD07] Bench-Capon, T., Dunne, P.: Argumentation in artificial intelligence. Artif. Intell. 171(10-15), 619–641 (2007)

[BH01] Besnard, P., Hunter, A.: A logic-based theory of deductive arguments. Artif. Intell. 128, 203–235 (2001)

[BH08] Besnard, P., Hunter, A.: Elements of Argumentation. MIT Press, Cambridge (2008)

[BMTV09] Beyersdorff, O., Meier, A., Thomas, M., Vollmer, H.: The complexity of propositional implication. Inf. Process. Lett. 109(18), 1071–1077 (2009)

[CML00] Chesñevar, C., Maguitman, A., Loui, R.: Logical models of argument. ACM Comput. Surv. 32, 337–383 (2000)

[CST10] Creignou, N., Schmidt, J., Thomas, M.: Complexity of propositional abduction for restricted sets of Boolean functions. In: Proc. 12th KR, pp. 8–16. AAAI, Menlo Park (2010)

[DKT06] Dung, P., Kowalski, R., Toni, F.: Dialectical proof procedures for assumption-based admissible argumentation. Artif. Intell. 170, 114–159 (2006)

[Dun95] Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995)

[EG95] Eiter, T., Gottlob, G.: The complexity of logic-based abduction. J. ACM 42(1), 3–42 (1995)

[GS04] García, A., Simari, G.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming 4(1), 95–138 (2004)

[HG10] Hirsch, R., Gorogiannis, N.: The complexity of the warranted formula problem in propositional argumentation. J. Log. Comput. 20, 481–499 (2010)

[Lew79] Lewis, H.: Satisfiability problems for propositional calculi. Mathematical Systems Theory 13, 45–53 (1979)

[Pap94] Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)

[Pos41] Post, E.: The two-valued iterative systems of mathematical logic. Ann. Math. Stud. 5, 1–122 (1941)

[PV02] Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In: Gabbay, D. (ed.) Handbook of Philosophical Logic. Kluwer, Dordrecht (2002)

[PW88] Papadimitriou, C., Wolfe, D.: The complexity of facets resolved. J. Comput. Syst. Sci. 37(1), 2–13 (1988)

[PWA03] Parsons, S., Wooldridge, M., Amgoud, L.: Properties and complexity of some formal inter-agent dialogues. J. Log. Comput. 13(3), 347–376 (2003)

[RS09] Rahwan, I., Simari, G. (eds.): Argumentation in Artificial Intelligence. Springer, Heidelberg (2009)

[Tho09] Thomas, M.: The complexity of circumscriptive inference in Post's lattice. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 290–302. Springer, Heidelberg (2009)

Retroactive Subsumption-Based Tabled Evaluation of Logic Programs

Flávio Cruz and Ricardo Rocha*

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
{flavioc,ricroc}@dcc.fc.up.pt

Abstract. Tabled evaluation is a recognized and powerful implementation technique that overcomes some limitations of traditional Prolog systems in dealing with recursion and redundant sub-computations. Tabling based systems use call similarity to determine if a tabled subgoal will produce their own answers or if it will consume from another subgoal. While call variance has been a very popular approach, call subsumption can yield superior time performance and space improvements as it allows greater reuse of answers. However, the call order of the subgoals can greatly affect the success and applicability of the call subsumption technique. In this work, we present an extension, named *Retroactive Call Subsumption*, that supports call subsumption by allowing full sharing of answers between subsumed/subsuming subgoals, independently on the order in which they are called. Our experiments using the YapTab tabling engine show considerable gains in evaluation time for some applications, at the expense of a very small overhead for the programs that cannot benefit from it.

Keywords: Tabled Evaluation, Call Subsumption, Implementation.

1 Introduction

Tabling [1] is an implementation technique that solves some of the shortcomings of Prolog systems based on the traditional SLD resolution method. Tabled resolution methods can considerably reduce the search space, avoid looping and have better termination properties than SLD resolution based methods. The tabling technique is a refinement of SLD resolution that stems from one simple idea: save intermediate answers from past computations so that they can be reused when a *similar call* appears during the resolution process. In a nutshell, first calls to tabled subgoals are evaluated as usual, using SLD resolution, but their answers are stored in a global data space, called the *table space*. Similar calls to tabled subgoals are then resolved by consuming the answers already stored in the table entry for the corresponding similar subgoal, instead of being re-evaluated against the program clauses. Call similarity thus determines if a subgoal will

* This work has been partially supported by the FCT research projects STAMPA (PTDC/EIA/67738/2006) and HORUS (PTDC/EIA-EIA/100897/2008).

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 130–142, 2010.
© Springer-Verlag Berlin Heidelberg 2010

produce their own answers or if it will consume from another subgoal. In general, we can distinguish two main approaches to determine similarity between tabled subgoals: *variant-based tabling* and *subsumption-based tabling*.

In variant-based tabling, two subgoals are considered to be similar if they are the same by renaming the variables. For example, subgoals $p(X, 1, Y)$ and $p(Y, 1, Z)$ are *variants* because both can be made identical to $p(VAR_0, 1, VAR_1)$ through variable renaming. In subsumption-based tabling, two subgoals are considered to be similar if one subgoal subsumes the other. A subgoal R is *subsumed* by a subgoal S (or a subgoal S *subsumes* a subgoal R) if R is more specific than S (or S is more general then R). For example, subgoal $p(X, 1, 2)$ is subsumed by subgoal $p(Y, 1, Z)$ because there is a substitution $\{Y = X, Z = 2\}$ that makes $p(X, 1, 2)$ an instance of $p(Y, 1, Z)$. Notice that, if R is subsumed by S, S will contain in its table entry the full set of answers that also satisfy R and thus, R can reuse and consume answers directly from S.

In general, subsumption-based tabling can yield superior time performance, as it allows greater reuse of answers, and better space usage, since the answer sets for the subsumed subgoals are not stored. However, the mechanisms to efficiently support subsumption-based tabling are more complex and hard to implement, which makes variant-based tabling more popular within the available tabling systems. To the best of our knowledge, until now, XSB Prolog was the unique system supporting subsumption-based tabling. The first implemented design was called *Dynamic Threaded Sequential Automata (DTSA)* [2], but was later replaced by an alternative table space organization called *Time-Stamped Trie (TST)* [3], that showed better space efficiency than DTSA.

Despite the good results and performance gains obtained with the DTSA and TST designs, both approaches suffer from a major problem: the order in which subgoals are called during a particular evaluation can greatly affect the success and applicability of the call subsumption technique. For example, consider again the subgoals $p(X, 1, 2)$ and $p(Y, 1, Z)$. If $p(X, 1, 2)$ is called after $p(Y, 1, Z)$, then $p(X, 1, 2)$ will be able to consume answers from $p(Y, 1, Z)$ tables. Otherwise, if $p(X, 1, 2)$ is called before, then no sharing will be possible, since answer reuse only happens when more general subgoals appear before specific ones.

In this work, we present an extension to the original TST design, named *Retroactive Call Subsumption (RCS)*, that supports call subsumption by allowing full sharing of answers between subsumed/subsuming subgoals, independently on the order in which they are called. The RCS proposal implements a strategy that allows retroactive sharing of answers among subgoals by means of selectively pruning and restarting the evaluation of subsumed subgoals. In order to support this, we introduce the following main extensions for subsumption-based tabling: (i) a new algorithm to efficiently traverse the table space searching for subsumed subgoals; (ii) a new table space organization, based on the ideas of the *common global trie* proposal [4], where answers are represented only once; and (iii) a new evaluation strategy to selectively prune and restart the evaluation of tabled nodes. We will focus our discussion on a concrete implementation, the YapTab system [5], but our proposals can be generalized and applied to other tabling

systems. Notice that, in order to do this work, first we have ported, from XSB Prolog to YapTab, the full code that implements the TST design[1].

The remainder of the paper is organized as follows. First, we briefly introduce the main background concepts about tabled evaluation in YapTab. Next, we present the RCS proposal and discuss the main operational challenges involved in its design. We then describe how we have extended YapTab to provide engine support for it. Finally, we present some experimental results and conclusions.

2 Tabled Evaluation in YapTab

Tabling consists of storing intermediate answers for subgoals so that they can be reused when a similar subgoal appears. Whenever a tabled subgoal is first called, a new entry is allocated in the table space. Table entries are used to keep track of subgoal calls and to store their answers. Each time a tabled subgoal is called, we know if it is a repeated call by inspecting the table space searching for a similar call. Within this model, the nodes in the search space are classified as either: *generator nodes*, if they are being called for the first time; *consumer nodes*, if they are repeated calls; or *interior nodes*, if they are non-tabled subgoals.

To support tabled evaluation, the YapTab design [5] extends the *Warren's Abstract Machine (WAM)* [6] execution model with the following four operations:

Tabled Subgoal Call: this operation searches the table space looking for a subgoal S similar to the current subgoal C being called. If such subgoal is found, C will be resolved using *answer resolution* and for that it allocates a consumer node and starts consuming the set of available answers from S. If not, C will be resolved using program clause resolution and for that it allocates a generator node, adds a new entry to the table space and initializes it with an empty set A_C of answers.

New Answer: this operation checks whether a newly found answer a for a generator node C is already in its table entry. If a is a repeated answer, the operation fails. Otherwise, a new answer set $A'_C = A_C \cup a$ is generated.

Answer Resolution: this operation checks whether a consumer node C has new answers available for consumption. If no answers are available, C is *suspended* and execution proceeds using a specific strategy [7]. Consumers must suspend because new answers may still be found by the corresponding variant/subsuming call S. Otherwise, given C's last consumed answer, we determine the unconsumed answer set $U_C \subseteq A_S$ and consume the next one.

Completion: this operation determines whether a subgoal S is completely evaluated. If this is not the case, this means that there are still consumers with unconsumed answers and execution thus proceeds to one of such consumers. Otherwise, the operation closes S's table entry, meaning that the full set of answers A_S was found, and future variant/subsumed calls to S can then reuse A_S without the need to suspend.

[1] We would like to thank Terrance Swift for his help in introducing us the XSB Prolog code that implements the TST design.

In YapTab, tabled nodes are implemented as WAM choice points extended with some extra fields. Furthermore, YapTab associates a data structure, named *subgoal frame*, to generator nodes and another, named *dependency frame*, to consumer nodes. Each subgoal frame stores information about the subgoal, namely the entry point to its answer set. Each dependency frame stores information about the consumer node, namely information for detecting completion. Both sets of subgoal and dependency frames are connected in creation time order forming a doubly-linked list.

3 Retroactive Call Subsumption

In this section, we present the RCS proposal in more detail, focusing the discussion on the new evaluation strategy that allows retroactive sharing of answers among subgoals by means of selectively pruning and restarting the evaluation of subsumed subgoals. Pruning the evaluation of a subsumed subgoal R requires knowing the parts of the execution stacks and respective choice points involved in its computation, and then transforming R's generator choice point in such a way that it can consume the answers from the subsuming subgoal S, instead of continuing its normal execution. We argue that there are two main types of pruning situations from where any other situation can be derived. The first type is *external pruning* and occurs when the subsuming subgoal S is an *external subgoal* to the evaluation of R. The second type is *internal pruning* and occurs when S is an *internal subgoal* to the evaluation of R. Although the two base cases are very distinct, they share some side-effects, which we discuss next.

3.1 External Pruning

Consider the program that follows and the query goal '`?- a(X), p(Y,Z)`'.

```
:- use_subsumptive_tabling p/2.

a(X) :- p(1,X).          p(1,3).
a(X) :- ...              p(X,Y) :- ...
```

Initially, `a(X)` calls `p(1,X)` which succeeds with {`X=3`}, and in the continuation `p(Y,Z)` is called (Fig. 1(a)). The call to `p(Y,Z)` then checks if there are more specific subgoals being evaluated and it finds `p(1,X)`. The subgoal frame for `p(1,X)` is thus marked as a *consumer subgoal frame* and its *producer subgoal* field is set to point to the subgoal frame for `p(Y,Z)`. Next, the choice point for `p(1,X)` is transformed from a generator node to a *retroactive node*, which amounts to pruning `p(1,X)`'s current evaluation by updating the *continuation alternative* choice point field to a pseudo-instruction called *retroactive_resolution* (Fig. 1(b)). Upon backtracking, this instruction will allow the retroactive node to transform itself in other types of nodes, as we will see.

After `p(Y,Z)` has completed, the evaluation backtracks to `p(1,X)` and, by executing the *retroactive_resolution* instruction, it is detected that the *producer subgoal* has already completed. The choice point for `p(1,X)` is then converted to

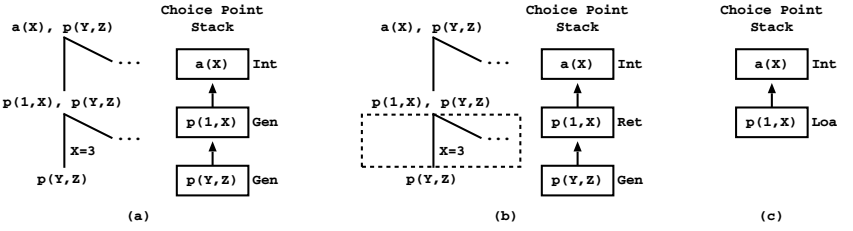


Fig. 1. The base case for external pruning

a *loader node*[2] (Fig. 1(c)) in order to load from `p(Y,Z)` all the answers relevant to `p(1,X)` minus the answers previously found when the choice point was a generator node (answer {`X=3`} in Fig. 1), therefore avoiding repeated answers.

In the previous example, the pruned evaluation of `p(1,X)` do not included other choice points, just the generator node for `p(1,X)`. The following example mixes subsumptive with variant checks and introduces pruning over choice points belonging to the subsumed subgoal.

```
:- use_variant_tabling [a/2, b/1], use_subsumptive_tabling p/2.

a(X,Y) :- p(1,X), b(Y).          p(1,X) :- a(_,X).
a(X,Y) :- ...                    p(1,X) :- b(X).
b(1).  b(2).                     p(X,Y) :- ...
```

The query goal to consider is '`?- a(X,Y), p(Z,W)`' and the first part of the evaluation is illustrated in Fig. 2.

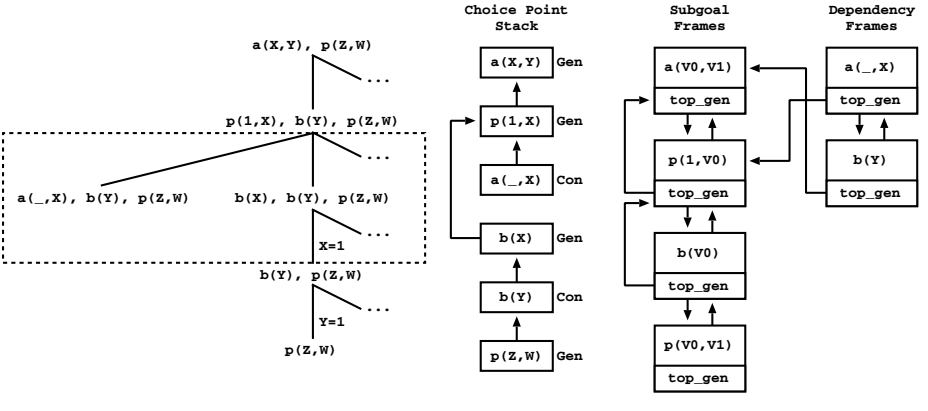


Fig. 2. Before external pruning over choice points belonging to the subsumed subgoal

[2] At the engine level, a loader node is implemented as a consumer node but without a dependency frame, since loader nodes do not need to suspend.

Execution starts by storing generator nodes for `a(X,Y)` and `p(1,X)`. Next, the first clause of `p/2` is executed and a consumer node for `a(_,X)` is allocated. As no answers for this variant subgoal exist, `a(_,X)` is suspended and the execution backtracks to `p(1,X)`. The second clause of `p/2` is then executed and subgoal `b(X)` is called, allocating a new generator node. In the continuation, a first answer for `b(X)` and `p(1,X)` is found, {`X=1`}, and execution proceeds with a call to `b(Y)`. As subgoal `b(Y)` is a variant of `b(X)`, a consumer node is created and the answer {`X=1`} is consumed, thus generating a first answer for `a(X,Y)`, {`X=1,Y=1`}. Finally, subgoal `p(Z,W)` is called and we proceed as in the previous example for the subsumed subgoal `p(1,X)`. But now, as the evaluation of `p(1,X)` includes other choice points, `a(_,X)` and `b(X)`, they should be pruned. Figure 3 shows the state of the computation after pruning. The prune action depends on the choice point type and is discussed next in more detail.

Pruning Interior Nodes. Interior nodes are related to normal Prolog execution and can be easily pruned by ignoring them altogether. This approach, while simple, suffers from the problem of *trapped choice points*. A more complex solution would involve modifications to the WAM garbage collector to collect unused space on the choice point stack.

Pruning Internal Consumers. The consumer node for `a(_,Y)` must be explicitly pruned as otherwise it might be resumed, if new answers are to be consumed, and incorrectly reactivate the pruned branch of `p(1,X)`. For each pruned internal consumer, we must thus delete its corresponding dependency frame.

Pruning Internal Generators. The generator node for `b(X)` must be explicitly pruned as otherwise it might be incorrectly completed. For each pruned internal generator, we must thus remove its corresponding subgoal frame from the subgoal frames stack and alter its state to *pruned*.

Generally, when pruning internal generators, we have two situations: (i) the generator does not have consumers that are external to the computation of the

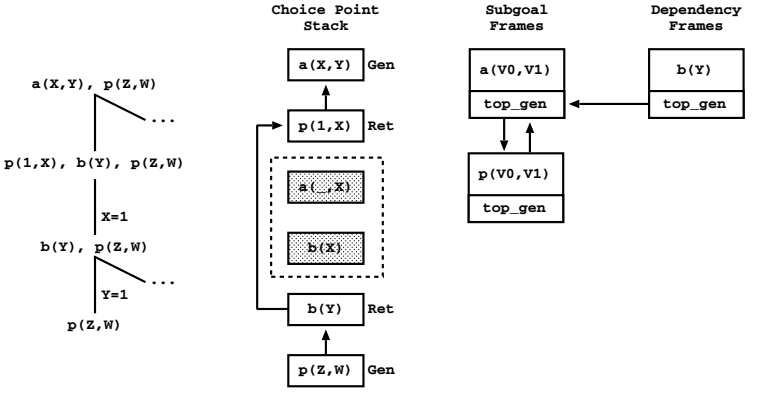


Fig. 3. After external pruning over choice points belonging to the subsumed subgoal

subsumed subgoal; or (ii) the generator has external consumers. The former situation does not introduce any problem, but the latter origins orphaned consumers. In our example, the consumer node for `b(Y)` is external to the evaluation of `p(1,X)` and when pruning `b(X)`, `b(Y)` becomes orphan.

Usually, a pruned generator is called again, during the evaluation of the subsuming subgoal, and before the computation reaches any of the orphaned consumers. Once reactivated, the subgoal frame for the pruned generator is pushed again into the top of the subgoal frame stack and its state altered to *evaluating*. Then, the new generator node starts by consuming the previously generated answers and only then executes the program clauses.

Orphaned Consumers. To deal with orphaned consumers, we use the same strategy as for subsumed subgoals and transform them into retroactive nodes. Upon backtracking, they then become either: (i) a loader node, if the pruned generator was reactivated and has completed; (ii) a consumer node, if the pruned generator was reactivated but has not completed yet; or (iii) a generator node, if the pruned generator was not reactivated until then. This latter situation only occurs in variant-based tabling, since in subsumption-based tabling, when the subsuming call is executed it will necessarily either call the same or a more general subgoal, reactivating a new producer for the orphaned consumers.

By default, orphaned consumers always keep their frames on the dependency frame stack. The frame is only removed if the retroactive node turns into a loader or a generator node. If the retroactive node turns again into a consumer node, lazy removal of dependency frames allows us to avoid removing and allocating a new frame and the potentially expensive operation of inserting it on the dependency frame stack in the correct order (ordered by choice point address).

Lost Consumers. A *lost consumer* is a retroactive node that will not be resumed through standard tabled evaluation, i.e., through backtracking or through answer resolution. When a lost consumer is not resumed, we might lose answers and incorrectly detect completion. As we will see next, to avoid that, we must ensure that all retroactive nodes are always resumed. While in the example of Fig. 3, both `p(1,X)` and `b(Y)` will be resumed by means of backtracking, this may not be always the case. Consider, for example, the program on Fig. 4 and the query goal '`?- a(X,Y)`'.

In this example, when the subsuming call `p(X,Y)` is executed, the generator node `b(1,X)` will be pruned and the external consumer `b(1,Y)` (the center node in the gray oval box) will be turned into a retroactive node and became a lost consumer. Notice that `b(1,Y)` will be resumed neither trough backtracking, since it is not on the current branch, nor through answer resolution, since its pruned generator will not be reactivated.

Hence, when later the computation backtracks to `a(X,Y)` to attempt completion, `b(1,Y)` still remains as a retroactive node. Clearly, to correctly detect completion, we must resume it in order to generate the answers {`X=0,Y=0`} and {`X=0,Y=1`}, as otherwise they will be lost. Therefore, to ensure that all retroactive nodes are resumed, we extended the completion operation to, while traversing

```
:- use_variant_tabling [a/2, b/2].
:- use_subsumptive_tabling p/2.

a(X,0) :- p(1,X).
a(0,Y) :- b(1,Y).
a(X,Y) :- p(X,Y).

b(1,Y) :- a(_,Y).
b(2,1).

p(X,Y) :- b(X,Y).
```

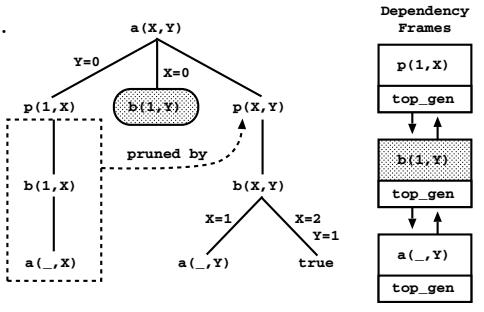


Fig. 4. A lost consumer after an external pruning

the dependency frame stack checking for new answers, also *check for retroactive nodes*, and resume the corresponding consumer node in both cases.

A more tricky situation may occur when a pruned subsumed subgoal R is also a leader node[3]. In such situations, R will not attempt completion and we will not be able to use the completion operation, as described above, to resume any potential lost consumer. Instead, when resuming the computation in R, if R is to be transformed into a loader node (this means that R is a *pseudo-leader* since no dependencies to upper nodes exist), we also traverse the dependency frame stack looking for younger retroactive nodes with unconsumed answers and, when that is the case, execution is first resumed in those nodes.

With these two simple extensions, our strategy is able to ensure that all retroactive nodes are always resumed.

Frontier Node. A *frontier node* is a choice point that is external to the computation of a subsumed subgoal R but that is chained to a choice point that is internal to the computation of R. In the example of Fig. 2, the choice point of `b(Y)` is a frontier node. In order to avoid execution to step into the pruned branch of a subsumed subgoal R upon backtracking, in general, a frontier node must be updated and linked to R. In the example of Fig. 3, the choice point of `b(Y)` is linked to `p(1,X)`. However, for cases where the subsumed subgoal appears outside the branch of the subsuming subgoal, there is no need to update the frontier node. This is safe, because the branch including the subsumed subgoal will only be resumed on consumers during completion and thus no backtracking to previous choice points will occur as they were fully explored before.

3.2 Internal Pruning

Internal pruning occurs when the subsuming subgoal S is internal to the evaluation of the subsumed subgoal R. In this type of pruning we want to keep one part of R running, the one that computes S.

[3] The youngest generator node which does not depend on older generators is called the leader node. A leader node defines the next completion point.

Our approach involves computing S using *local scheduling*[4] [7], but without returning answers to the environment of R, as it has been pruned. Instead, we jump directly to the choice point of R, which was transformed into a retroactive node, and resume the computation there in order to consume the matching answers found by S. When resuming the retroactive node for R, it can become either: (i) a loader node, if S has completed; or (ii) a consumer node, if S has not completed because it is not the leader node, i.e., the leader node is above R.

Notice that, when the completion operation is later attempted at the leader node, the computation can still be resumed, as usual and without any special handling, at R or at the internal consumers of S, until no unconsumed answers are available. Moreover, the effects of pruning involving internal generators and external consumers discussed for external pruning, still apply to internal pruning.

Our engine also supports multiple internal pruning. Consider, for instance, that a subgoal R_1 calls recursively internal subgoals R_2, .., R_n until a subgoal S is called that subsumes R_1, R_2, ..., R_n. In such cases, we ignore all intermediate subgoals and answers are only pushed from S to R_1, the top subgoal. For an example, consider the query goal '`?- p(1,X)`' and the following program:

```
:- use_subsumptive_tabling p/2.

p(1,X) :- p(2,X).
p(2,X) :- p(X,_).
p(X,Y) :- ....
```

Execution starts by storing generator nodes for `p(1,X)` and `p(2,X)`, and then `p(2,X)` calls `p(X,_)` that subsumes both `p(1,X)` and `p(2,X)`. Pruning is done between the top subsumed subgoal `p(1,X)` and the subsuming subgoal `p(X,_)` and the node for `p(2,X)` is ignored (Fig. 5(a)). The choice point for `p(1,X)` is transformed into a retroactive node and execution proceeds by applying local scheduling to evaluate `p(X,_)` (Fig. 5(b)).

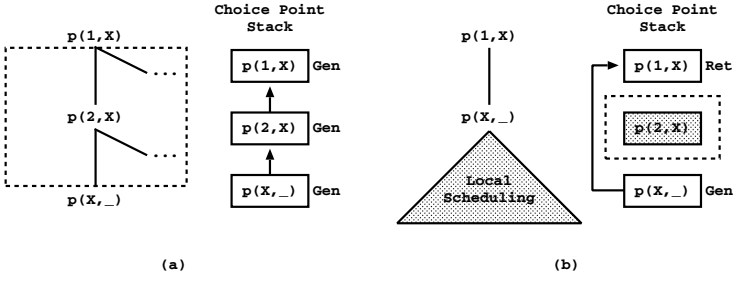


Fig. 5. Multiple internal pruning

[4] Local scheduling is a scheduling strategy that tries to evaluate subgoals as independently as possible. The key idea is that whenever new answers are found, they are added to the table space as usual but execution fails. Hence, answers are only returned when all program clauses for the subgoal at hand were fully explored.

Later, if `p(X,_)` completes, the computation is resumed at `p(1,X)` and the retroactive node is transformed into a loader node, thus loading the matching answers from `p(X,_)`. If `p(_,X)` could not complete because it is not the leader node, the evaluation still resumes at `p(1,X)` and the retroactive node is transformed into a consumer node, and for that it allocates a new dependency frame and inserts it in the proper order in the dependency frame stack.

4 Implementation Details

In this section, we describe implementation details worthy of note. First, we will describe some other topics related to pruning, next we present how the table space is organized, and, then, we give a brief overview of the algorithm used to search for subsumed subgoals.

From Consumers to Generators. In order to be able to transform a consumer node into a retroactive node and then into a generator node, all consumer choice points are allocated, by default, as generator choice points. We are studying a more sophisticated solution that will uniformize the choice point representation of tabled nodes in order to simplify this problem.

External or Internal. Each subsumptive subgoal frame was extended with two new fields: *start_code* and *end_code*, that initially are standard WAM variables. The *start_code* field is bound to an arbitrary value when the subgoal starts executing and unbound when the subgoal backtracks, therefore allowing us to easily detect if the subgoal is in the current branch. The *end_code* field is bound when a new answer is found, i.e., when the code has reached the end of a program clause, thus allowing us to easily detect if a pruned subgoal is external or internal.

To reconstruct the dependency tree of generators and consumers, we extended both subgoal and dependency frames with a new *top_gen* field (see Figs 2 and 3). When a new frame is created, this field is set to point to the subgoal frame corresponding to the generator node whose code is in execution, if any. This subgoal frame is represented by a global variable that is updated when a new generator is called or when it reaches the end of a program clause. Dependency frames save the value of this global variable in order to correctly restore it when a consumer node is resumed.

In order to detect if a subgoal S is external or internal to another subgoal R, the *top_gen* links are traversed until: (i) R is reached and, in this case, S is internal; or (ii) we reach a subgoal older than R and, in this case, S is external.

Table Space. In our subsumptive engine, all answers for a tabled predicate are stored in a *Single Time Stamped Trie (STST)* that is common to all subgoals calls for the predicate. This approach reduces memory usage, allows easy sharing of answers between different subgoals and, in particular, allows us to efficiently load answers from subsumed/subsuming subgoals. The STST keeps a time stamp for the last answer inserted and each subgoal frame keeps a time stamp for the

last answer generated or consumed. This time stamps are then used to easily compute the answers to be considered when pruning a subsumed subgoal.

This approach also allows reusing the answers on the STST when a new subgoal is called. As an example, consider that two unrelated (no subsumption involved) subgoals S_1 and S_2 are fully evaluated. If a subgoal S is then called, it is possible that some of the answers on the STST match S even if S neither subsumes S_1 nor S_2. Hence, instead of eagerly running the predicate clauses, we start by loading the matching answers already on the STST, which can be enough if, for example, S gets pruned by a cut. While this approach has some advantages, it can lead to redundant computations if later, when running the program clauses, S generates more general answers than the ones initially loaded.

Searching Subsumed Subgoals. Each subgoal trie node was extended with a new field called *in_eval* which stores the number of subgoals, represented below the node, that are in evaluation. The subgoal trie path is incremented or decremented when the subgoal enters or exists the computation, respectively. Our algorithm for searching subsumed subgoals then uses this field to quickly discard irrelevant trie branches as it descends the trie. The search is done by matching the subgoal arguments through backtracking along the subgoal trie. On a successful match, the algorithm stores a continuation for an alternative node before descending into the next trie level. If a match fails, a continuation is popped from the continuation stack and the state of the computation is restored (bindings and the trie node). The matching works by the following rules. A non-variable term argument must always match with a non-variable trie symbol. Unbound variable term arguments can match any trie symbol and are bound to the symbol before descending. When a variable term bounds to a trie variable, it must always match against the same trie variable on the following matches.

5 Experimental Results

In this section, we present some experimental results comparing our new RCS proposal with traditional call subsumption. The environment for our experiments was an Intel Core(TM) 2 Quad 2.66 GHz with 4 GBytes of memory and running the Linux kernel 2.6.31 with YapTab 6.0.3 and XSB Prolog 3.2.

First, we measured the RCS overhead for programs that do not take advantage of it, i.e., programs that never call more general subgoals after specific ones, and for that we used a set of benchmark programs downloaded from the XSB Prolog repository[5] with different configurations of data (see Table 1).

Table 1 shows the running time, in milliseconds, for YapTab using RCS (column **RCS**), and its ratio compared with traditional call subsumption for both YapTab (column **CS/RCS**) and XSB Prolog (column **XSB/RCS**). Our results show that, for this set of programs, RCS support adds a very small overhead to running time when compared with traditional call subsumption, and for some programs it can even run faster, probably due to cache behavior effects.

[5] http://xsb.cvs.sourceforge.net/viewvc/xsb/xsbtests

Table 1. RCS running times on programs that do not benefit from it

Program/Data		RCS	CS/RCS	XSB/RCS
right first	*binary tree*	170	0.94	0.96
	chain	2,900	0.99	1.10
	grid	14,456	1.18	1.20
	pyramid	11,520	1.00	1.01
left last	*binary tree*	146	0.93	0.96
	chain	3,068	0.93	1.24
	grid	13,356	1.00	1.28
	pyramid	3,260	1.08	0.90
double last	*binary tree*	952	0.94	1.03
	chain	2,912	0.90	1.41
samegen	*binary tree*	8,272	0.97	1.20
	chain	44	1.18	0.64
Average			1.00	1.08

Table 2. RCS running times on programs that are expcted to benefit from it

Program/Data		RCS	CS/RCS	XSB/RCS
left first	*binary tree*	156	1.18	1.18
	chain	44	1.00	1.18
	grid	40	1.20	1.40
	pyramid	200	0.92	0.96
double first	*binary tree*	784	1.62	1.46
	chain	2,916	0.95	1.24
	grid	2,132	0.97	1.30
	pyramid	12,004	1.01	1.19
reach first	*iproto*	2,244	1.54	2.77
	leader	3,152	1.92	3.82
	sieve	19,997	1.94	1.96
reach last	*iproto*	2,300	1.51	2.77
	leader	3,128	1.94	4.33
	sieve	17,693	2.18	2.97
Average			1.42	2.01

Next, we tried some benchmarks where RCS was expected to perform better. We used a *path/2* program, left and doubly recursive variants, to compute the query goal '`?- path(X,1)`' and also experimented a *reach/2* program with three different transition relation graphs used in model-checking applications.

From Table 2, we can observe that for some *path/2* benchmarks, RCS performs worse, possibly because the time saved by pruning evaluation branches does not pay the incurred overhead in traversing the new STST table organization to collect relevant answers. For the model-checking benchmarks, RCS performance is notoriously better than traditional call subsumption, with speedups reaching 2.18 for YapTab and 4.33 for XSB. For these benchmarks, the performance

impact of using the STST is much smaller when compared to the time saved by pruning evaluation branches of subsumed subgoals.

6 Conclusions and Further Work

We have presented a new tabling extension called Retroactive Call Subsumption that supports full sharing of answers between subsumptive subgoals, regardless of the order in which they are called, and we have described the main concepts and operational challenges of its design in the context of the YapTab system.

Initial experiments comparing our proposal with traditional call subsumption semantics, showed low overheads when executing standard programs and good results when applied to tabled programs that can benefit from it. Further work involves refining the system, testing it against more real-world applications, and studying the impact on performance and memory usage of new data structures, like the Single Time Stamped Trie.

As argued by Johnson et al. [3], for some programs it may be useful to lose some goal directness by means of *call abstraction* and, instead of calling a goal G, we may decide to call a more general goal G' and then reuse the answers from G' to solve G. Our implementation already includes all the machinery necessary to do that, which makes it a good framework to further support call abstraction by devising various analysis techniques of call patterns.

References

1. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. Journal of the ACM 43(1), 20–74 (1996)
2. Rao, P., Ramakrishnan, C.R., Ramakrishnan, I.V.: A Thread in Time Saves Tabling Time. In: Joint International Conference and Symposium on Logic Programming, pp. 112–126. The MIT Press, Cambridge (1996)
3. Johnson, E., Ramakrishnan, C.R., Ramakrishnan, I.V., Rao, P.: A Space Efficient Engine for Subsumption-Based Tabled Evaluation of Logic Programs. In: Middeldorp, A. (ed.) FLOPS 1999. LNCS, vol. 1722, pp. 284–300. Springer, Heidelberg (1999)
4. Costa, J., Rocha, R.: Global Storing Mechanisms for Tabled Evaluation. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 708–712. Springer, Heidelberg (2008)
5. Rocha, R., Silva, F., Santos Costa, V.: On applying or-parallelism and tabling to logic programs. Theory and Practice of Logic Programming 5(1 & 2), 161–205 (2005)
6. Warren, D.H.D.: An Abstract Prolog Instruction Set. Technical Note 309, SRI International (1983)
7. Freire, J., Swift, T., Warren, D.S.: Beyond Depth-First: Improving Tabled Logic Programs through Alternative Scheduling Strategies. In: Kuchen, H., Swierstra, S.D. (eds.) PLILP 1996. LNCS, vol. 1140, pp. 243–258. Springer, Heidelberg (1996)

Preference-Based Inconsistency Assessment in Multi-Context Systems*

Thomas Eiter, Michael Fink, and Antonius Weinzierl

Institute of Information Systems
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,fink,weinzierl}@kr.tuwien.ac.at

Abstract. Resolving inconsistency in knowledge-integration systems is a major issue, especially when interlinking heterogeneous, autonomous sources. The latter can be done using a multi-context system, also in presence of non-monotonicity. Recent work considered diagnosis and explanation of inconsistency in such systems in terms of faulty information exchange. To discriminate between different solutions, we consider inconsistency assessment using preference. We present means to a) filter undesired diagnoses b) select the most preferred ones given an arbitrary preference order and c) use CP-nets for efficient selection. Furthermore, we show how to incorporate the assessment into a Multi-Context System by a transformational approach. In a range of settings, the complexity does not increase compared to the basic case and key properties like decentralized information exchange and information hiding are preserved.

Keywords: Inconsistency Management, Multi-Context Systems, Hybrid Reasoning Systems, Nonmonotonic Reasoning, Preferences.

1 Introduction

Inconsistencies in heterogeneous, nonmonotonic knowledge-integration systems often do not have a single cause, but emerge from interaction, i.e., by the exchange of knowledge between knowledge bases. The nonmonotonic Multi-Context System (MCS) framework of [4], which extends seminal works by [9,6], is a logic-based approach to flexibly model the information exchange between heterogeneous (nonmonotonic) knowledge bases, which exist a priori and incorporate external knowledge via so-called bridge rules. Recently, formal notions for explaining inconsistency in such MCSs in terms of faulty bridge rules have been developed [8], serving the purpose of inconsistency analysis with the eventual aim of resolving inconsistency. However, multiple possibilities for this call for a further assessment, taking application specific criteria into account.

To the best of our knowledge, no general method has been proposed to assess inconsistencies in MCSs which is flexible enough to adapt to application specific

* Supported by the Vienna Science and Technology Fund (WWTF), grant ICT08-020.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 143–155, 2010.
© Springer-Verlag Berlin Heidelberg 2010

criteria. Although, for instance, [2] provides methods based on local trust and provenance to determine preferred models for a MCS avoiding inconsistency, the proposal requests to choose one out of four predefined evaluation algorithms. Our work instead aims at general techniques for assessing inconsistency in MCSs that can be 'instantiated' to encode application-specific properties for preferred consistency restorations.

For example, consider a health-care decision-support system that interlinks knowledge sources about patient histories, lab test results, a disease ontology, and a decision support system for patient treatment. Here, an inconsistency might easily arise if some scenario of contradicting information has not been anticipated. E.g., the ontology classifies symptoms as atypical pneumonia, which requires strong antibiotics, but a patient is allergic to it; the treatment system may then raise an inconsistency. One possibility to resolve it is to ignore the imported disease information. While technically fine, this solution might be unacceptable, as a constraint "No illness of a patient may be ignored" should be fulfilled.

To account for such selection criteria on consistency restorations, we take a preference-based approach. Two basic elements of preference-based selection can be found in the literature: filters, which discard unpreferred solutions that fail some preference condition, and qualitative comparison relations establishing preference orders to single out the most appealing solutions. Our main contributions enabling these for inconsistency assessment in MCSs are summarized as follows.

- We formalize both preference approaches above in the setting of MCSs. For preference orders, we further investigate the application of conditional preference networks (CP-nets), which exhibit appealing features of locality and privacy. CP-nets [3] capture a natural class of preference statements like "If my new car is from Japan, I prefer hybrid over diesel engine, assuming all else is equal".
- We further show how to realize the preference approaches inside the MCS framework by using meta-reasoning on consistency restorations. For a given MCS and a filter, preference order, or CP-net, a rewriting yields a transformed system such that consistency restorations of the latter directly correspond to preferred consistency restorations of the original system (wrt. the given filter, preference order, or CP-net).
- For preference notions that are not inherently centralized, the realization allows that preferred solutions are found in a decentralized, localized manner, maintaining privacy and information hiding. Thus we preserve key properties of MCSs also for inconsistency assessment.

Our results not only refine existing methods for inconsistency handling in MCSs without complexity increase, but also show the versatility of the basic framework to couch advanced reasoning tasks, including self-reflective assessment.

2 Preliminaries

This section introduces MCS and diagnoses in general; it is largely based on [8].

A heterogeneous nonmonotonic MCS [4] consists of *contexts*, which comprise knowledge bases in underlying *logics*, and *bridge rules* to control the information flow between contexts.

A logic $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$ consists, in an abstract view, of

- a set $\mathbf{KB}_L$ of knowledge bases of L, each being a set (of "formulas"),
- a set $\mathbf{BS}_L$ of possible belief sets, whose elements are "beliefs", and
- a "semantics" function $\mathbf{ACC}_L : \mathbf{KB}_L \to 2^{\mathbf{BS}_L}$ which assigns each knowledge base a set of acceptable belief sets.

This concept of a *logic* captures many monotonic and nonmonotonic logics, e.g., classical logic, description logics, modal logics, default logics, circumscription, and logic programs under the answer set semantics.

A *bridge rule* can add information to a context, depending on the belief sets which are accepted at other contexts. Let $L = (L_1, \ldots, L_n)$ be a sequence of logics. An L_k-bridge rule r over L is of the form

$$(k : s) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathbf{not}\ (c_{j+1} : p_{j+1}), \ldots, \mathbf{not}\ (c_m : p_m). \quad (1)$$

where $1 \leq c_i \leq n$, p_i is an element of some belief set of L_{c_i}, k refers to the context receiving information s. We denote by $hd\,(r)$ the formula s in the head of r.

A *multi-context system (MCS)* is a collection $M = (C_1, \ldots, C_n)$ of contexts $C_i = (L_i, kb_i, br_i)$, $1 \leq i \leq n$, where $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ is a logic, $kb_i \in \mathbf{KB}_i$ a knowledge base, and br_i is a set of L_i-bridge rules over $(L_1, \ldots, L_n)$. In addition, for each $H \subseteq \{hd\,(r) \mid r \in br_i\}$ we have $kb_i \cup H \in \mathbf{KB}_i$, i.e., bridge rule heads are compatible with knowledge bases. By $br_M = \bigcup_{i=1}^{n} br_i$ we denote the set of bridge rules of M.

A *belief state* of an MCS $M = (C_1, \ldots, C_n)$ is a sequence $S = (S_1, \ldots, S_n)$ such that $S_i \in \mathbf{BS}_i$. A bridge rule (1) is *applicable* in a belief state S iff for $1 \leq i \leq j$: $p_i \in S_{c_i}$ and for $j < l \leq m$: $p_l \notin S_{c_l}$.

Example 1. Consider two scientists, Prof. K and Dr. J, planning to write a paper. We formalize their reasoning in an MCS M using two contexts each employing answer set semantics. Dr. J will write most of the paper and Prof. K only participates if either he finds time or if Dr. J thinks the paper needs improvement (bridge rule r_1). Dr. J knows that the participation of Prof. K results in a good paper (r_2 and kb_J) and he will name Prof. K as author if she participates (r_3). The knowledge bases of the contexts are:

$$kb_K = \{has\_time. \quad contribute \leftarrow improve. \quad contribute \leftarrow has\_time.\}$$
$$kb_J = \{good \leftarrow coauthored.\}$$

The bridge rules are $r_1 = (K{:}improve) \leftarrow \mathbf{not}\ (J{:}good).$,
$r_2 = (J{:}coauthored) \leftarrow (K{:}contribute).$, and $r_3 = (J{:}name\_K) \leftarrow (K{:}contribute).$

Equilibrium semantics selects certain belief states of an MCS $M = (C_1, \ldots, C_n)$ as acceptable. Intuitively, an equilibrium is a belief state $S = (S_1, \ldots, S_n)$ where each context C_i respects all bridge rules applicable in S and accepts S_i. Formally, S is an equilibrium of M, iff for $1 \leq i \leq n$,

$$S_i \in \mathbf{ACC}_i(kb_i \cup \{hd\,(r) \mid r \in br_i \text{ applicable in } S\}).$$

Example 2 (Ex. 1 ctd.). The MCS has just one equilibrium $S = (\{has\_time, contribute\}, \{coauthored, name\_K, good\})$ where both scientists author a good paper. Bridge rules r_2 and r_3 are applicable in S.

Inconsistency in an MCS is the lack of an equilibrium.

Example 3 (Ex. 1 ctd.). Assume Prof. K has no time, so she only contributes, if Dr. J considers the paper to be not good, i.e, $kb_K = \{contribute \leftarrow improve.\ contribute \leftarrow has\_time.\}$. Then there is a loop with an odd number of negations via bridge rules r_1 and r_2. This makes the MCS inconsistent.

For any MCS M and set R of bridge rules (fitting M), we denote by $M[R]$ the MCS obtained from M by replacing br_M with R (e.g., $M[br_M] = M$ and $M[\emptyset]$ is M with no bridge rules); by $M \models \bot$ we denote that M has no equilibrium (is inconsistent). For any set of bridge rules A, $heads(A) = \{\alpha \leftarrow \top \mid \alpha \leftarrow \beta \in A\}$ are the rules in A in unconditional form.

Diagnoses. As well-known, in nonmonotonic reasoning, adding knowledge can both cause and prevent inconsistency; the same is true for removing knowledge. The consistency-based explanation of inconsistency, therefore considers pairs (D_1, D_2) of sets of bridge rules, such that if the rules in D_1 are deactivated, and the rules in D_2 are added in unconditional form, the MCS becomes consistent (i.e., it admits an equilibrium). Adding rules unconditionally makes sense due to non-monotonicity; the idea is related to that of consistency restoring rules [1].

Formally, a *diagnosis* of an MCS M is a pair $D = (D_1, D_2)$, $D_1, D_2 \subseteq br_M$, s.t. $M[br_M \setminus D_1 \cup heads(D_2)] \not\models \bot$; by $D^{\pm}(M)$ we denote the set of all diagnoses. To obtain a more relevant set of diagnoses, pointwise subset-minimal diagnoses are preferred; we denote by $D^{\pm}_m(M)$ the set of all such diagnoses of an MCS M.

In our example $D^{\pm}_m(M) = \{(\{r_1\}, \emptyset), (\{r_2\}, \emptyset)(\emptyset, \{r_2\}), (\emptyset, \{r_1\})\}$; the first two diagnoses break the cycle by removing a rule, the last two "stabilize" it.

3 Filtering and Comparing Diagnoses

In this section we introduce ways to assess consistency restorations of inconsistent MCSs. First, we consider selection-based preference and provide a method to check whether diagnoses adhere to user-defined criteria. This allows to filter out undesired diagnoses. Then we turn to comparison-based preference, addressing the general problem of using arbitrary preference relations on diagnoses, before we focus on CP-nets, representing (semi-)local preference relations.

3.1 Filtering Diagnoses

Filters allow a designer of an MCSs to apply sanity checks on diagnoses, thus they can be seen as hard constraints on diagnoses: diagnoses that fail to satisfy the conditions are filtered out and not considered for consistency restoration.

Definition 1. *Let M be an MCS with bridge rules br_M. A* diagnosis filter *for M is a function $f{:}2^{br_M} \times 2^{br_M} \to \{0,1\}$ and the set of* filtered diagnoses *is $D_f^{\pm}(M) = \{D \in D^{\pm}(M) \mid f(D) = 1\}$. By $D_{f,m}^{\pm}(M)$ we denote the set of all subset-minimal such diagnoses.*

Example 4 (Ex. 3 ctd.). Consider the diagnoses $D = (\{r_2\}, \emptyset)$ and $D' = (\emptyset, \{r_2\})$, where the contribution of Prof. K is either enforced or forbidden. For both cases, the authorship information conveyed by r_3 is wrong. Using a filter, we can declare diagnoses undesired if they modify r_2 without modifying r_3 accordingly, in particular $f(D) = f(D') = 0$.

As it is a key strength of MCS to integrate different knowledge bases in a decentralized manner, users of MCS will want to specify their constraints on diagnoses in a logic of their choice, decentralized, and under the provision that they do not have to disclose information considered private. In Section 4 we realize filters within the MCS formalism, such that these properties are retained.

3.2 Comparing Diagnoses

To compare minimal diagnoses, we first consider an arbitrary preference order to select most preferred diagnoses, and further on focus on CP-nets. A *preference order* over diagnoses for an MCS M is a transitive binary relation $\preceq$ on $2^{br_M} \times 2^{br_M}$; we say that D is preferred to D' iff $D \preceq D'$.

Definition 2. *Let M be an inconsistent MCS. A diagnosis $D \in D^{\pm}(M)$ of M is called pre-most preferred iff for all $D' \in 2^{br_M} \times 2^{br_M}$ with $D' \preceq D \wedge D \not\preceq D'$ it holds that $D' \notin D^{\pm}(M)$. A diagnosis $D \in D^{\pm}(M)$ is called* most preferred, *iff D is subset-minimal among all pre-most preferred diagnoses.*

Given that MCSs are decentralized systems, users may want to express preferences on diagnoses solely based on a local set of bridge rules, assuming all other things equal. Such preferences can be formalized using CP-nets, which are an extension of *ceteris paribus* orders ("all else being equal"). They represent local preference and have successfully been used for preference elicitation (e.g. [7]).

Example 5. Assume an MCS where several corporations make contracts using bridge rules. Contract details, such as when a contract will start, how long it is valid, who owns what to whom, etc, are encoded with bridge rules. For instance, C_1 is leasing a car from C_2 with the following properties encoded as bridge rules $r_1 = (C_1 : pay(car, 500)) \leftarrow (C_2 : price(car, 500))$ and $r_2 = (C_1 : due(car, monthly)) \leftarrow (C_2 : due(car, monthly))$. If r_2 is removed to restore consistency, r_1 becomes meaningless and possibly confuses further reasoning. Removing both rules is then preferred to removing only r_2.

A *CP-net* is a directed graph (V, E) where V is a finite set of variables (attributes) and $E \subseteq V \times V$ is the conditional dependency between variables. For $v \in V$ we denote the set of parents of v by $pa(v) = \{v' \in V \mid (v', v) \in E\}$. Furthermore, the set of outcomes of a variable v is denoted by $dom(v)$. Preferences

on the outcomes of a variable are specified in terms of total preorders, which allow indifference. A relation $\precsim$ is a *total preorder*, iff it is transitive, reflexive, and for any two elements o, o' of $\precsim$ it holds that $o \precsim o' \vee o' \precsim o$.

Each vertex v in a CP-net (V, E) is associated with a *conditional preference table* (CPT) p_v that maps each combination of outcomes of parents of v, i.e., $o \in dom(p_1) \times \ldots \times dom(p_n)$, to a total preorder $\precsim_v(o) \subseteq dom(v) \times dom(v)$ over the outcomes of v.

We associate a CP-net (V, E) with an MCS M, if every variable $v \in V$ is assigned a set of bridge rules $rules(v) \subseteq br_M$, such that the assignment of rules is disjoint, i.e., $\forall v, v' \in V : rules(v) \cap rules(v') = \emptyset$. Moreover, $dom(v)$ for every v is given by $dom(v) = \{unchanged_r, removed_r, unconditional_r \mid r \in rules(v)\}$. In the following we confine here to CP-nets that are acyclic, i.e, the directed graph (V, E) contains no cycles, and whose preference graph over outcomes is acyclic.

Example 6 (Ex. 5 ctd.). Recall r_1 and r_2 encoding properties of a leasing contract.

If r_2 is removed, r_1 is preferred to be removed, too. Consider an associated CP-net $N = (\{v_1, v_2\}, \{(v_2, v_1)\})$, i.e. $pa(v_1) = \{v_2\}$ and $pa(v_2) = \emptyset$, where $rules(v_1) = \{r_1\}$, $rules(v_2) = \{r_2\}$. Assuming that adding rules unconditionally is always considered to be the worst option, v_1's conditional preference table is:

$$p_{v_1}(unchanged_{r_2}) = unchanged_{r_1} \prec_{v_1} removed_{r_1} \prec_{v_1} unconditional_{r_1} \quad (2)$$

$$p_{v_1}(removed_{r_2}) = removed_{r_1} \prec_{v_1} unchanged_{r_1} \prec_{v_1} unconditional_{r_1} \quad (3)$$

$$p_{v_1}(unconditional_{r_2}) = unchanged_{r_1} \prec_{v_1} removed_{r_1} \prec_{v_1} unconditional_{r_1} \quad (4)$$

For v_2 the table p_{v_2} is $unchanged_{r_2} \prec_{v_2} removed_{r_2} \prec_{v_2} unconditional_{r_2}$.

A CP-net N induces a preference graph G_N over outcomes, where each global outcome is a node in the preference graph. An arc from outcome o_i to o_j indicates that a preference for o_j over o_i can be determined directly from one conditional preference table of the CP-net (cf. [3]). The transitive closure G_N^+ of a preference graph induces a partial order on global outcomes. Furthermore, for a CP-net associated with an MCS, every global outcome represents a potential diagnosis.

Proposition 1. *Let M be an inconsistent MCS, and let N be a CP-net associated with M. Then G_N^+ induces a preference order $\prec$ over diagnoses of M.*

By $D_{opt}^{\pm}(M, N)$ we denote the subset-minimal among the most preferred diagnoses according to G_N^+, i.e., for which no other diagnosis is more preferred.

The semantics of CP-nets may also be defined in terms of *flips*: Let $|V| = m$, and let $\boldsymbol{a} = (a_1, \ldots, a_i, \ldots, a_m)$ and $\boldsymbol{b} = (a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_m)$ be two global outcomes with $a_j, b_j \in dom(v_j)$, that differ only in the outcome of one variable. The flipping of v_i from a_i to b_i is *improving*, iff in the CPT of v_i outcome b_i is preferred over a_i, given all other parent variables set as in $\boldsymbol{a}$ and $\boldsymbol{b}$. The converse notion of an improving flip is called a *worsening* flip. A global outcome is *optimal*, if no improving flips are possible. Notably, for CP-nets an optimal outcome is reachable from any outcome by a finite sequence of improving flips.

In terms of flips, the most preferred diagnoses $D^{\pm}_{opt}(M,N)$ of an MCS are: $D^{\pm}_{opt}(M,N) = \min_{\subseteq}\{D \in D^{\pm}(M) \mid \forall D' \in D^{\pm}(M) : \mathit{iflips}(D,D') = \emptyset\}$, where $\mathit{iflips}(D,D')$ denotes the set of sequences of improving flips from D to D'.

4 MCS-Realization

We now present ways to realize filters, preference orders, and CP-nets. All realizations use a rewriting technique transforming an MCS M into an extended MCS M', where certain new contexts can do meta-reasoning on diagnoses of the original M. This is achieved in a way, such that a diagnosis of M' directly corresponds to a diagnosis of M, and subset-minimal diagnoses of M' coincide with the preferred diagnoses of M.

Meta-reasoning as described below allows certain contexts to observe whether a bridge rule of M is part of a diagnosis. For this, the context observes the body and head beliefs of a bridge rule. For a diagnosis (D_1, D_2) and a bridge rule r, if the body of r is satisfied, but its head is not believed, then $r \in D_1$; if the body is not satisfied, but the head is believed, then $r \in D_2$. The observation of body and head beliefs is accomplished by additional bridge rules in M', that are not subject to diagnosis. We thus adapt the notion of diagnosis such that certain bridge rules, tagged as protected, are never part of it.

Definition 3. *Let M be an MCS with* protected rules $br_P \subseteq br_M$. *A* diagnosis excluding protected rules br_P *is a diagnosis $(D_1, D_2) \in D^{\pm}(M)$, where $D_1, D_2 \subseteq br_M \setminus br_P$. We denote the set of all minimal such diagnoses by $D^{\pm}_m(M, br_P)$.*

A direct consequence is the following:

Proposition 2. *Let M be an inconsistent MCS with protected rules br_P. Then $D^{\pm}_{(m)}(M, br_P) \subseteq D^{\pm}_{(m)}(M)$, i.e., every (minimal) diagnosis excluding protected rules is a (minimal) diagnosis.*

Furthermore one can show that the duality between diagnoses and inconsistency explanations (cf. [8]) also holds for diagnoses and inconsistency explanations excluding protected rules, and that computing such diagnoses has the same complexity as computing ordinary diagnoses.

Meta-Reasoning Transformation: Using additional protected bridge rules in order to observe a bridge rule r with head $(k : s)$, we aim at monitoring the import of belief s into context k. Accessing k directly however, will in general not serve this purpose, since s could be in an accepted belief set of k also without import. In order to observe r properly, we therefore introduce a relay context for k, which can then be accessed by an observer.

Given an MCS M and a set of bridge rules br_o to be observed, an observation context ob for br_o is a context with bridge rules $br_{ob} = br^{ob}_b \cup br^{ob}_h$ with $br^{ob}_b = \{r^{ob}_b \mid r \in br_o\}$ and $br^{ob}_h = \{r^{ob}_h \mid r \in br_o\}$, where r^{ob}_b and r^{ob}_h are of the form

$$(ob : body_r) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathbf{not}\ (c_{j+1} : p_{j+1}), \ldots, \mathbf{not}\ (c_m : p_m). \quad (5)$$

$$(ob : head_r) \leftarrow (relay_k : s). \quad (6)$$

for a bridge rule (1), respectively. Here, $relay_k$ is the relay context for context k (cf. below). Context ob is conservative iff $\mathbf{ACC}_{ob}(S) \neq \emptyset$ for every $S \subseteq \{hd(br_{ob})\}$.

Now, let $B_k = \{s \mid (k:s) \leftarrow \top \in heads(br_o)\}$. We say that $relay_k$ is a relay context for context C_k wrt. br_o iff $\mathbf{KB}_{relay_k} = \mathbf{BS}_{relay_k} = 2^{B_k}$, $\mathbf{ACC}_{relay_k}(S) = \{S\}$, $kb_{relay_k} = \emptyset$, and $br_{relay_k} = \{r_{relay} \mid r \in br_o\}$, where r_{relay} is of the form

$$(relay_k : s) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathbf{not}\ (c_{j+1} : p_{j+1}), \ldots, \mathbf{not}\ (c_m : p_m). \quad (7)$$

for a bridge rule of the form (1). Furthermore, we associate with context $C_k = (L_k, kb_k, br_k)$ its relayed context $C_k^{rel} = (L_k, kb_k, (br_k \setminus br_{ob}) \cup br_k^{rel})$ wrt. br_o, where $br_k^{rel} = \{r_{rel} \mid r \in br_o\}$, and r_{rel} is for a bridge rule (1) of the form

$$(k : s) \leftarrow (relay_k : s). \quad (8)$$

Based on this, the meta-reasoning transformation of an MCS is as follows.

Definition 4. *Given an MCS $M = (C_1, \ldots, C_n)$, let $B = \{(ob_1, br_{o_1}), \ldots, (ob_m, br_{o_m})\}$ be an association of observation contexts $ob_i \notin M$ to disjoint sets of bridge rules $br_{o_i} \subseteq br_M$. The* meta-reasoning transformation *M^B of M wrt. B is the MCS $M^B = (C_1^{rel}, \ldots, C_n^{rel}, relay_1, \ldots, relay_n, ob_1, \ldots ob_m)$, where C_i^{rel} and $relay_i$ are relayed contexts and relay contexts wrt. $\bigcup_{k=1}^m br_{o_k}$, respectively, and $br_P^B = \bigcup_{i=1}^n br_i^{rel} \cup \bigcup_{k=1}^m br_{ob_k}$ are protected rules.*

In cases where contexts are known to not interfere with the beliefs they are importing, a simpler transformation can be obtained where observation contexts directly import from the original contexts and relay contexts are omitted.

Suppose that ob is an ASP context for the observation of a set of rules br_o of an MCS M. Then the following ASP rules allow ob to check whether r is part of a subset-minimal diagnosis:

$$r_{removed} \leftarrow body_r, not\ head_r. \quad (9)$$

$$r_{unconditional} \leftarrow not\ body_r, head_r. \quad (10)$$

$$r_{unchanged} \leftarrow not\ r_{removed}, not\ r_{unconditional}. \quad (11)$$

Note that this is correct for diagnoses $D = (D_1, D_2)$ with $D_1 \cap D_2 = \emptyset$ as otherwise for $r \in D_1 \cap D_2$, ob will not observe $r_{removed}$. We call a diagnosis $D = (D_1, D_2)$ *safe* iff $D_1 \cap D_2 = \emptyset$ holds and for any equilibrium S of $M[br_M \setminus D_1 \cup heads(D_2)]$ holds that $r \in D_1$ only if r is applicable in S and $r \in D_2$ only if r is not applicable in S. Note that all minimal diagnoses are safe. In the following theorem we consider only safe diagnoses.

Theorem 1. *Let M^B be the meta-reasoning transformation of an MCS M wrt. $B = \{(ob_1, br_{o_1}), \ldots, (ob_n, br_{o_n})\}$, let $ob_1, \ldots, ob_n$ be conservative, and let $br_o = \bigcup_{k=1}^n br_{o_k}$. Then,*
(i) $(D_1, D_2) \in D_{(m)}^{\pm}(M^B, br_p^B)$ implies $(D_1', D_2') \in D_{(m)}^{\pm}(M)$, where $D_i' = (D_i \cap br_M) \cup \{r \in br_M \mid r_{relay} \in D_i\}$ for $1 \leq i \leq 2$, and
(ii) $(D_1, D_2) \in D_{(m)}^{\pm}(M)$ implies $(D_1', D_2') \in D_{(m)}^{\pm}(M^B, br_p^B)$, where $D_i' = (D_i \setminus br_o) \cup \{r_{relay} \mid r \in D_i \cap br_o\}$ for $1 \leq i \leq 2$.

This theorem effectively states that the meta-reasoning transformation enables observation contexts to correctly observe the effects of diagnosis. This is the basis of the following realizations, which use non-conservative observation contexts for assessing and pruning diagnoses.

4.1 Filters

Using the above transformation, users of MCSs can analyze diagnoses inside observation contexts a way they see fit. If a diagnosis is considered inappropriate, the observer just needs to become inconsistent which prevents a corresponding diagnosis of the transformed system. This holds because the assessment uses protected bridge rules only. We next present a transformation realizing a general filter. For generality, it uses a central context m_f for analysis.

Definition 5. *Let M be an MCS and f a filter for M. A* filter-transformation *of M wrt. f is a meta-reasoning transformation M^B with $B = \{(m_f, br_M)\}$ and the logic L_f of m_f is such that for any diagnosis $D = (D_1, D_2)$:*

$$\mathbf{ACC}_{m_f}\left(kb_{m_f} \cup \{r_{body} | r \in D_1\} \cup \{r_{head} | r \in D_2\}\right) = \emptyset^{1} \text{ iff } f(D) = 0.$$

Example 7 (Ex. 4 ctd.). For our scientists, we want to filter diagnoses that modify r_2 and r_3 differently, e.g., $D = (\{r_2\}, \emptyset)$. The filter-transformation yields a system with five contexts, $K, relay_K, J, relay_J$, and m_f. The rewritten rules for r_2 are (analogous for r_1 and r_3):

$$(relay_J : coauthored) \leftarrow (K : contribute).$$
$$(J : coauthored) \leftarrow (relay_J : coauthored).$$
$$(m : body_{r_2}) \leftarrow (K : contribute).$$
$$(m : head_{r_2}) \leftarrow (relay_J : coauthored).$$

We use answer-set semantics for the assessment context m_f. To realize the filter function, m_f contains rules (9) - (11) for r_1 and r_2 and:

$$\bot \leftarrow not\ same\_change.$$
$$same\_change \leftarrow r_{1\,unconditional}, r_{2\,unconditional}.$$
$$same\_change \leftarrow r_{1\,unchanged}, r_{2\,unchanged}.$$
$$same\_change \leftarrow r_{1\,removed}, r_{2\,removed}.$$

One can show that the transformation indeed realizes any given filter:

Theorem 2. *Given an inconsistent MCS M, let f be a filter on diagnoses and let M_f be a filter-transformation for f of M with protected rules br_P. Then $D \in D_m^{\pm}(M_f, br_P)$ iff $D \in D_{f,m}^{\pm}(M)$.*

If f is not given abstractly as a function, but as a family of constraints, each set of constraints is realizable using a separate assessment context, observing just the bridge rules it needs to assess. Realizing such a filter is decentralized, adheres to information hiding, and each observer's logic can be chosen as desired.

[1] For logics having always an acceptable belief set, inconsistency can still be created using a new bridge rule $(m_f : inc) \leftarrow (m_f : cause\_inc), \mathbf{not}\ (m_f : inc)$.

4.2 Preference Orders and CP-Nets

The general idea for realizing best outcomes of a CP-net is to create an order-preserving mapping from the CP-net preference to the subset-order of diagnoses. We add a new context m_v for each variable v of the CP-net which "observes" the bridge rules with outcomes represented by v. It searches for improving flips on a path to a more preferred diagnosis. If the combined local guesses succeed, some bridge rules can be removed; ensuring that the zero-length path allows no removal, the most preferred diagnoses are those without removal. To accomplish this, we use prioritized bridge rules whose minimization has precedence.

Definition 6. *Let M be an MCS with bridge rules br_M, protected rules br_P, and prioritized rules $br_H \subseteq br_M$. The set of minimal prioritized diagnoses is*

$$D_m^{\pm}(M, br_P, br_H) = \{\, D \in D_m^{\pm}(M, br_P) \mid \forall D' \in D_m^{\pm}(M, br_P) : \\ D' \cap br_H \subseteq D \cap br_H \Rightarrow D' \cap br_H = D \cap br_H \,\}\,.$$

where $(D_1, D_2) \cap S := (D_1 \cap S, D_2 \cap S)$.

Note that given D, $D_m^{\pm}(M, br_P)$, and br_H, deciding $D \in D_m^{\pm}(M, br_P, br_H)$ is easy.

Let M be an MCS and consider an associated CP-net $N = (V, E)$. We assume that the CPT of any $v \in V$ is given by $v_n = 2^{|dom(p_v)|} \times 2 \times |dom(v)|$ "binarized" preferences (two successive outcomes); let $m = \sum_{v \in V} v_n$ be their total number. Given any total and strict preference order $<_r$ on the set $V_r \subseteq V$ of root nodes in N, we call $E' = E \cup \{(v, v') \mid v <_r v'\}$ a *root extension* of E.

Let $2M$ be the mirrored M, i.e., add a copy C_i' of each of context C_i in M with disjoint beliefs (alphabetic variants). An observer associated with $v \in V$ sees $rules(v)$ and $crules(v) = \{cr \mid r \in rules(v)\}$, where cr is the rule copy of r.

We say a set Q *encodes* o iff (a) $head_r \in Q \Leftrightarrow body_r \in Q$ for $o = unchanged_r$, (b) $head_r \notin Q$ and $body_r \in Q$ for $o = removed_r$, (c) $head_r \in Q$ and $body_r \notin Q$ for $o = unconditional_r$; that Q encodes co is analog. Eventually, let

$$\mathcal{L}_v = \{vf_{i,k},\, vg_{i,k},\, vng_{i,k},\, v'used_k,\, v'diff,\, o,\, co,\, io,\, o_k,\, co_k,\, eq,\, false\},$$

where $1 \leq i \leq v_n$, $1 \leq k \leq m$, $v' \in V$, and $o \in dom(v)$. A set $S \subseteq 2^{\mathcal{L}_v}$ is *compatible with* Q, iff

- $vdiff \in S$ iff (a) $v'diff \in Q$ for some $v' \neq v$, or (b) $o \in Q$, and $co' \in Q$, such that o corresponds to r, co' corresponds to cr, and $o \neq o'$;
- $vused_k \in S$ iff $vg_{i,k} \in Q$ or $v'used_k \in Q$, for some $1 \leq i \leq v_n$ respectively, and $1 \leq k \leq m$;
- $eq \in S$ iff $eq \in Q$ or v is $<_r$-maximal in V_r and $v'diff \notin S \cup Q$ for any $v' \in V$;
- $o_1 \in S$ iff $o \in Q$;
- $o_{k+1} \in S$ for $o \in dom(v)$ iff (a) o is most preferred according to v's i-th CPT entry and $vg_{i,k} \in Q$, or (b) o is the outcome of $o_k \in S$ and $vg_{i,k} \notin Q$;
- $false \in S$ iff for some $1 \leq i \leq v_n$, $1 \leq k \leq m$: (a) $io \in Q$ and $o \in Q$, (b) $eq \in S$ and $vf_{i,k} \in Q$, (c) $vg_{i,k}, vng_{i,k} \in Q$, (d) $vng_{i,k}, vf_{i,k} \in Q$, (e)

$v'used_k, vg_{i,k} \in Q$, where $v' \neq v$, (f) $v'used_k, v''used_k \in Q$, where $v' \neq v''$, (g) $vg_{i,k} \in Q$ and either v's i-th CPT entry is not applicable wrt. $\{o \mid o_k \in Q \wedge o \in \bigcup_{v' \in pa(v)} dom(v')\}$, or it is not improving wrt. $o \in dom(v)$ such that $o_k \in Q$; (h) $o \neq o'$ for o and o' from $dom(v)$, such that $o_m \in S$, and $co' \in Q$.

Definition 7. *Let $N = (V, E)$, $V = \{v_1, \ldots, v_n\}$, be a CP-net associated with an MCS M, and let E' be a root extension of E. The* CP-net transformation of M wrt. N *is the meta-reasoning transformation $2M^B$ of $2M$ wrt. $B = \{(m_{v_i}, rules(v_i) \cup crules(v_i)) \mid 1 \leq i \leq n\}$, putting in observation contexts*

- *for every $v \in V$, $1 \leq i \leq v_n$, and $1 \leq k \leq m$: protected bridge rules $(m_v : vg_{i,k}) \leftarrow not\ (m_v : vng_{i,k})$ and $(m_v : vng_{i,k}) \leftarrow not\ (m_v : vg_{i,k})$, and prioritized bridge rules $(m_v : vf_{i,k}) \leftarrow \top$ and $(m_v : io) \leftarrow \top$ for $o \in dom(v)$;*
- *for every $(v', v) \in E'$, a protected bridge rule $(m_v : eq) \leftarrow (m_{v'} : eq)$;*
- *for every $(v, v') \in E'$, and $1 \leq k \leq m$, protected bridge rules $(m_v : v'used_k) \leftarrow (m_{v'} : v'used_k)$ and $(m_v : v'diff) \leftarrow (m_{v'} : v'diff)$;*
- *for every $(v', v) \in E$, $o \in dom(v')$, and $1 \leq k \leq m$: protected bridge rules $(m_v : o_k) \leftarrow (m_{v'} : o_k)$.*

Furthermore, the logic of every m_v ($v \in V$) has $\mathbf{BS}_{m_v} = 2^{\mathcal{L}_v}$, and $\mathbf{ACC}_{m_v}(kb_{m_v} \cup Q) = S$ for any set S in $\mathbf{BS}_{m_v}$ compatible with Q and false $\notin S$.

Let br^i_H denote the set of all prioritized bridge rules of the form $(m_v : vf_{i,k}) \leftarrow \top$ in $2M^B$. Intuitively, this transformation ensures the following for any observations o and co'. If $o = o'$ (the outcomes correspond to the same diagnosis of the original system), then bridge rules br^i_H need to be removed to obtain a diagnosis of $2M^B$. If $o \neq o'$ (different diagnoses), then there are two cases: either o' is reachable from o via a sequence of improving flips (thus a diagnosis of $2M^B$ exists removing only some of br^i_H), otherwise there is no diagnosis of $2M^B$ for this observation (an inconsistency by the protected rules, which cannot be resolved by removing prioritized rules).

Example 8 (Ex. 6 ctd.). Recall the contracts example, and assume that the resulting MCS M is inconsistent having only two diagnoses $D = (\{r_1, r_2\}, \emptyset)$ and $D' = (\{r_2\}, \emptyset)$. Note that D is preferred over D' wrt. N. Consider the case where $o = D$ and $co' = D'$ are observed in $2M^B$: Flipping r_1 from *unchanged* to *removed* (using the 3rd rule of v_2's CPT) improves the outcome, hence prioritized rules $br^i_H \setminus \{m_{v_1} : v_1 f_{3,1}\}$ have to be removed to restore consistency. Since this is a subset of br^i_H, D is not most preferred (see also the following theorem).

Theorem 3. *Let M_N be the CP-net transformation of an inconsistent MCS M wrt. an associated CP-net N with protected rules br_P and prioritized rules br_H. Then, $D \in D^{\pm}_{opt}(N, M)$ iff $D' \in D^{\pm}_m(M_N, br_P, br_H)$ such that $D' \cap br^i_H = br^i_H$ and $D' \cap br_M = D$.*

The techniques of meta-reasoning and prioritized diagnoses can be used to realize arbitrary preference orders on diagnoses. This is achieved by introducing a global assessment context, i.e., an observation context for all bridge rules of the original system, and (exponentially many) new prioritized bridge rules, to which the preference order is mapped. For space reasons, however, we omit details.

5 Discussion

Computational complexity: The generalized notions of diagnoses can be realized by transformations to diagnoses of an MCS, using assessment contexts. The respective bridge rules can be set up efficiently for filters and preference orders. The number of bridge rules for filters increases by at most a factor of 4 per bridge rule while for a CP-net (V, E) it is quadratic in the size of the CPTs. Then, the complexity of preferred diagnoses does not increase over that of ordinary diagnoses, if the preference assessment in the analysis contexts has not higher complexity than regular contexts.

In particular, deciding whether a given pair (D_1, D_2) of bridge rules is a prioritized diagnosis (excluding protected bridge rules) is of the same complexity as recognizing ordinary diagnoses; from the results in [8], the complexity ranges, depending on the computational complexity of contexts, from **coNP** (for **P** and **NP** contexts) to $\mathbf{D_2^P}$ (for $\mathbf{\Sigma_2^P}$contexts, e.g. disjunctive ASP contexts). Detecting subset-minimal diagnosis is $\mathbf{D^P}$-complete (for **P**, **NP**, and **coNP**contexts).

From the requirement that preferred diagnoses also be minimal follows that this complexity can not be improved, even for easily evaluable preference orders.

Decentralization: A key property of MCSs is decentralized information exchange. Filters and preference orders can be realized in such a way. If a filter or preference order is composed of local tests, then the realization can be broken down to these local, decentral tests. An example of this is the realization of CP-nets, where information is only exchanged as much as necessary to realize the CP-net.

Quantitative Assessment: As system knowledge to rank diagnoses is not always available, one may consider a quantitative inconsistency measure. Following the approach of [10], which is based on the cardinalities of the minimal inconsistent sets a certain formula belongs to, a measure on minimal inconsistent sets of bridge rules may be established. A notion of such sets is given in [8], termed inconsistency explanation, which is a pair of bridge rules (E_1, E_2) where E_1 is a minimal inconsistent set of bridge rules causing inconsistency and E_2 contains rules which could resolve the inconsistency if some were applicable.

Based on this we may define an *inconsistency measure* as follows. Let M be an MCS and $r \in br_M$, and let $A_r^i(M) = \{(E_1, E_2) \in E_m^{\pm}(M) \mid r \in E_i\}$, $i = 1, 2$ where $E_m^{\pm}(M)$ is the set of subset-minimal inconsistency explanations of M:

$$m(M, r) = \Big(\sum_{(E_1,E_2)\in A_r^1(M)} \frac{1}{|E_1|}, \sum_{(E_1,E_2)\in A_r^2(M)} \frac{1}{|E_2|} \Big).$$

Thus, $m(M, r)$ measures the inconsistency of r in M by counting the relative contribution of r to the minimal inconsistent sets in M, respectively, its contribution to resolving inconsistency. A key property of a measure is monotonicity (in the form of sub-additivity), which m is lacking in general. Finding such an inconsistency measure for MCSs faces the problem of providing a monotonic measure for a non-monotonic system. It remains to explore whether restrictions on inconsistency explanations can serve this purpose.

6 Related Work and Conclusion

Although inconsistency handling and preferences are widely used in knowledge-based systems, their application to systems interlinking different knowledge bases is still rare. In [5] argumentation context systems which equip special MCS with mediators are introduced. Mediators realize two tasks: First, they guard a context by controlling its import information. Second, they restore consistency using local information. In our approach, the first task is done by import relays of the meta-reasoning transformation, and the second task is achieved by the analyzing contexts which may be local or global as needed. This allows us not only to establish global filters and preferences, but also the implementation is local and decentralized as possible.

Bikakis et al. [2] propose a certain way of inconsistency removal which is based on local trust orders. They propose several trust-based algorithms using such orders combined with provenance; their provenance-free algorithm can be realized with our approach. In our view, information hiding is an important aspect of MCSs, which however is in conflict with provenance. Thus the goals behind the works are different.

Future work includes the development and implementation of particular filter and CP-net transformations as well as further analysis of our and other possible measures of inconsistency in MCS.

References

1. Balduccini, M., Gelfond, M.: Logic programs with consistency-restoring rules. In: International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series, pp. 9–18 (2003)
2. Bikakis, A., Antoniou, G., Hassapis, P.: Alternative strategies for conflict resolution in multi-context systems. In: AIAI, pp. 31–40 (2009)
3. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. (JAIR) 21, 135–191 (2004)
4. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI, pp. 385–390. AAAI Press, Menlo Park (2007)
5. Brewka, G., Eiter, T.: Argumentation context systems: A framework for abstract group argumentation. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 44–57. Springer, Heidelberg (2009)
6. Brewka, G., Roelofsen, F., Serafini, L.: Contextual default reasoning. In: IJCAI, pp. 268–273 (2007)
7. Domshlak, C., Brafman, R.I., Shimony, S.E.: Preference-based configuration of web page content. In: Nebel, B. (ed.) IJCAI, pp. 1451–1456. Morgan Kaufmann, San Francisco (2001)
8. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in nonmonotonic multi-context systems. In: KR (2010)
9. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. Artif. Intell. 65(1), 29–70 (1994)
10. Hunter, A., Konieczny, S.: Measuring inconsistency through minimal inconsistent sets. In: Brewka, G., Lang, J. (eds.) KR, pp. 358–366. AAAI Press, Menlo Park (2008)

A Logical Semantics for Description Logic Programs*

Michael Fink[1] and David Pearce[2]

[1] Vienna University of Technology, Austria
fink@kr.tuwien.ac.at
[2] Universidad Politcnica de Madrid, Spain
david.pearce@upm.es

Abstract. We present a new semantics for Description Logic programs [1] (dl-programs) that combine reasoning about ontologies in description logics with non-monotonic rules interpreted under answer set semantics. Our semantics is equivalent to that of [1], but is more logical in style, being based on the logic **QHT** of quantified here-and-there that provides a foundation for ordinary logic programs under answer set semantics and removes the need for program reducts. Here we extend the concept of **QHT**-model to encompass dl-programs. As an application we characterise some logical relations between dl-programs, by mating the idea of **QHT**-equivalence with the concept of query inseparability taken from description logics.

1 Introduction

Amalgamating description logics and nonmonotonic logic programs in order to combine rule-based reasoning with ontologies is a growing field of research in knowledge representation and reasoning. Its relevance stems from the aim to build powerful AI systems for Semantic Web reasoning, gradually extending the expressiveness and reasoning capabilities of their underlying formal framework. There have been several different proposals for merging description logics and logic programs into a more tightly or a more loosely integrated semantical framework. Among the best known methods are those based on stable model semantics or answer set programming (ASP); see eg., [1,2,3,4,5,6]. We shall focus on dl-programs [1] which are given as a pair $\mathcal{D} = (\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a description logic (classical) knowledge base, and $\mathcal{P}$ is a set of so-called dl-rules. Intuitively, the intended models are simply models of $\mathcal{P}$. However the rules of $\mathcal{P}$ may contain special expressions, called dl-atoms, that refer to concepts in $\mathcal{T}$. These atoms are evaluated in a candidate model for $\mathcal{P}$ by posing queries to the classical base $\mathcal{T}$.

As for ordinary ASP, the semantics of dl-programs has been defined by means of program reducts of $\mathcal{P}$. However, it is more involved, since the meaning assigned to concepts appearing in dl-atoms via $\mathcal{T}$ has to be taken into account as well, and the interpretations of the two parts of the program are to some extent distinct. While we can understand the $\mathcal{T}$ component roughly in the sense of classical logic, the answer set semantics does not associate any logic to the $\mathcal{P}$ component and thus to the dl-program

* Partially supported by the MCICINN projects TIN2006-15455, TIN2009-14562-CO5, and CSD2007-00022, as well as by the WWTF project ICT08-020.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 156–168, 2010.
© Springer-Verlag Berlin Heidelberg 2010

as a whole. This is clearly an obstacle to studying intertheory relations and modularity properties that are relevant for applications. It is therefore useful to try to reformulate the answer set semantics of dl-programs in a style that is closer to ordinary logical semantics.

Fortunately there is a suitable logical foundation for ASP. Answer sets can be understood as minimal models in an ordinary, monotonic logic: the logic of *here-and-there*. In first-order form this logic, called the *quantified logic of here-and-there*, in symbols **QHT**, provides a foundation for non-ground answer set programs [7]. In **QHT** one can define a notion of minimal model, called *equilibrium model* [8], that exactly corresponds to answer sets. The logic associated with just these minimal models is known as *equilibrium logic*. An important feature of **QHT** is that equivalence in this logic is a necessary and sufficient condition for two programs or nonmonotonic theories to be strongly equivalent, meaning that they are inter-substitutable without loss in all contexts [15]. We may call this the strong equivalence property.

If hybrid theories like dl-programs are to become a successful, practical tool in knowledge-based reasoning, we need to study how ontologies and rules can be combined in a modular fashion. Knowing for instance in which contexts one hybrid theory can be replaced by another without loss is important for formalising knowledge and for transforming and simplifying theories. There has already been a strong interest recently in developing logical treatments of modularity for ontologies reconstructed in description logics (DL). An approach based on conservative extensions and entailment and difference concepts can be found in [9,10,11]. On the other hand, in ASP, work on (strong) equivalence relations between programs began already in [12], and the study of variations of this basic concept has formed a very active area of research since, especially as a tool for program transformation and optimisation. Recently focus has turned from propositional programs to theories and programs in first-order logic [13,14,15]. This is important for the study of dl-programs where first-order languages are needed.

The aim of this paper is to provide a more logical style of semantics for dl-programs by extending the concepts of **QHT**-model and equilibrium model to embrace dl-rules. This helps to make the semantics simpler and more uniform. As an illustration of its use we consider ways to define and study strong forms of equivalence between programs that may be useful for combining ontologies and rules in a modular fashion. Briefly:

- We formulate different, logical semantics for dl-programs using **QHT**-models, removing the need for program reducts.
- We combine the idea of **QHT**-equivalence with the concept of query inseparability and apply the new semantics to characterise different notions of equivalence between dl-programs.
- Besides strong and weak answer set semantics for dl-programs, we define an alternative semantics which precisely captures the semantics of dl-programs realised as HEX programs, ie., under the so-called *FLP-reduct* [16].

In the next section we introduce necessary background, before characterising dl-program semantics by means of **QHT** in Section 3. We study equivalence concepts for dl-programs in Section 4, followed by a discussion of extensions to dl-programs under HEX semantics and conclusions (Sections 5 and 6).

2 Preliminaries

Quantified Equilibrium Logic. In this paper we restrict attention to function-free languages with a single negation symbol, '$\neg$', working with a quantified version of the logic *here-and-there*. In other respects we follow the treatment of [17,7]. We consider first-order languages $\mathcal{L} = \langle C, P\rangle$ built over a set of *constant* symbols, C, and a set of *predicate* symbols, P. The sets of $\mathcal{L}$-terms, ground $\mathcal{L}$-terms, $\mathcal{L}$-formulas, $\mathcal{L}$-sentences and atomic $\mathcal{L}$-sentences are defined in the usual way. If D is a non-empty set of domain constants, we denote by $At(D, P)$ the set of ground atomic sentences of $\langle D, P\rangle$. By an $\mathcal{L}$-interpretation I over a set D we mean a subset of $At(D, P)$. A *classical* $\mathcal{L}$-structure can be regarded as a tuple $\mathcal{M} = \langle (D, \sigma), I\rangle$ where I is an $\mathcal{L}$-interpretation over D and $\sigma\colon C \cup D \to D$ is a mapping, called the *assignment*, such that $\sigma(d) = d$ for all $d \in D$. If $D = C$ and $\sigma = id$, $\mathcal{M}$ is an *Herbrand structure*.

A *here-and-there* $\mathcal{L}$-structure with static domains, or $\mathbf{QHT}(\mathcal{L})$*-structure*, is a tuple $\mathcal{M} = \langle (D, \sigma), I_h, I_t\rangle$ where $\langle (D, \sigma), I_h\rangle$ and $\langle (D, \sigma), I_t\rangle$ are classical $\mathcal{L}$-structures such that $I_h \subseteq I_t$. We can think of a here-and-there structure $\mathcal{M}$ as similar to a first-order classical model, but having two parts, or components, h and t, that correspond to two different points or "worlds", 'here' and 'there', in the sense of Kripke semantics for intuitionistic logic [18], where the worlds are ordered by $h \leq t$. At each world $w \in \{h, t\}$ one verifies a set of atoms I_w in the expanded language for the domain D. We call the model static, since, in contrast to say intuitionistic logic, the same domain serves each of the worlds. Since $h \leq t$, whatever is verified at h remains true at t. The satisfaction relation for $\mathcal{M}$ is defined so as to reflect the two different components, so we write $\mathcal{M}, w \models \varphi$ to denote that φ is true in $\mathcal{M}$ with respect to the w component. The recursive definition of the satisfaction relation forces us to consider formulas from $\langle C \cup D, P\rangle$. Evidently we should require that an atomic sentence is true at w just in case it belongs to the w-interpretation. Formally, if $p(t_1, \ldots, t_n) \in At(C \cup D, P)$, r and s are $\mathcal{L}$-terms, and $w \in \{h, t\}$ then

$$\begin{aligned} \mathcal{M}, w \models p(t_1, \ldots, t_n) \quad &\text{iff} \quad p(\sigma(t_1), \ldots, \sigma(t_n)) \in I_w. \\ \mathcal{M}, w \models r = s \quad &\text{iff} \quad \sigma(r) = \sigma(s) \end{aligned}$$

The second clause means that our semantics satisfies the axiom of "decidible equality"

$$\forall x \forall y (x = y \vee x \neq y).$$

Then $\models$ is extended recursively using the usual Kripke truth conditions for $\wedge, \vee, \rightarrow, \neg, \forall, \exists$ in intuitionistic logic bearing in mind our assumptions about the two worlds h and t and the single domain D, see eg. [18,7].

Truth of a sentence in a model is defined as follows: $\mathcal{M} \models \varphi$ iff $\mathcal{M}, w \models \varphi$ for each $w \in \{h, t\}$. In a model $\mathcal{M}$ we also use the symbols H and T, possibly with subscripts, to denote the interpretations I_h and I_t respectively; so, an $\mathcal{L}$-structure may be written in the form $\langle U, H, T\rangle$, where $U = (D, \sigma)$. A structure $\langle U, H, T\rangle$ is called *total* if $H = T$, whence it is equivalent to a classical structure.

The resulting logic is called *Quantified Here-and-There Logic with static domains and decidable equality*, and denoted in [15] by $\mathbf{SQHT}^{=}$, where a complete axiomatisation can be found. To simplify notation we drop the labels for static domains and

equality and refer to this logic simply as quantified here-and-there, **QHT**. Quantified equilibrium logic, or QEL, is based on a suitable notion of minimal model.

Definition 1. *Among* **QHT***-structures over a given language we define the order* $\trianglelefteq$ *by:* $\langle (D,\sigma), H, T\rangle \trianglelefteq \langle (D',\sigma'), H', T'\rangle$ *if* $D = D'$, $\sigma = \sigma'$, $T = T'$ *and* $H \subseteq H'$. *If the subset relation is strict, we write '*$\triangleleft$*'. Let* Γ *be a set of sentences and* $\mathcal{M} = \langle (D,\sigma), H, T\rangle$ *a model of* Γ. $\mathcal{M}$ *is said to be an* equilibrium *model of* Γ *if it is minimal under* $\trianglelefteq$ *among models of* Γ*, and it is total.*

Answer sets. We assume the reader is familiar with the usual definitions of answer set based on *Herbrand* models and ground programs, eg. [19]. Two variations of this semantics, the open [20] and generalised open answer set [5] semantics, consider non-ground programs and open domains, thereby relaxing the standard name assumption. In addition, [21] offers a very general concept of stable model for arbitrary first-order formulas, defining the property of being a stable model syntactically via a second-order condition.

The correspondence between QEL and answer set semantics is by now quite well known and has been described in several works (see [22,17,7,21]). By the usual convention, when $\mathcal{P}$ is a logic program with variables we consider the models of its universal closure expressed as a set of logical formulas. It follows that if $\mathcal{P}$ is a logic program (of any form), a total **QHT** model $\langle U, T, T\rangle$ of $\mathcal{P}$ is an equilibrium model of $\mathcal{P}$ iff it is a stable model of $\mathcal{P}$ in the sense of [21]. Moreover, two logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *strongly equivalent* iff they coincide on their **QHT**-models. Placing additional restrictions on **QHT** models, we obtain a correspondence to other notions of answer set such as those based on a standard name assumption.

DL-programs. Turning to dl-programs [1,23], we start without restricting the syntax of the classical part or the knowledge base that is combined with logic program rules; later on we shall consider some concepts and properties that apply to dl-programs based on (particular) description logics (for a background and corresponding notation used cf. [24]). In other words, we consider arbitrary function-free first-order theories that are combined with dl-rules and, for the moment, we allow for arbitrary formulas as queries in dl-atoms. Moreover, disjunction is allowed in rule heads, while we require that the classical theory and the logic program share a single set of constants.

More formally, let $\mathcal{L}_\mathcal{T} = \langle C, P_\mathcal{T}\rangle$ and $\mathcal{L}_\mathcal{P} = \langle C, P_\mathcal{P}\rangle$ be function-free first-order languages, such that $P_\mathcal{T} \cap P_\mathcal{P} = \emptyset$. Symbols in $P_\mathcal{T}$, respectively in $P_\mathcal{P}$, are called *classical predicates* and *rule predicates*, respectively. A *dl-atom* is of the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](t_1, \ldots, t_n), \tag{1}$$

where $S_i \in P_\mathcal{T}$ and $p_i \in P_\mathcal{P}$ are k-ary predicate symbols, $op_i \in \{\uplus, \sqcup, \sqcap\}$, Q is an n-ary classical predicate or a formula in $\mathcal{L}_\mathcal{T}$ with n free variables, and $t_1, \ldots, t_n$ are terms. A *dl-rule* is like a logic program rule of the form

$$b_1 \wedge \ldots \wedge b_m \wedge \neg b_{m+1} \wedge \ldots \wedge \neg b_n \rightarrow h_1 \vee \ldots \vee h_l \tag{2}$$

with the restriction that head atoms $h_1, \ldots, h_l$ are equality-free atoms of $\mathcal{L}_\mathcal{P}$, and body atoms $b_1, \ldots, b_n$ are either atoms of $\mathcal{L}_\mathcal{P}$ or dl-atoms. The positive body $\{b_1, \ldots, b_m\}$

and the negative body $\{b_{m+1}, \ldots, b_n\}$ of a dl-rule r are denoted by $B^+(r)$ and $B^-(r)$, respectively. The expression $h_1 \vee \ldots \vee h_l$ is abbreviated by $Hd(r)$. A *dl-program* over $\mathcal{L} = \langle C, P_\mathcal{T} \cup P_\mathcal{P} \rangle$ is a pair $\mathcal{D} = (\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a finite first-order theory over $\mathcal{L}_\mathcal{T}$ and $\mathcal{P}$ is a set of dl-rules.

Example 1. Consider the following vocabulary dealing with wine: constants frb and ldm are used for '*Freixenet Brut*' and '*Lambrusco di Modena*', respectively; the classical predicates $W(x)$, $R(x)$, $S(x)$, and $L(x)$, represent the concepts of *White Wine*, *Red Wine*, *Sparkling Wine*, and *Lambrusco*; $w(x)$, $r(x)$, $s(x)$, and $l(x)$ are rule predicates intended to reason about the above concepts in rules; an additional rule predicate $sc(x)$ encodes whether a wine is *served cold*.

Now, let $(\mathcal{T}, \mathcal{P})$ be the following dl-program over this vocabulary.

$\mathcal{T}$	$\mathcal{P}$	
$L \sqsubseteq R \sqcap S$	$l(ldm)$	$sc(X) \vee \neg sc(X)$
$\neg W \sqcap \neg R \sqsubseteq \bot$	$s(frb)$	$l(X) \rightarrow \neg sc(X)$
$R \sqcap W \sqsubseteq \bot$	$DL[S \uplus s, L \uplus l; R](X) \rightarrow r(X)$	$\neg r(X) \rightarrow w(X)$

Intuitivey, the dl-rule says: in $\mathcal{T}$ add to S the contents of s and add to L the contents of l; if $R(X)$ now follows (in the enlarged $\mathcal{T}$), then $r(X)$. □

Turning to the formal semantics of dl-programs, let us denote the set of dl-atoms in a rule r, respectively in a set of rules $\mathcal{P}$, by $DL(r)$ and $DL(\mathcal{P})$, respectively, and let $\models_c$ denote classical entailment.

An Herbrand structure $\mathcal{M} = \langle U, I \rangle$ (with $U = (D, \sigma)$) is a *model* of a literal l under $\mathcal{T}$ if $l \in I$. It is a model of a ground dl-atom of the form (1) under $\mathcal{T}$ if $\mathcal{T} \cup \bigcup_{i=1}^{m} A_i(I) \models_c Q(t_1, \ldots, t_n)$, where

- $A_i(I) = \{S_i(\boldsymbol{e}) \mid p_i(\boldsymbol{e}) \in I\}$, for $op_i = \uplus$,
- $A_i(I) = \{\neg S_i(\boldsymbol{e}) \mid p_i(\boldsymbol{e}) \in I\}$, for $op_i = \uplus$,
- $A_i(I) = \{\neg S_i(\boldsymbol{e}) \mid p_i(\boldsymbol{e}) \notin I\}$, for $op_i = \cap$,

and $\boldsymbol{e} = e_1, \ldots, e_n$ are ground terms.

As usual, $\mathcal{M}$ is a *model* of a ground dl-rule r under $\mathcal{T}$ if $\mathcal{M}$ is a model of some $h_i \in \{h_1, \ldots, h_l\}$ under $\mathcal{T}$, whenever $\mathcal{M}$ is a model of all $b_i \in \{b_1, \ldots, b_m\}$ under $\mathcal{T}$ and it is no model of any $b_i \in \{b_{m+1}, \ldots, b_n\}$ under $\mathcal{T}$. $\mathcal{M}$ is a model of a dl-program $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ if $\mathcal{M}$ is a model of every $r \in gr_U(\mathcal{P})$ under $\mathcal{T}$.

Furthermore, given a dl-program $\mathcal{D} = (\mathcal{T}, \mathcal{P})$, the *weak dl-transform* of $\mathcal{P}$ relative to $\mathcal{T}$ and a model $\mathcal{M}$ of $\mathcal{P}$, denoted $w\mathcal{P}^\mathcal{M}_\mathcal{T}$, is the logic program obtained from $gr_U(\mathcal{P})$ by deleting

- each $r \in gr_U(\mathcal{P})$ such that either $\mathcal{M}$ is not a model of some $\alpha \in B^+(r) \cap DL(r)$, or a model of some $\alpha \in B^-(r)$, and
- all literals in $B^-(r) \cup (B^+(r) \cap DL(r))$ from each remaining $r \in gr_U(\mathcal{P})$.

If $\mathcal{M}$ is an answer set of the logic program $w\mathcal{P}^\mathcal{M}_\mathcal{T}$, then $\mathcal{M}$ is a *weak answer set* of $\mathcal{D}$.

Now assume that, $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ has an associated set of ground dl-atoms $DL^+(\mathcal{P})$ known to be monotonic, and for any ground rule r, let $DL^?(r) = DL(r) \setminus DL^+(\mathcal{P})$. The *strong dl-transform* of $\mathcal{P}$ relative to $\mathcal{T}$ and a model $\mathcal{M}$ of $\mathcal{P}$, denoted $s\mathcal{P}^\mathcal{M}_\mathcal{T}$, is the logic program obtained from $gr_U(\mathcal{P})$ as before replacing $DL(r)$ by $DL^?(r)$. If $\mathcal{M}$ is the least model of $(\mathcal{T}, s\mathcal{P}^\mathcal{M}_\mathcal{T})$, then $\mathcal{M}$ is a *strong answer set* of $\mathcal{D}$.

3 Logical Semantics

We reformulate the semantics for dl-programs in a style that is closer to ordinary logical semantics and in particular to the logic **QHT**. This makes it easier to characterise logical properties of dl-programs and relations between them.

Dl-atoms and rules are defined as above in (1), (2). We use the usual semantics for **QHT**, so the truth conditions for ordinary atoms, conjunctions, disjunctions, negation and implications in a model $\mathcal{M} = \langle U, H, T\rangle$ are the same as before. For dl-atoms we define three semantics, the last two of which correspond to weak and strong answer sets respectively. Informally these semantics work as follows. The truth of a dl-atom (1) is checked as before by inspecting whether the query Q follows classically from a certain extension of the theory $\mathcal{T}$. The difference is that, as a base model for computing the A_i, as well as for defining the truth of a dl-atom, we now use a **QHT** model instead of a classical Herbrand model. This allows a more uniform treatment of the different operators. We begin with a semantics that corresponds to a variation of strong answer sets.

Definition 2 (models of dl-atoms). *Let α be a ground dl-atom of the form (1) and let $\mathcal{M} = \langle U, H, T\rangle$ be a* **QHT** *structure. Then, $\mathcal{M}$ is a* model *of α under $\mathcal{T}$ iff $\mathcal{M}, w \models \alpha$ for $w = h, t$; where $\mathcal{M}, w \models \alpha$ iff $\mathcal{T} \cup \bigcup_{i=1}^{m} A_i(w) \models_c Q(t_1, \ldots, t_n)$, where*

- $A_i(w) = \{S_i(\boldsymbol{e}) \mid \mathcal{M}, w \models p_i(\boldsymbol{e})\}$, *for* $op_i = \uplus$,
- $A_i(w) = \{\neg S_i(\boldsymbol{e}) \mid \mathcal{M}, w \models p_i(\boldsymbol{e})\}$, *for* $op_i = \cup$,
- $A_i(w) = \{\neg S_i(\boldsymbol{e}) \mid \mathcal{M}, w \models \neg p_i(\boldsymbol{e})\}$, *for* $op_i = \cap$,

and $\boldsymbol{e} = e_1, \ldots, e_n$ are ground terms.

Definition 3 (weak models of dl-atoms). *Let α be a ground dl-atom of the form (1) and let $\mathcal{M} = \langle U, H, T\rangle$ be a* **QHT** *structure. Then we say that $\mathcal{M}$ is a* weak model *of α under $\mathcal{T}$ iff $\mathcal{M}, w \models \alpha$ for $w = h, t$; where $\mathcal{M}, t \models \alpha$ is defined as in the semantics of Definition 2 and $\mathcal{M}, h \models \alpha \Leftrightarrow \mathcal{M}, t \models \alpha$.*

Observe that now (U, T) need not be an Herbrand model. Notice that in the first semantics operators are evaluated at both worlds h and t in the model, while in the second, weak semantics they are essentially evaluated only at t which then determines the value at h.

Finally we introduce a variant of the first semantics that corresponds to strong answer sets. For this we need to distinguish between atoms known to be monotonic and others. As before we use the symbols $DL^{+}(\mathcal{P})$ and $DL^{?}(\mathcal{P})$ for these. Let us adopt the convention that all atoms containing an occurrence of the operator $op_i = \cap$ belong to $DL^{?}(\mathcal{P})$, while all others are in $DL^{+}(\mathcal{P})$.

Definition 4 (strong models of dl-atoms). *Let α be a ground dl-atom of the form (1) and let $\mathcal{M} = \langle U, H, T\rangle$ be a* **QHT** *structure. Then we say that $\mathcal{M}$ is a* strong model *of α under $\mathcal{T}$ iff $\mathcal{M}, w \models \alpha$ for $w = h, t$; where for all atoms α, $\mathcal{M}, t \models \alpha$ is defined as in the semantics of Definition 2, while $\mathcal{M}, h \models \alpha$ is defined as in the semantics of Definition 2 if $\alpha \in DL^{+}(\mathcal{P})$, and as in Definition 3, ie. by $\mathcal{M}, h \models \alpha \Leftrightarrow \mathcal{M}, t \models \alpha$, otherwise.*

A dl-rule r is true in a model $\mathcal{M}$ under $\mathcal{T}$, in symbols $\mathcal{M} \models_{\mathcal{T}} r$, if the rule is satisfied according to the usual **QHT** semantics. We may suppress the subscript $\mathcal{T}$ if the context is clear. The following property is important but easy to verify.

Proposition 1 (persistence). *For any model $\mathcal{M}$ and rule r, $\mathcal{M}, h \models r \Rightarrow \mathcal{M}, t \models r$, for each of the semantics.*

The notions of model (resp. weak and strong model) and equilibrium model (resp. weak, strong equilibrium model) are now defined in the obvious way.

Definition 5. *A* **QHT** *structure $\mathcal{M} = \langle U, H, T\rangle$ is a* model *(resp.* weak model, strong model*) of a dl-program $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ if $\mathcal{M} \models_{\mathcal{T}} r$ for each $r \in \mathcal{P}$ under the semantics (resp. the weak, the strong semantics) for dl-atoms. It is said to be an* equilibrium model *(resp.* weak, strong equilibrium model*) of $\mathcal{D}$ if $H = T$ and $\mathcal{M}$ is a minimal model (resp. weak, strong model) of $\mathcal{P}$ under $\mathcal{T}$ wrt $\trianglelefteq$, ie. there is no model of $\mathcal{D}$ (resp. weak model, strong model of $\mathcal{D}$) of the form $\langle U, H', T\rangle$ where H' is a proper subset of H.*

For reasons of space we do not give a detailed proof of our main theorem, Proposition 2 below, which established the correctness of our semantics. However the proof is based on the following two lemmas which are fairly routine. We formulate for the case of strong models; similar properties hold for weak models.

Lemma 1. *Let $\mathcal{M} = \langle U, H, T\rangle$ be a* **QHT** *strong Herbrand model of $\mathcal{P}$ under $\mathcal{T}$. Then $\langle H, T\rangle \models s\mathcal{P}_{\mathcal{T}}^{\mathcal{M}}$.*

Lemma 2. *Let $\mathcal{M} = \langle U, H, T\rangle$ be a* **QHT** *strong Herbrand model of $\mathcal{P}$ under $\mathcal{T}$. Then $\mathcal{M}$ is a minimal model of $\mathcal{P}$ under $\mathcal{T}$ wrt $\trianglelefteq$ if and only if $\langle U, H\rangle$ is a minimal model of $(\mathcal{T}, s\mathcal{P}_{\mathcal{T}}^{\mathcal{M}})$.*

From these properties we can derive:

Proposition 2. *A total Herbrand* **QHT** *structure $\mathcal{M} = \langle U, T, T\rangle$ is a weak (resp. strong) equilibrium model of a dl-program $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ iff $\langle U, T\rangle$ is a weak (resp. strong) answer set of $\mathcal{D}$.*

Example 2. Reconsider $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ from Example 1 with universe $U = (\{frb, ldm\}, id)$. The structures $\mathcal{M} = \langle U, T, T\rangle$ and $\mathcal{M} = \langle U, T', T'\rangle$, where $T = \{s(frb), l(ldm), w(frb), r(ldm)\}$ and $T' = T \cup \{sc(frb)\}$, are weak and strong equlibrium models of $\mathcal{D}$ (note that the only dl-atom is monotone, and that, for every $\{l(ldm)\} \subseteq H \subset T'$, the dl-atom is true for ldm). They are also weak, as well as strong, answer sets of $\mathcal{D}$. □

Although the alternative semantics is therefore equivalent to the original one, there are several features worth emphasising. First, since we have removed the need for reducts, we can extend the semantics to more general types of rules and formulas just using the usual truth conditions for **QHT** models.[1] Secondly, although we shall consider here just the usual dl-programs with Herbrand models, our semantics is not limited to this and we could in principle consider non-Herbrand interpretations, as in the case of hybrid knowledge bases. Thirdly, we now have a more homogeneous and logical semantics that may help us derive logical properties of dl-programs.

Finally, an advantage of the first semantics is that, by using **QHT** structures, we do not have to distinguish semantically between monotone and possibly non-monotone

[1] In principle we could extend the syntax of rules r to any formula providing that Proposition 1 continues to hold.

operators. All operators are treated similarly. The difference between models and weak models is merely that the former evaluate dl-atoms by looking only at the t-world. Notice that although we apply the words 'weak' and 'strong' to models, these labels are really used to reflect the difference between weak and strong equilibrium models or answer sets. For example, while every strong equilibrium model is also a weak one, not every strong model (or ordinary model) need be a weak one. Observe that if all dl-atoms containing $op_i = \uplus$ are 'pure', in the sense that they do not contain occurrences of $\uplus$ or $\cup$, then the first semantics and the strong semantics coincide.

4 Equivalence Concepts

To illustrate the use of the new semantics, we introduce and study some concepts of equivalence between dl-programs. We can consider different equivalence relations between dl-programs according to how the different components, $\mathcal{T}$ and $\mathcal{P}$, are allowed to vary. If $\mathcal{D} = (\mathcal{T}, \mathcal{P})$ is a dl-program, $\mathcal{T}$' is a classical theory and $\mathcal{P}$' is a set of dl-rules, then $\mathcal{D} \cup \mathcal{T}'$ stands for the program $(\mathcal{T} \cup \mathcal{T}', \mathcal{P})$ and $\mathcal{D} \cup \mathcal{P}'$ stands for the program $(\mathcal{T}, \mathcal{P} \cup \mathcal{P}')$.

Definition 6 (Equivalence for dl-programs). *Two dl-programs $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be* equivalent *if they have the same equilibrium models, they are $\mathcal{T}$-*equivalent *if $\mathcal{D}_1 \cup \mathcal{T}$ and $\mathcal{D}_2 \cup \mathcal{T}$ are equivalent for any $\mathcal{T}$, they are $\mathcal{P}$-*equivalent *if $\mathcal{D}_1 \cup \mathcal{P}$ and $\mathcal{D}_2 \cup \mathcal{P}$ are equivalent for any $\mathcal{P}$, and they are* strongly equivalent *if $\mathcal{D}_1 \cup \mathcal{T} \cup \mathcal{P}$ and $\mathcal{D}_2 \cup \mathcal{T} \cup \mathcal{P}$ are equivalent for any $\mathcal{T}$ and $\mathcal{P}$.*

Having the same equilibrium models is to be understood under any of the given semantics. However, unless our results are specific to one semantics, we don't further specify which one. We also say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are **QHT**-equivalent if they have the same **QHT** models (in any of the given senses). Lastly, it is useful to introduce relativised versions of these concepts. Thus, if Σ is a signature or vocabulary and $\mathcal{P}$ is a set of dl-rules, we say that $\mathcal{P}$ is a set of Σ-dl-rules if all classical predicates appearing in any dl-atom are from Σ.

Definition 7 (Σ-equivalence for dl-programs). *Given a signature Σ, two dl-programs $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be Σ-$\mathcal{T}$-*equivalent *if $\mathcal{D}_1 \cup \mathcal{T}$ and $\mathcal{D}_2 \cup \mathcal{T}$ are equivalent for any theory $\mathcal{T}$ in Σ, they are Σ-$\mathcal{P}$-*equivalent *if $\mathcal{D}_1 \cup \mathcal{P}$ and $\mathcal{D}_2 \cup \mathcal{P}$ are equivalent for any set of Σ-dl-rules $\mathcal{P}$, and they are* strongly Σ-equivalent *if $\mathcal{D}_1 \cup \mathcal{T} \cup \mathcal{P}$ and $\mathcal{D}_2 \cup \mathcal{T} \cup \mathcal{P}$ are equivalent for any $\mathcal{T}$ and $\mathcal{P}$, such that $\mathcal{T}$ in Σ and $\mathcal{P}$ is a set of Σ-dl-rules.*

A first, simple observation is that if two ordinary answer set programs are strongly equivalent they cannot be separated by additional dl-rules.

Proposition 3. *Let Π_1, Π_2 be two strongly equivalent logic programs. Let $\mathcal{R}$ be any set of dl-rules and let $(\mathcal{T}, \mathcal{P}_1), (\mathcal{T}, \mathcal{P}_2)$ be dl-programs where $\mathcal{P}_1 = \Pi_1 \cup \{\mathcal{R}\}$ and $\mathcal{P}_2 = \Pi_2 \cup \{\mathcal{R}\}$. Then $(\mathcal{T}, \mathcal{P}_1)$ and $(\mathcal{T}, \mathcal{P}_2)$ are equivalent under all the given semantics for dl-programs.*

This simple observation can be generalised. Notice that we keep $\mathcal{T}$ fixed in each case since otherwise a given rule $r \in \mathcal{R}$ could have a completely different interpretation in one of the extended dl-programs than it does in the other.

Proposition 4. *Two dl-programs, $(\mathcal{T}, \mathcal{P}_1)$ and $(\mathcal{T}, \mathcal{P}_2)$, are $\mathcal{P}$-equivalent (under a given semantics) if and only if they are* $\mathbf{QHT}$*-equivalent (under the same semantics).*

Proof (Sketch). For the 'if' direction the argument is the same as for Proposition 3: if $(\mathcal{T}, \mathcal{P}_1)$ and $(\mathcal{T}, \mathcal{P}_2)$ have the same $\mathbf{QHT}$ models, then, whatever set of dl-rules $\mathcal{R}$ that is added to them will yield the same set of $\mathbf{QHT}$ models in each case, and hence the same equilibrium models. For the 'only if' direction we can use the proofs of strong equivalence theorems found in [15]. The only additional property we need to check for the case of dl-rules is that if $\mathcal{M} = \langle U, H, T\rangle$ is a $\mathbf{QHT}$ model of a program $\mathcal{P}$ under $\mathcal{T}$, then $\mathcal{M} = \langle U, T, T\rangle$ is also a $\mathbf{QHT}$ model of $\mathcal{P}$ under $\mathcal{T}$. But this is guaranteed by the persistence property stated in Proposition 1. □

We now turn to the case of a varying knowledge base. To deal with the situation where $\mathcal{T}$ is allowed to vary, we consider an equivalence concept drawn from the area of ontologies reconstructed in description logics (DL). We assume the reader is familiar with the standard notions of TBox and ABox (see eg. the following references). In the papers [9,10,11] on modular ontologies there are several slightly different terminologies and notations. However, basically these works consider an ontology to be represented by a TBox, while a knowledge base is a combination of a TBox together with an ABox. We state here a definition from [11,25]. To simplify notation we assume that some DL is given, while Σ is a vocabulary or signature.[2] Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be TBoxes.

Definition 8. *The* Σ-query difference *between $\mathcal{T}_1$ and $\mathcal{T}_2$, in symbols* $\mathrm{Diff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$*, is the set of pairs $(\mathcal{A}, Q(x))$ where $\mathcal{A}$ is an ABox and $Q(x) \in \Sigma$ is a query such that $(\mathcal{T}_1, \mathcal{A}) \not\models_c Q(\mathbf{a})$ and $(\mathcal{T}_2, \mathcal{A}) \models_c Q(\mathbf{a})$, for some tuple $\mathbf{a}$ of object names from $\mathcal{A}$. We say that $\mathcal{T}_1$ Σ-query entails $\mathcal{T}_2$ if $\mathrm{Diff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \emptyset$. Furthermore we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are* Σ-query inseparable *if each Σ-query entails the other.*

In other words, query inseparability means equivalence for all ABoxes and Σ-queries. Let us turn to dl-programs and let us suppose for the moment that their classical part comprises an ontology or TBox, so a dl-program has the form $(\mathcal{T}, \mathcal{P})$ for some TBox, $\mathcal{T}$. Now the way in which a ground dl-atom is evaluated in an Herbrand interpretation $\mathcal{M}$ is similar to the effect of adding an ABox $\mathcal{A}$ to $\mathcal{T}$ and then checking whether a ground query $Q(\mathbf{a})$ follows from $(\mathcal{T}, \mathcal{A})$. This yields the following property. From now on we make the assumption that the same syntactic class of queries is allowed in each case of TBoxes and dl-programs, for example arbitrary queries, conjunctive queries or some intermediate class.[3]

Proposition 5. *Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-query inseparable TBoxes, and let P be any set of Σ-dl-rules. Then the dl-programs $(\mathcal{T}_1, P)$ and $(\mathcal{T}_2, P)$ are equivalent.*

In order to explore the notion of Σ-$\mathcal{T}$-equivalence we can make use of the concept of *strong* query entailment from [11]. The strong Σ-*query difference* between $\mathcal{T}_1$ and

[2] For [11,25] the signature does not include constant symbols.

[3] In general the concept of query inseparability depends not only on the vocabulary Σ but also on the given query language or syntax; different ones have been considered in the literature. To save space we leave this variable implicit and merely suppose that the query language operating over $\mathcal{T}$ in the DL is the same one that is used for evaluating dl-atoms in the dl-program.

$\mathcal{T}_2$, in symbols $\text{sDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$, is the set of triples $(\mathcal{T}, \mathcal{A}, q(x))$ such that $\mathcal{T}$ is a Σ-TBox and $(\mathcal{A}, q(x)) \in \text{Diff}_\Sigma(\mathcal{T}_1 \cup \mathcal{T}, \mathcal{T}_2 \cup \mathcal{T})$. Then $\mathcal{T}_1$ *strongly* Σ-query entails $\mathcal{T}_2$ if $\text{sDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \emptyset$, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are *strongly* Σ-query *inseparable* if each strongly Σ-query entails the other. Moreover, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *strongly query inseparable* if they are Σ-query inseparable for any Σ. This leads to the following observation by an obvious extension to the argument for Proposition 5.

Proposition 6. *Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are strongly Σ-query inseparable TBoxes, and let P be any set of Σ-dl-rules. Then $(\mathcal{T}_1, P)$ and $(\mathcal{T}_2, P)$ are Σ-$\mathcal{T}$-equivalent.*

An interesting result of [11] is that in some DLs, such as *DL-Lite*$_{bool}$, query and strong query inseparability coincide and are equivalent to the notion of strong concept inseparability (also defined there). In that case we would have the consequence that if $\mathcal{T}_1$ $\mathcal{T}_2$ are Σ-query inseparable then the dl-programs $(\mathcal{T}_1, \mathcal{P})$ and $(\mathcal{T}_2, \mathcal{P})$ are Σ-$\mathcal{T}$-equivalent.

By combining ideas from Propositions 4 and 5 we can obtain some sufficient conditions for Σ-$\mathcal{P}$-equivalence under varying TBoxes.

Proposition 7. *Let $\mathcal{D}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{D}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ be* **QHT***-equivalent dl-programs where $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-query inseparable TBoxes. Then $\mathcal{D}_1$ and $\mathcal{D}_2$ are Σ-$\mathcal{P}$-equivalent.*

Analogous to Proposition 6, we obtain a 'strong' version of Proposition 7 by replacing Σ-query inseparability by strong Σ-query inseparability.

Proposition 8. *Let $\mathcal{D}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{D}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ be* **QHT***-equivalent dl-programs where $\mathcal{T}_1$ and $\mathcal{T}_2$ are strongly Σ-query inseparable TBoxes. Then the dl-programs $\mathcal{D}_1$ and $\mathcal{D}_2$ are strongly Σ-equivalent.*

The following corollary that drops reference to Σ is straightforward. It also generalises Proposition 4.

Corollary 1. *Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are strongly query inseparable. Then $(\mathcal{T}_1, \mathcal{P}_1)$ and $(\mathcal{T}_2, \mathcal{P}_2)$ are strongly equivalent iff they are* **QHT***-equivalent.*

Further generalisations may be possible by applying the concept of relativised program equivalence, but we leave this for future work.

To illustrate the above concepts, let us consider a simple example.

Example 3. Using the vocabulary of Example 1, let $\mathcal{D}_1 = (\mathcal{T}_1, \mathcal{P}_1)$ and $\mathcal{D}_2 = (\mathcal{T}_2, \mathcal{P}_2)$ be dl-programs given by:

$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{P}'$
$L \sqsubseteq R \sqcap S$		$l(ldm)$
$\neg W \sqcap \neg R \sqsubseteq \bot$		$s(frb)$
$R \sqcap W \sqsubseteq \bot$	$R \sqsubseteq \neg W$	$sc(X) \vee \neg sc(X)$
	$W \sqsubseteq \neg R$	$l(x) \rightarrow \neg sc(x)$

and $\mathcal{P}_1 = \mathcal{P}' \cup \{r_{1_1}, r_{2_1}, r_3\}$, $\mathcal{P}_2 = \mathcal{P}' \cup \{r_{1_2}, r_{2_2}, r_3\}$, where $r_{1_1} = w(X) \rightarrow sc(X)$, $r_{1_2} = w(X) \wedge \neg sc(X) \rightarrow \bot$, $r_3 = DL[S \uplus s, L \uplus l; S](X) \wedge \neg r(X) \rightarrow w(X)$, $r_{2_1} = DL[S \uplus s, L \uplus l; R](X) \rightarrow r(X)$, $r_{2_2} = DL[S \uplus s, L \uplus l; \neg W](X) \rightarrow r(X)$.

Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are TBoxes in *DL-Lite*$_{bool}$. If Σ is the classical language as given in Example 1, then $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-query equivalent and therefore strongly Σ-query equivalent by the results of [11]. Moreover, $\mathcal{D}_1$ and $\mathcal{D}_2$ are **QHT**-equivalent. To

see the latter, first observe that the ordinary rules in each of the programs are strongly equivalent. Secondly, the dl-atom $DL[S \uplus s, L \uplus l; R](X)$ has the same models under $\mathcal{T}_1$ as the dl-atom $DL[S \uplus s, L \uplus l; \neg W](X)$ has under $\mathcal{T}_2$, because in both theories the concepts R and $\neg W$ are equivalent. Therefore, $\mathcal{D}_1$ and $\mathcal{D}_2$ are strongly equivalent. □

5 HEX Programs

Another type of hybrid theory, called HEX program, was introduced in [6]. This combines answer set programs with higher-order atoms and external atoms. In particular, the external atoms can refer, as in dl-programs, to concepts belonging to a classical knowledge base or ontology. In such a case one can compare the semantics of the HEX program with that of the corresponding dl-program. Although both are based on answer sets, the two semantics are only partially in agreement. Specifically, as shown in [6], they agree on programs all of whose external atoms (dl-atoms) contain only monotone operators. Then, the answer sets of the HEX program coincide with the strong answer sets of the dl-program.

The study of equivalence concepts for HEX programs in general is beyond the scope of this work. However, we can easily deal with the case where such programs contain external atoms having precisely the form of dl-atoms (monotonic or otherwise). For in this case the HEX semantics is in agreement with our first, alternative semantics for dl-programs, given in Definition 2. Without giving a detailed account of HEX programs, we indicate briefly why this is so.

Formally, external atoms in HEX programs have their own special notation and semantics. However, since dl-atoms can easily be simulated in HEX programs, for the purposes of our comparison let us keep the usual notation as for dl-programs. In that case, a HEX program is just a disjunctive logic program $\mathcal{P}$ containing rules of form (2) whose bodies can contain dl-atoms of form (1). The interpretation of such rules is similar to that of dl-programs except that a different form of program reduct is used. In [6] this is called *FLP-reduct* following the first use of this notion in [16].

Assume that we are given such a HEX program $\mathcal{P}$ along with some knowledge base $\mathcal{T}$ with respect to which the external atoms are evaluated (in what follows we shall leave the $\mathcal{T}$ component as implicit). Then the truth of an external atom of form (1) in a classical Herbrand model $\mathcal{M}$ is defined as for dl-programs in Section 4 above. Ground rules are also satisfied in $\mathcal{M}$ in the same way. Given $\mathcal{P}$ and a classical Herbrand model $\mathcal{M} = \langle U, T \rangle$, the reduct of $\mathcal{P}$ wrt. $\mathcal{M}$, denoted by $\mathcal{P}^{\mathcal{M}}$, is the set of all $r \in gr_U(\mathcal{P})$ such that $\mathcal{M} \models B(r)$. Then $\mathcal{M}$ is said to be an *answer set* of $\mathcal{P}$ iff it is a minimal model of $\mathcal{P}^{\mathcal{M}}$.

Proposition 9. *Let $\mathcal{P}$ be a HEX program as above with external atoms in the form of dl-atoms. A* **QHT** *Herbrand structure $\langle U, T, T \rangle$ is an equilibrium model of $\mathcal{P}$ under the semantics of Definition 2 if and only if $\langle U, T \rangle$ is an answer set of $\mathcal{P}$.*

All our observations and results about equivalences of dl-programs hold for any of the three semantics given. By Proposition 9 they carry over to HEX programs with external access to a TBox $\mathcal{T}$.

6 Conclusion

The logic **QHT** of quantified here-and-there provides a foundation for the answer set semantics of logic programs, and sharing the same **QHT**-models is a necessary and sufficient condition for two programs or theories to be strongly equivalent. In this paper we have shown how the concept of **QHT**-model can be extended to embrace also dl-programs interpreted under answer set semantics, removing the need for reducts and allowing a more logical style of semantics. Slight variations in the concept of **QHT**-model give rise to the weak and the strong answer set semantics as well as to a variation based on HEX programs.

As an application of the new semantics we considered some strong forms of equivalence between dl-programs as a first step towards the modular combination of ontologies and rules. Since a dl-program is a pair $(\mathcal{T}, \mathcal{P})$, strong forms of equivalence are obtained by considering theory extensions, which can be relativised to either the $\mathcal{T}$ component or the $\mathcal{P}$ component.

As in ordinary answer set programmming, the property that two theories have the same **QHT** models is again significant. This property is a necessary and sufficient condition for the $\mathcal{P}$-equivalence of dl-programs, if they are based on the same classical theory $\mathcal{T}$ or on possibly different but query inseparable TBoxes.

For the other main kind of equivalence, where the $\mathcal{T}$ component may vary, the situation is as follows. For dl-programs based on TBoxes we can use the idea of Σ-query inseparability to characterise forms of $\mathcal{T}$-equivalence for dl-programs based on the same $\mathcal{P}$ component or on **QHT**-equivalent programs. The concept of Σ-query inseparability has been studied for description logics such as DL-Lite and $\mathcal{EL}$ and model-theoretic characterisations are available and in some cases implemented [11, 26, 9, 25].

One direction of future work is to study modularity issues and equivalence concepts and their properties for dl-programs based on specific DLs such as $\mathcal{EL}$. Such properties may include algorithmic aspects and an analysis of computational complexity. Another direction of work is the study of more specific and relativised notions of equivalence between programs.

References

1. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: KR, pp. 141–151. AAAI Press, Menlo Park (2004)
2. Rosati, R.: Dl+log: Tight integration of description logics and disjunctive datalog. In: KR, pp. 68–78. AAAI Press, Menlo Park (2006)
3. Rosati, R.: On the decidability and complexity of integrating ontologies and rules. J. Web Sem. 3(1), 61–73 (2005)
4. Rosati, R.: Semantic and computational advantages of the safe integration of ontologies and rules. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 50–64. Springer, Heidelberg (2005)
5. Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., Nieuwenborgh, D.V.: Guarded hybrid knowledge bases. TPLP 8(3), 411–429 (2008)
6. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: IJCAI, pp. 90–96 (2005)

7. Pearce, D., Valverde, A.: Quantified equilibrium logic and foundations for answer set programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 546–560. Springer, Heidelberg (2008)
8. Pearce, D.: Equilibrium logic. AMAI 47(1-2), 3–41 (2006)
9. Lutz, C., Wolter, F.: Conservative extensions in the lightweight description logic $\mathcal{EL}$. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 84–99. Springer, Heidelberg (2007)
10. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: IJCAI, pp. 453–458 (2007)
11. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Can you tell the difference between dl-lite ontologies? In: KR, pp. 285–295. AAAI Press, Menlo Park (2008)
12. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM TOCL 2(4), 526–541 (2001)
13. Fink, M.: A general framework for equivalences in answer-set programming by countermodels in the logic of here-and-there. In: TPLP (2010) (forthcoming)
14. Fink, M.: Equivalences in answer-set programming by countermodels in the logic of here-and-there. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 99–113. Springer, Heidelberg (2008)
15. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 188–200. Springer, Heidelberg (2007)
16. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
17. Pearce, D., Valverde, A.: Quantified equilibrium logic an the first order logic of here-and-there. Technical Report MA-06-02, Univ. Rey Juan Carlos (2006)
18. van Dalen, D.: Logic and Structure, 4th edn. Springer, Heidelberg (2004)
19. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
20. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Open answer set programming with guarded programs. ACM TOCL 9(4) (2008)
21. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: IJCAI, pp. 372–379 (2007)
22. Pearce, D., Valverde, A.: A first order nonmonotonic extension of constructive logic. Studia Logica 80(2-3), 321–346 (2005)
23. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. AIJ 172(12-13), 1495–1539 (2008)
24. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
25. Lutz, C., Wolter, F.: Deciding inseparability and conservative extensions in the description logic el. J. Symb. Comput. 45(2), 194–228 (2010)
26. Kontchakov, R., Wolter, F., Zakharyaschev, M.: A logic-based framework for ontology comparison and module extraction in dl-lite (2010) (submitted)
27. Fink, M., Pearce, D.: Some equivalence concepts for hybrid theories. In: CAEPIA, Spanish Association for Artifical Intelligence, pp. 327–336 (2009)

An Incremental Answer Set Programming Based System for Finite Model Computation

Martin Gebser[1], Orkunt Sabuncu[1], and Torsten Schaub[1,2]

[1] Universität Potsdam
{gebser,orkunt,torsten}@cs.uni-potsdam.de
[2] Simon Fraser University, Canada, and Griffith University, Australia

Abstract. We address the problem of Finite Model Computation (FMC) of first-order theories and show that FMC can efficiently and transparently be solved by taking advantage of a recent extension of Answer Set Programming (ASP), called incremental Answer Set Programming (iASP). The idea is to use the incremental parameter in iASP programs to account for the domain size of a model. The FMC problem is then successively addressed for increasing domain sizes until an answer set, representing a finite model of the original first-order theory, is found. We implemented a system based on the iASP solver *iClingo* and demonstrate its competitiveness by showing that it slightly outperforms the winner of the FNT division of CADE's Automated Theorem Proving (ATP) competition.

1 Introduction

While Finite Model Computation (FMC;[1]) constitutes an established research area in the field of Automated Theorem Proving (ATP;[2]), Answer Set Programming (ASP;[3]) has become a widely used approach for declarative problem solving, featuring manifold applications in the field of Knowledge Representation and Reasoning. Up to now, however, both FMC and ASP have been studied in separation, presumably due to their distinct hosting research fields. We address this gap and show that FMC can efficiently and transparently be solved by taking advantage of a recent extension of ASP, called incremental Answer Set Programming (iASP;[4]).

Approaches to FMC for first-order theories [5,6] fall in two major categories, translational and constraint solving approaches. In translational approaches [7,8], the FMC problem is divided into multiple satisfiability problems in propositional logic. This division is based on the size of the finite domain. A Satisfiability (SAT;[9]) solver searches in turn for a model of the subproblem having a finite domain of fixed size, which is gradually increased until a model is found for the subproblem at hand. In the constraint solving approach [10,11], a system computes a model by incrementally casting FMC into a constraint satisfaction problem. While systems based on constraint solving are efficient for problems with many unit equalities, translation-based ones are applicable to a much wider range of problems [6].

In fact, translational approaches to FMC bear a strong resemblance to iASP. The latter was developed for dealing with dynamic problems like model checking and planning. To this end, iASP foresees an integer-valued parameter that is consecutively increased until a problem is found to be satisfiable. Likewise, in translation-based FMC,

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 169–181, 2010.
© Springer-Verlag Berlin Heidelberg 2010

the size of the interpretations' domain is increased until a model is found. This similarity in methodologies motivates us to encode and solve FMC by means of iASP.

The idea is to use the incremental parameter in iASP to account for the domain size. Separate subproblems considered in translational approaches are obtained by grounding an iASP encoding, where care is taken to avoid redundancies between subproblems. The parameter capturing the domain size is then successively incremented until an answer set is found. In the successful case, an answer set obtained for parameter value i provides a finite model of the input theory with domain size i.

We implemented a system based on the iASP solver *iClingo* [4] and compared its performance to various FMC systems. To this end, we used the problems from the FNT division of last year's CADE ATP competition. The results demonstrate the efficiency of our system. *iClingo* solved the same number of problems as *Paradox* [8] in approximately half of its run time on average. Note that *Paradox* won first places in the FNT division in 2007, 2008, and 2009.

The paper is organized as follows. The next section introduces basic concepts about the translational approach to FMC and about iASP. Section 3 describes our incremental encoding of FMC and how it is generated from a given set of clauses. Information about our system can be found in Section 4. We empirically evaluate our system in Section 5 and conclude in Section 6. An input first-order theory along with logic programs, as used by our FMC system based on iASP, are provided in appendixes.

2 Background

We assume the reader to be familiar with the terminology and basic definitions of first-order logic and ASP. In what follows, we thus focus on the introduction of concepts needed in the remainder of this paper.

In our method, we translate first-order theories into sets of flat clauses. A clause is *flat* if (i) all its predicates and functions have only variables as arguments, (ii) all occurrences of constants and functions are within equality predicates, and (iii) each equality predicate has at least one variable as an argument. Any first-order clause can be transformed into an equisatisfiable flat clause via *flattening* [7,8,6], done by repeatedly applying the rewrite rule $C[t] \rightsquigarrow (C[X] \vee (X \neq t))$, where t is a term offending flatness and X is a fresh variable. For instance, the clause $(f(X) = g(Y))$ can be turned into the flat clause $(Z = g(Y)) \vee (Z \neq f(X))$. In the translational approach to FMC, flattening is used to bring the input into a form that is easy to instantiate using domain elements.

As regards ASP, we rely on the language supported by grounders *lparse* [12] and *gringo* [13], providing normal and choice rules as well as cardinality and integrity constraints. As usual, rules with variables are regarded as representatives for all respective ground instances. Beyond that, our approach makes use of iASP [4] that allows for dealing with incrementally growing domains. In iASP, a parameterized domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over positive integers. In view of this, we sometimes denote P and Q by $P[k]$ and $Q[k]$. The base program B describes static knowledge, independent of parameter k. The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k. Our goal is then to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i] \tag{1}$$

has an answer set for some (minimum) integer $i \geq 1$. In what follows, we refer to rules in B, $P[k]$, and $Q[k]$ as being *static*, *cumulative*, and *volatile*, respectively.

3 Approach

In this section, we present our encoding of FMC in iASP. The first task, associating terms with domain elements, is dealt with in Section 3.1. Based on this, Section 3.2 describes the evaluation of (flat) clauses within iASP programs. In Section 3.3, we explain how a model of a first-order theory is then read off from an answer set. Section 3.4 presents an encoding optimization by means of symmetry breaking. Finally, we show the soundness and completeness of our approach in Section 3.5.

Throughout this section, we illustrate our approach on a running example. Assume that the following first-order theory is given as starting point:

$$\begin{array}{l} p(a) \\ (\forall X)\ \neg q(X, X) \\ (\forall X)\ (p(X) \rightarrow (\exists Y)\ q(X, Y)). \end{array} \tag{2}$$

The first preprocessing step, clausification of the theory, yields the following:

$$\begin{array}{l} p(a) \\ \neg q(X, X) \\ \neg p(X) \vee q(X, sko(X)). \end{array}$$

The second step, flattening, transforms these clauses into the following ones:

$$\begin{array}{l} p(X) \vee (X \neq a) \\ \neg q(X, X) \\ \neg p(X) \vee q(X, Y) \vee (Y \neq sko(X)). \end{array} \tag{3}$$

Such flat clauses form the basis for our iASP encoding. Before we present it, note that the theory in (3) has a model I over domain $\{1, 2\}$ given by:

$$\begin{array}{rl} a^I & = 1 \\ sko^I & = \{1 \mapsto 2, 2 \mapsto 2\} \\ p^I & = \{1\} \\ q^I & = \{(1, 2)\}. \end{array} \tag{4}$$

Importantly, I is also a model of the original theory in (2), even if sko^I is dropped.

3.1 Interpreting Terms

In order to determine a model, we need to associate the (non-variable) terms in the input with domain elements. To this end, every constant c is represented by a fact $cons(c)$., belonging to the *static* part of our iASP program. For instance, the constant a found in (3) gives rise to the following fact:

$$cons(a). \tag{5}$$

Our iASP encoding uses the predicate $assign(T, D)$ to represent that a term T is mapped to a domain element D. Here and in the following, we write k to refer to the incremental variable in an iASP program. Unless stated otherwise, all rules provided in the sequel are *cumulative* by default. For constants, the following (choice) rule then allows for mapping them to the kth domain element:

$$\{assign(T, k)\} \leftarrow cons(T). \tag{6}$$

Note that, by using k in $assign(T, k)$, it is guaranteed that instances of the rule are particular to each incremental step.

Unlike with constants, the argument tuples of (non-zero arity) functions grow when k increases. To deal with this, we first declare auxiliary facts to represent available domain elements:

$$dom(k). \qquad arg(k, k). \tag{7}$$

Predicates dom and arg are then used to qualify the arguments of an n-ary function f in the following rule:

$$\begin{aligned} func(f(X_1, \ldots, X_n)) \leftarrow\ & dom(X_1), \ldots, dom(X_n), \\ & 1\{arg(X_1, k), \ldots, arg(X_n, k)\}. \end{aligned} \tag{8}$$

The cardinality constraint $1\{arg(X_1, k), \ldots, arg(X_n, k)\}$ stipulates at least one of the arguments $X_1, \ldots, X_n$ of f to be k. As in (6), though using a different methodology, this makes sure that the (relevant) instances are particular to a value of k. However, note that rules of the above form need to be provided separately for each function in the input, given that the arities of functions matter. For the unary function sko in (3), applying the described scheme leads to the following rule:

$$func(sko(X)) \leftarrow dom(X), 1\{arg(X, k)\}. \tag{9}$$

To represent new mappings via a function when k increases, the previous methodology can easily be extended to requiring some argument or alternatively the function value to be k. The following (choice) rule encodes mappings via an n-ary function f:

$$\begin{aligned} \{assign(f(X_1, \ldots, X_n), Y)\} \leftarrow\ & dom(X_1), \ldots, dom(X_n), dom(Y), \\ & 1\{arg(X_1, k), \ldots, arg(X_n, k), arg(Y, k)\}. \end{aligned} \tag{10}$$

For instance, the rule encoding mappings via unary function sko is as follows:

$$\{assign(sko(X), Y)\} \leftarrow dom(X), dom(Y), 1\{arg(X, k), arg(Y, k)\}. \tag{11}$$

Observe that the cardinality constraint $1\{arg(X,k), arg(Y,k)\}$ necessitates at least one of argument X or value Y of function sko to be k, which in the same fashion as before makes the (relevant) instances of the rule particular to each incremental step.

To see how the previous rules are handled in iASP computations, we below show the instances of (7) and (11) generated in and accumulated over three incremental steps:

Step 1	Step 2	Step 3
$dom(1).\quad arg(1,1).$	$dom(2).\quad arg(2,2).$	$dom(3).\quad arg(3,3).$
$\{assign(sko(1),1)\}.$	$\{assign(sko(1),2)\}.$	$\{assign(sko(1),3)\}.$
	$\{assign(sko(2),1)\}.$	$\{assign(sko(2),3)\}.$
	$\{assign(sko(2),2)\}.$	$\{assign(sko(3),1)\}.$
		$\{assign(sko(3),2)\}.$
		$\{assign(sko(3),3)\}.$

Given that the body of (11) only relies on facts (over predicates dom and arg), its ground instances can be evaluated and then be reduced: if a ground body holds, the corresponding (choice) head is generated in a step; otherwise, the ground rule is trivially satisfied and needs not be considered any further. Hence, all rules shown above have an empty body after grounding. Notice, for example, that rule $\{assign(sko(1),1)\}.$ is generated in the first step, while it is not among the new ground rules in the second and third step.

Finally, a mapping of terms to domain elements must be unique and total. To this end, translation-based FMC approaches add uniqueness and totality axioms for each term to an instantiated theory. In iASP, such requirements can be encoded as follows:

$$\leftarrow assign(T,D), assign(T,k), D < k. \tag{12}$$

$$\leftarrow cons(T), \{assign(T,D) : dom(D)\}0. \tag{13}$$

$$\leftarrow func(T), \{assign(T,D) : dom(D)\}0. \tag{14}$$

While the integrity constraint in (12) forces the mapping of each term to be unique, the ones in (13) and (14) stipulate each term to be mapped to some domain element. However, since the domain grows over incremental steps and new facts are added for predicate dom, ground instances of (13) and (14) are only valid in the step where they are generated. Hence, the integrity constraints in (13) and (14) belong to the *volatile* part of our iASP program.

3.2 Interpreting Clauses

To evaluate an input theory, we also need to interpret its predicates. To this end, we include a rule of the following form for every n-ary predicate p in our iASP program:

$$\begin{aligned}\{p(X_1,\ldots,X_n)\} \leftarrow\ & dom(X_1),\ldots,dom(X_n),\\ & 1\{arg(X_1,k),\ldots,arg(X_n,k)\}.\end{aligned} \tag{15}$$

As discussed above, requiring $1\{arg(X_1,k),\ldots,arg(X_n,k)\}$ to hold guarantees that (relevant) instances are particular to each incremental step. The only exception to this

is $n = 0$ (a predicate p of arity zero), in which case the rule $\{p\}.$ belongs to the *static* part of our program. Also note that, unlike constants and functions, we do not reify predicates, as assigning a truth value can be expressed more naturally without it. For example, the following rules allow for interpreting the predicates p and q in (3):

$$\begin{aligned} \{p(X)\} &\leftarrow dom(X), 1\{arg(X,k)\}. \\ \{q(X,Y)\} &\leftarrow dom(X), dom(Y), 1\{arg(X,k), arg(Y,k)\}. \end{aligned} \tag{16}$$

Following [14], the basic idea of encoding a (flat) clause is to represent it by an integrity constraint containing the complements of the literals in the clause. However, clauses may contain equality literals of the form $(X = Y)$ or $(X \neq Y)$, where at least one of the terms X and Y is a variable, and so we also need to consider complements of such literals. W.l.o.g., we below assume that the left-hand side of every equality literal is a variable, while the right-hand side is either a variable or a non-variable term. In view of this convention, we define the encoding $\overline{L}$ of the complement of a (classical or equality) literal L as follows:

$$\overline{L} = \begin{cases} not\ p(X_1,\ldots,X_n) & \text{if } L = p(X_1,\ldots,X_n) \\ p(X_1,\ldots,X_n) & \text{if } L = \neg p(X_1,\ldots,X_n) \\ not\ assign(t,X) & \text{if } L = (X = t) \text{ for some non-variable term } t \\ assign(t,X) & \text{if } L = (X \neq t) \text{ for some non-variable term } t \\ X \neq Y & \text{if } L = (X = Y) \text{ for some variable } Y \\ X = Y & \text{if } L = (X \neq Y) \text{ for some variable } Y. \end{cases}$$

Observe that the first two cases refer to the interpretation of a predicate p, the third and the fourth to the mapping of non-variable terms to domain elements, and the last two to built-in comparison operators of grounders like *lparse* and *gringo*.

With the complements of literals at hand, we can now encode a flat clause containing literals $L_1,\ldots,L_m$ and variables $X_1,\ldots,X_n$ by an integrity constraint as follows:

$$\leftarrow \overline{L}_1,\ldots,\overline{L}_m, dom(X_1),\ldots,dom(X_n), 1\{arg(X_1,k),\ldots,arg(X_n,k)\}. \tag{17}$$

Note that we use the same technique as before to separate the (relevant) instances obtained at each incremental step. For our running example, the clauses in (3) give rise to the following integrity constraints:

$$\begin{aligned} &\leftarrow not\ p(X), assign(a,X), dom(X), 1\{arg(X,k)\}. \\ &\leftarrow q(X,X), dom(X), 1\{arg(X,k)\}. \\ &\leftarrow p(X), not\ q(X,Y), assign(sko(X),Y), \\ &\quad dom(X), dom(Y), 1\{arg(X,k), arg(Y,k)\}. \end{aligned} \tag{18}$$

While the first two integrity constraints each contribute a single instance at an incremental step, $(2 * k) - 1$ instances are obtained for the third one.

Although they are unlikely to occur in first-order theories, *propositional* clauses without variables and equality literals require a slightly different treatment. For a propositional clause containing (classical) literals $L_1,\ldots,L_m$, instead of (17), we include the following simpler integrity constraint in the *static* part of our iASP program:

$$\leftarrow \overline{L}_1,\ldots,\overline{L}_m. \tag{19}$$

3.3 Extracting Models

The rules that represent the mapping of terms to domain elements (described in Section 3.1) along with those representing satisfiability of flat clauses (described in Section 3.2) constitute our iASP program for FMC. To compute an answer set, the incremental variable k is increased by one at each step. This corresponds to the addition of a new domain element. If an answer set is found in a step i, it means that the input theory has a model over a domain of size i. In fact, from an answer set A of our iASP program, a model I of the input theory over domain $\{d \mid dom(d) \in A\}$ is extracted as follows:

$$\begin{aligned} c^I &= d \text{ where } cons(c), assign(c,d) \in A, \\ f^I &= \{(d_1,\ldots,d_n) \mapsto d \mid assign(f(d_1,\ldots,d_n),d) \in A\}, \\ p^I &= \{(d_1,\ldots,d_n) \mid p(d_1,\ldots,d_n) \in A\}. \end{aligned}$$

For the iASP program encoding the theory in (3), composed of the rules in (5–7, 9, 11–14, 16, 18), the following answer set is obtained in the second incremental step:

$$\left\{ \begin{array}{l} dom(1), dom(2), arg(1,1), arg(2,2), \\ cons(a), assign(a,1), \\ func(sko(1)), assign(sko(1),2), \\ func(sko(2)), assign(sko(2),2), \\ p(1), q(1,2) \end{array} \right\}$$

The corresponding model over domain $\{1,2\}$ is the one shown in (4).

3.4 Breaking Symmetries

In view of the fact that interpretations obtained by permuting domain elements are isomorphic, an input theory can have many symmetric models. For example, an alternative model to the one in (4) can easily be obtained by swapping domain elements 1 and 2. Such symmetries tend to degrade the performance of FMC systems. Hence, systems based on the constraint solving approach, such as *Sem* and *Falcon*, apply variants of a dynamic symmetry breaking technique called least number heuristic [11]. Translation-based systems, such as *Paradox* and *FM-Darwin*, statically break symmetries by narrowing how terms can be mapped to domain elements.

Our approach to symmetry breaking is also a static one that aims at reducing the possibilities of mapping constants to domain elements. To this end, we use the technique described in [8, 15], fixing an order of the constants in the input by uniquely assigning a rank in $[1,n]$, where n is the total number of constants, to each of them. Given such a ranking in terms of facts over predicate $order$, we can replace the rule in (6) with:

$$\{assign(T,k)\} \leftarrow cons(T), order(T,O), k \leq O.$$

For instance, if the set of constants is $\{c_1,c_2,c_3\}$ and the order is given by facts $order(c_i,i)$. for $i \in \{1,2,3\}$, the following instances of the above rule are generated in and accumulated over three incremental steps:

Step 1	Step 2	Step 3
$\{assign(c_1,1)\}.$		
$\{assign(c_2,1)\}.$	$\{assign(c_2,2)\}.$	
$\{assign(c_3,1)\}.$	$\{assign(c_3,2)\}.$	$\{assign(c_3,3)\}.$

That is, while all three constants can be mapped to the first domain element, c_1 cannot be mapped to the second one, and only c_3 can be mapped to the third one.

Finally, we note that our iASP encoding of the theory in (3) yields 10 answer sets in the second incremental step. If we apply the described symmetry breaking, it disallows mapping the single constant a to the second domain element, which prunes 5 of the 10 models. Although our simple technique can in general not break all symmetries related to the mapping of terms because it does not incorporate functions, the experiments in Section 5 demonstrate that it may nonetheless lead to significant performance gains. Unlike with constants, given a priori, additionally incorporating functions into our approach to symmetry breaking would require the extension of predicate *order* to newly composed functional terms in each incremental step. For the special case of unary functions, such an extension [8] is implemented in *Paradox*; with *FM-Darwin*, it has not turned out to be more effective than symmetry breaking for only constants [15].

3.5 Soundness and Completeness

Before stating our theorem, we first define the parameterized domain description formed for a set $\mathcal{T}$ of flat clauses. The signature $\langle \mathcal{F}_0, \mathcal{F}, \mathcal{P}_0, \mathcal{P} \rangle$ of $\mathcal{T}$ is built from a set $\mathcal{F}_0$ of *constants*, a set $\mathcal{F}$ of (non-zero arity) *functions*, a set $\mathcal{P}_0$ of *zero arity predicates*, and a set $\mathcal{P}$ of *non-zero arity predicates*. For $\mathcal{T}$, we then form the parameterized domain description (B, P, Q) in the following way:

$$
\begin{aligned}
B &= \{\ cons(c).\mid c \in \mathcal{F}_0\ \} \cup \{\ \{p\}.\mid p \in \mathcal{P}_0\ \} \cup \Pi^{\mathcal{T}_0}\ ,\\
P &= \{\ dom(k).\quad arg(k,k).\quad \{assign(T,k)\} \leftarrow cons(T).\\
&\qquad \leftarrow assign(T,D), assign(T,k), D<k.\ \} \cup \Pi^{\mathcal{F}} \cup \Pi^{\mathcal{P}} \cup \Pi^{\mathcal{T}}\ ,\ \text{and}\\
Q &= \{\leftarrow cons(T), \{assign(T,D) : dom(D)\}0.\\
&\qquad \leftarrow func(T), \{assign(T,D) : dom(D)\}0.\ \}\ ,
\end{aligned}
$$

where $\Pi^{\mathcal{F}}$ contains rules of form (8) and form (10) for each function $f \in \mathcal{F}$, $\Pi^{\mathcal{P}}$ contains a rule of form (15) for each predicate $p \in \mathcal{P}$, $\Pi^{\mathcal{T}}$ contains a rule of form (17) for each non-propositional clause in $\mathcal{T}$, and $\Pi^{\mathcal{T}_0}$ contains a rule of form (19) for each propositional clause in $\mathcal{T}$. With these concepts at hand, we are ready to formulate the soundness and completeness of our approach.

Theorem 1. *Let $\mathcal{T}$ be a set of flat clauses and (B, P, Q) the parameterized domain description for $\mathcal{T}$. Then, the logic program $R[k/i]$, as defined in (7), has an answer set for some positive integer i iff $\mathcal{T}$ has a finite model over a domain of size i.*

Note that the theorem still applies when including symmetry breaking, as described in the previous section, in view of the fact that it may eliminate some isomorphic models, but not all of them.

4 System

We use *FM-Darwin* to read an input in TPTP format, a format for first-order theories widely used within the community of ATP, to clausify it if needed, and to flatten the clauses at hand. Additionally, *FM-Darwin* applies some input optimizations before flattening, such as renaming deep ground subterms to avoid the generation of flat clauses with many variables [15]. For obtaining flat clauses from an input theory specified in a file `tptp_input.p`, *FM-Darwin* is invoked as follows:

```
darwin -fd true -pfdp Exit tptp_input.p
```

Having an input in terms of flat clauses, we can apply the transformations described in Section 3.1 and 3.2 to generate an iASP program. To this end, we implemented a compiler called *fmc2iasp*[1], written in Python. It outputs the rules that are specific to an input theory, while the theory-independent rules in (6), (7), and (12–14) are provided in a separate file. This separation allows us to test encoding variants without changing *fmc2iasp*, for instance, the symmetry breaking described in Section 3.4. Finally, we use *iClingo* to incrementally ground the obtained iASP program and to search for answer sets representing finite models of the input theory. Provided that `fmc.lp` is the file containing theory-independent rules, the following command-line call is used for FMC:

```
darwin -fd true -pfdp Exit tptp_input.p | fmc2iasp.py |
cat fmc.lp - | iclingo
```

5 Experiments

We consider the following systems: *iClingo* (2.0.5), *Clingo* (2.0.5), *Paradox* (3.0), *FM-Darwin* (1.4.5), and *Mace4* (2009-11A). While *Paradox* and *FM-Darwin* are based on the translational approach to FMC, *Mace4* applies the constraint solving approach. For *iClingo* and *Clingo*, we used command line switch `--heuristic=VSIDS`, as it improved search performance.[2] Our experiments have been performed on a 3.4GHz Intel Xeon machine running Linux, imposing 300 seconds as time and 2GB as memory limit.

FMC instances stem from the FNT (First-order form Non-Theorems) division of the 2009 CADE ATP competition. The instances in this division are satisfiable and suitable for evaluating FMC systems, among which *Paradox* won the first place. The considered problem domains are: common-sense reasoning (CSR), geography (GEG), geometry (GEO), graph theory (GRA), groups (GRP), homological algebra (HAL), knowledge representation (KRS), lattices (LAT), logic calculi (LCL), management (MGT), miscellaneous (MSC), natural language processing (NLP), number theory (NUM), processes (PRO), software verification (SWV), syntactic (SYN).[3]

Table 1 shows benchmark results for each of the problem domains. Column # displays how many instances of a problem domain belong to the test suite. For each system and problem domain, average run time in seconds is taken over the solved instances; their

[1] `http://potassco.sourceforge.net/`

[2] Note that *Minisat*, used internally by *Paradox*, also applies VSIDS as decision heuristic [16].

[3] `http://www.cs.miami.edu/~tptp/`

Table 1. Benchmark results for problems in the FNT division of the 2009 CADE competition

Benchmark	#	*iClingo (1)*	*iClingo (2)*	*Clingo*	*Paradox*	*FM-Darwin*	*Mace4*
CSR	1	2.28 (1)	2.19 (1)	4.20 (1)	—	20.96 (1)	—
GEG	1	—	—	—	229.12 (1)	—	—
GEO	12	0.07 (12)	0.06 (12)	0.09 (12)	0.08 (12)	0.09 (12)	0.02 (12)
GRA	2	3.48 (1)	—	12.33 (1)	0.50 (1)	—	—
GRP	1	5.58 (1)	215.62 (1)	78.90 (1)	0.65 (1)	—	0.26 (1)
HAL	2	2.35 (2)	2.40 (2)	2.68 (2)	0.68 (2)	11.43 (1)	—
KRS	6	0.13 (6)	0.13 (6)	0.24 (6)	0.20 (6)	30.76 (6)	0.02 (4)
LAT	5	0.09 (5)	0.09 (5)	0.12 (5)	0.10 (5)	0.07 (5)	0.03 (5)
LCL	17	8.71 (17)	9.36 (17)	11.44 (17)	3.72 (17)	1.56 (17)	5.07 (8)
MGT	4	0.06 (4)	0.06 (4)	0.08 (4)	0.07 (4)	0.12 (4)	0.98 (4)
MSC	3	9.31 (2)	0.20 (1)	16.51 (2)	121.03 (2)	0.19 (1)	—
NLP	9	1.60 (9)	1.98 (9)	3.06 (9)	0.25 (9)	0.28 (8)	22.19 (1)
NUM	1	0.19 (1)	0.19 (1)	0.26 (1)	0.26 (1)	0.13 (1)	202.45 (1)
PRO	9	1.09 (9)	8.99 (9)	1.91 (9)	0.37 (9)	0.78 (9)	31.56 (7)
SWV	8	0.13 (4)	0.12 (4)	0.17 (4)	0.16 (4)	45.13 (5)	0.03 (2)
SYN	18	0.56 (18)	0.57 (18)	0.68 (18)	0.40 (18)	3.88 (12)	0.66 (5)
Total	99	2.39 (92)	5.49 (90)	4.24 (92)	6.02 (92)	6.43 (82)	9.88 (50)

number is given in parentheses. A dash in an entry means that a system could not solve any instance of the corresponding problem domain within the run time and memory limits. For each system, the last row shows its average run time over all solved instances and provides their number in parentheses. The evaluation criteria in CADE competitions are first number of solved instances and then average run time as tie breaker.

In Table 1, we see that *Mace4* and *FM-Darwin* solved 50 and 82 instances, respectively, out of the 99 instances in total. *Paradox*, the winner of the FNT division in the 2009 CADE competition, solved 92 instances in 6.02 seconds on average. While the version of our system not using symmetry breaking (described in Section 3.4), denoted by *iClingo (2)*, solved two instances less, the one with symmetry breaking, denoted by *iClingo (1)*, also solved 92 instances. As it spent only 2.39 seconds on average, according to the CADE criteria, our system slightly outperformed *Paradox*. For assessing the advantages due to incremental grounding and solving, we also ran *Clingo*, performing iterative deepening search by successively grounding and solving our iASP encoding for fixed domains of increasing size. The average run time achieved with *Clingo*, 4.24 seconds, is substantially greater than the one of *iClingo (1)*, and the gap becomes more apparent the more domain elements are needed.

However, a general problem with the translational approach is that flattening may increase the number of variables in a clause, which can deteriorate grounding performance. We observed this clearly for the instance in the GEG domain, where the flat clauses contain about seven variables. While *iClingo* could not ground the resulting iASP program within the given limits, *Paradox* still solved it (in 229.12 seconds). The fact that the underlying first-order theory has many sorts, so that sort inference [8] of *Paradox* helps, shows that there is still potential to improve the translational approach via iASP. On the other hand, for the instances in groups CSR and MSC, we speculate

that clausification and further preprocessing steps of *Paradox* may be the cause for its deteriorated performance.

6 Discussion

We presented an efficient yet transparent approach to computing finite models of first-order theories by means of ASP. Our approach takes advantage of an incremental extension of ASP that allows us to consecutively search for models with given domain size by incrementing the corresponding parameter in the iASP encoding. The declarative nature of our approach makes it easily modifiable and leaves room for further improvements. Moreover, our approach is rather competitive and has even a slight edge on the hitherto leading system for FMC. Finally, our approach complements the work in [17], where FMC systems were used for computing the answer sets of tight[4] logic programs in order to circumvent grounding.

In [18], a special class of first-order formulas, called Effectively Propositional (EPR) formulas, was addressed via ASP. EPR formulas must not contain function symbols in their clause forms. Although our approach takes more general input than this, it can currently not decide EPR formulas. To this end, we had either to extract a bound on the incremental parameter to make the system halt or to provide an alternative dedicated encoding of EPR formulas. Such extensions are interesting topics for future research.

Acknowledgments. This work was supported by the German Science Foundation (DFG) under grant SCHA 550/8-1.

References

1. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. Kluwer Academic, Dordrecht (2004)
2. Bibel, W.: Automated Theorem Proving. Vieweg (1987)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University, Cambridge (2003)
4. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 190–205. Springer, Heidelberg (2008)
5. Zhang, J., Huang, Z.: Reducing symmetries to generate easier SAT instances. Electronic Notes in Theoretical Computer Science 125(3), 149–164 (2005)
6. Tammet, T.: Finite model building: Improvements and comparisons. In: Baumgartner, P., Fermüller, C. (eds.) Proceedings of the Workshop on Model Computation — Principles, Algorithms, Applications, MODEL 2003 (2003)
7. McCune, W.: A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical Report ANL/MCS-TM-194, Argonne National Laboratory (1994)
8. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: Baumgartner, P., Fermüller, C. (eds.) Proceedings of the Workshop on Model Computation — Principles, Algorithms, Applications, MODEL 2003 (2003)

[4] Tight programs are free of recursion through positive literals (cf. [3]).

9. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS (2009)
10. Zhang, J., Zhang, H.: SEM: A system for enumerating models. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995), pp. 298–303. Morgan Kaufmann, San Francisco (1995)
11. Zhang, J.: Constructing finite algebras with FALCON. Journal of Automated Reasoning 17(1), 1–22 (1996)
12. Syrjänen, T.: Lparse 1.0 user's manual, http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz
13. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to gringo, clasp, clingo, and iclingo, http://potassco.sourceforge.net
14. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138(1-2), 181–234 (2002)
15. Baumgartner, P., Fuchs, A., de Nivelle, H., Tinelli, C.: Computing finite models by reduction to function-free clause logic. Journal of Applied Logic 7(1), 58–74 (2009)
16. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
17. Sabuncu, O., Alpaslan, F.: Computing answer sets using model generation theorem provers. In: Costantini, S., Watson, R. (eds.) Proceedings of the 4th International Workshop on Answer Set Programming (ASP 2007), pp. 225–240 (2007)
18. Lierler, Y., Lifschitz, V.: Logic programs vs. first-order formulas in textual inference, http://z.cs.utexas.edu/users/ai-lab/publications_recent.php

A Input Theory

The input theory (2), written in TPTP format, is as follows:

```
fof(1, axiom, p(a)).
fof(2, axiom, ! [X]: (~q(X,X)) ).
fof(3, axiom, ! [X]: (p(X) => (? [Y]: q(X,Y))) ).
```

B Theory-Independent iASP Program

The theory-independent program part with symmetry breaking (cf. Section 3.4), in the input language of *iClingo*, is as follows:

```
#cumulative k.

dom(k).
arg(k,k).

{ assign(T,k) } :- cons(T), order(T,O), k<=O.

:- assign(T,D), assign(T,k), D<k.

#volatile k.

:- cons(T), { assign(T,D):dom(D) } 0.
:- func(T), { assign(T,D):dom(D) } 0.
```

C Theory-Dependent iASP Program

The rules generated by *fmc2iasp* for the flat clauses in (3) are as follows:

```
#cumulative k.

% functions
func(sko(X0)) :- dom(X0), 1 { arg(X0,k) }.
{ assign(sko(X0),Y) } :- dom(X0;Y), 1 { arg(X0;Y,k) }.

% predicates
{ p(X0) } :- dom(X0), 1 { arg(X0,k) }.
{ q(X0,X1) } :- dom(X0;X1), 1 { arg(X0;X1,k) }.

% flat clauses
:- not p(X0), assign(a,X0), dom(X0), 1 { arg(X0,k) }.
:- q(X0,X0), dom(X0), 1 { arg(X0,k) }.
:- p(X0), not q(X0,X1), assign(sko(X0),X1),
   dom(X0;X1), 1 { arg(X0;X1,k) }.

#base.

cons(a).
order(a,1).

#hide.
#show assign/2.
#show q/2.
#show p/1.
```

In order to compute a finite model of (2), we can use this program concatenated with the rules from Appendix B, as it is described in Section 4.

Parametrized Logic Programming

Ricardo Gonçalves* and José Júlio Alferes

CENTRIA, Dep. Informática, FCT/Universidade Nova de Lisboa, Portugal

Abstract. Traditionally, a logic program is built up to reason about atomic first-order formulas. The key idea of parametrized logic programming is that, instead of atomic first-order formulas, a parametrized logic program reasons about formulas of a given parameter logic. Of course, the main challenge is to define the semantics of such general programs. In this work we introduce the novel definitions along with some motivating examples. This approach allows us to prove general results that can be instantiated for every particular choice of the parameter logic. Important general results we can prove include the existence of semantics and the alternating fix-point theorem of well-founded semantics. To reenforce the soundness of our general framework we show that some known approaches in the literature of logic programming, such as paraconsistent answer-sets and the MKNF semantics for hybrid knowledge bases, are obtained as particular choices of the parameter logic.

1 Introduction

Usually, we write a logic program to reason about atomic formulas. The truth value of each of these atoms does not influence the truth values of the others. Thus, in this sense, atomic first-order formulas are independent of each other. One way to increase the expressivity of a logic program is to allow dependence between the atoms. Pursuing this path, some approaches in the literature allow complex formulas to appear in the body and head of rules [4,15].

Our approach, parametrized logic programming, also aims this increased expressivity, but has its roots in the area of combination of logics [2]. Combination mechanisms are operations that take logics as arguments and produce new logics as a result. Combined logics are not only important from a theoretical point of view but, what is more, have also a deep practical signicance, namely in areas like knowledge representation in articial intelligence. In fact, the use of formal logic as a tool for knowledge representation frequently requires the integration of several logic systems into a homogeneous environment. Fibring of logics [5] is one of the most general and flexible mechanism for combining logics and it has parametrization of logics [1] as an important special case. Roughly speaking, parametrization of logics consists of replacing the atomic part of a given logic $\mathcal{L}$ by a logic $\mathcal{L}'$, which is called the parameter logic.

* The first author was supported by FCT under the postdoctoral grant SFRH/BPD/47245/2008.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 182–194, 2010.
© Springer-Verlag Berlin Heidelberg 2010

In the case of logic programming, our idea is to fix a parameter logic $\mathcal{L}$ and build up logic programs by replacing atomic sentences with formulas of $\mathcal{L}$. These more expressive atoms (the formulas of $\mathcal{L}$) are no longer independent and their interdependence is governed by $\mathcal{L}$ through its consequence relation.

It is important to stress that our approach differs from existing ones, such as [15,3], that aim to find the "logic" of logic programming, i.e., find a suitable logic that generalizes the well-known role of classical logic in the semantics of definite logic programs. Contrarily, the keystone idea of parametrized logic programming is the decoupling between the metalevel language and the parametrized logical language. In the metalevel language we have the usual constructors of logic programs rules: $\leftarrow, ",", ", not\ and" or"$. In the parametrized logical language we have the language of the parameter logic $\mathcal{L}$.

The major challenge when envisaging such a general language is to define the semantics. Here, our key idea is to generalize the concept of interpretation with the notion of logical theory. This is in fact a non-trivial novel approach and it overcomes the difficulty of defining semantics of more complex logic programs. For example, it is well-known that classical interpretations are not suitable for defining the semantics of extended logic programs, thus leading to the artificial trick of considering classically negated atoms as new atoms.

As one application, we show that our approach can be very useful in the topic of combining rules and ontologies, which is an important topic in for the Semantic Web. In fact, we prove that one of the main frameworks for combining rules and ontologies for the semantic web, the MKNF semantics for hybrid knowledge bases, can be captured as a particular case of our approach, and, moreover, our language is richer than that of MKNF.

We start by introducing the novel definitions along with some motivating examples. We then define a general version of stable model semantics and of the well-founded semantics and prove that the non-parametrized is a special case obtained by a natural choice of the parameter logic. In the last section we prove that the semantics of MKNF knowledge bases is nothing but that of logic programs parametrized by the appropriate description logic. We end by drawing some conclusions and presenting some paths for future work.

2 Parametrized Normal Logic Programs

In this section we introduce the syntax and semantics of normal parametrized logic programs. To focus on the novel key ideas we will not consider here epistemic disjunction *or*, along with its intrinsic difficulties[1].

2.1 Language

The syntax of a normal parametrized logic program has the same structure of that of a normal logic program. The only difference is that the atomic symbols

[1] In fact, a results more general than this one, namely generalising for parametrized Here-There theories [13], is subject of current work.

of a normal parametrized logic program are replaced by formulas of a parameter logic. Let us start by introducing the necessary concepts related with the notion of (monotonic) logic.

Definition 1. *A (monotonic) logic is a pair* $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ *where* L *is a set of formulas and* $\vdash_{\mathcal{L}}$ *is a Tarskian consequence relation [16] over* L*, i.e. satisfying the following conditions, for every* $T \cup \Phi \cup \{\varphi\} \subseteq L$*,* **Reflexivity:** *if* $\varphi \in T$ *then* $T \vdash_{\mathcal{L}} \varphi$*;* **Cut:** *if* $T \vdash_{\mathcal{L}} \varphi$ *for all* $\varphi \in \Phi$*, and* $\Phi \vdash_{\mathcal{L}} \psi$ *then* $T \vdash_{\mathcal{L}} \psi$*;* **Weakening:** *if* $T \vdash_{\mathcal{L}} \varphi$ *and* $T \subseteq \Phi$ *then* $\Phi \vdash_{\mathcal{L}} \varphi$*.*

When clear from the context we write $\vdash$ instead of $\vdash_{\mathcal{L}}$. Let $Th(\mathcal{L})$ be the set of *theories of* $\mathcal{L}$, i.e. the set of subsets of L closed under the relation $\vdash_{\mathcal{L}}$. It is well-known that, for every (monotonic) logic $\mathcal{L}$, the tuple $\langle Th(\mathcal{L}), \subseteq \rangle$ is a complete lattice with smallest element the set $Theo = \emptyset^{\vdash}$ of theorems of $\mathcal{L}$ and the greatest element the set L of all formulas of $\mathcal{L}$. Given a subset A of L we denote by $A^{\vdash}$ the smallest theory that contains A. $A^{\vdash}$ is also called the theory generated by A.

In what follows we consider fixed a (monotonic) logic $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ and call it the *parameter logic*. The formulas of $\mathcal{L}$ are dubbed *(parametrized) atoms* and a *(parametrized) literal* is either a parametrized atom φ or its negation *not* φ, where as usual *not* denotes negation as failure. We dub *default literal* those of the form *not* φ.

Definition 2. *A* normal $\mathcal{L}$-parametrized logic program *is a set of rules*

$$\varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \delta_1, \ldots, not\ \delta_m$$

where $\varphi, \psi_1, \ldots, \psi_n, \delta_1, \ldots, \delta_m \in L$*.*

A definite $\mathcal{L}$-parametrized logic program *is a set of rules without negations as failure, i.e. of the form* $\varphi \leftarrow \psi_1, \ldots, \psi_n$ *where* $\varphi, \psi_1, \ldots, \psi_n \in L$*.*

2.2 Semantics

Given this general language of parametrized logic programs, we define its stable model semantics and its well-founded semantics, as generalisations of the stable model semantics [8] and well-founded semantics [7] of normal logic programs.

Interpretations and Models. In the traditional approach a (2-valued) interpretation is just a set of atoms. In our approach, since we substitute atoms by formulas of a parameter logic, the first idea is to take sets of formulas of the parameter logic as (2-valued) interpretations. The problem is that, contrary to the case of atoms, the parametrized atoms are not independent of each other. This interdependence is governed by the consequence relation of the parameter logic. For example, if we take classical propositional logic (CPL) as the parameter logic, we have that if the parametrized atom $p \wedge q$ is true then so are the parametrized atoms p and q. To account for this interdependence, we use theories (sets of formulas closed under the consequence of the logic) as the generalisation of interpretations, thus capturing the above mentioned interdependence.

Definition 3. *A (parametrized) 2-valued interpretation is a theory of $\mathcal{L}$.*

As usual, a 2-valued interpretation T can be seen as a tuple $\langle T, F\rangle$ such that T is a theory of $\mathcal{L}$, and F is the complement, wrt L, of T. Note that, defined as such, F is not a theory, viz. it is not closed under the consequence of the logic. E.g. F does not, and should not, include tautologies in the parameter logic. This must be taken into account when defining parametrized 3-valued interpretations.

In 3-valued interpretations, as defined below, formulas are either true, undefined or false. True formulas, as we just seen, must be closed under the consequence of the logic; non-false formulas (i.e. true or undefined formulas) must also be closed; false formulas are just the complement of non-false formulas, and as such are not closed.

Definition 4. *A (parametrized) 3-valued interpretation is determined by any two theories T and T_U such $T \subseteq T_U$. For similarity with the usual definition of 3-valued interpretations, we represent one such interpretation as a tuple $\langle T, F\rangle$ where $F = L - T_U$.*

Any interpretation $I = \langle T, F\rangle$ can be equivalently defined as a function $I : L \to \{0, \frac{1}{2}, 1\}$ in the usual way. We dub *empty interpretation* the 3-valued interpretation $\langle \emptyset^{\vdash}, \emptyset\rangle$. Given an interpretation I we can extend it to literals by setting $I(not\ \varphi) = 1 - I(\varphi)$ for every $\varphi \in L$.

Definition 5. *An interpretation I satisfies a rule*

$$\varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \delta_1, \ldots, not\ \delta_m$$

if $Min\{I(\psi_1), \ldots, I(\psi_n), I(not\ \delta_1), \ldots, I(not\ \delta_m)\} \leq I(\varphi)$.

An interpretation is a model of logic program P if it satisfies every rule of P. We denote by $Mod_2^{\mathcal{L}}(P)$ the set of 2-valued models of P and by $Mod_3^{\mathcal{L}}(P)$ the set of 3-valued models of P.

The usual orderings defined over 2- and 3-valued interpretations can easily be generalised. Moreover, given one such ordering, minimal and least interpretations may be defined in the usual way.

Definition 6 (Classical ordering). *If I and J are two interpretations then we say that $I \leq J$ if $I(\varphi) \leq J(\varphi)$ for every $\varphi \in L$.*

Definition 7 (Fitting ordering). *If $I_1 = \langle T_1, F_1\rangle$ and $I_2 = \langle T_2, F_2\rangle$ are two 3-valued interpretations then we say that $I_1 \leq_F I_2$ if $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$.*

Stable Model Semantics. As in the case of non-parametrized, we start by assigning semantics to definite parametrized programs. The stable model of a definite program is its least 2-valued model. In order to generalise this definition to the parametrized case we need to prove that the least parametrized 2-valued model exists for every definite $\mathcal{L}$-parametrized logic program.

Theorem 1. *Every definite $\mathcal{L}$-parametrized logic program has a least 2-valued model.*

Proof. Consider the set $S_P^{\mathcal{L}} = \bigcap_{M \in Mod_2^{\mathcal{L}}(P)} M$. Note that the set $S_P^{\mathcal{L}}$ always exists since $\langle Th_{\mathcal{L}}, \subseteq \rangle$ is a complete lattice for every (monotonic) logic $\mathcal{L}$. It is clear that $S_P^{\mathcal{L}}$ is included in every model of P and it is trivial to prove that $S_P^{\mathcal{L}}$ is still a model of P.

It is important to note that this theorem holds for every choice of the parameter logic $\mathcal{L}$.

To define the stable model semantics of a normal $\mathcal{L}$-parametrized logic programs we use a Gelfond-Lifschitz like operator.

Definition 8. *Let P be a normal $\mathcal{L}$-parametrized logic program and I a 2-valued interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:*

- *remove from P all rules which contain a literal not φ such that $I \vdash_{\mathcal{L}} \varphi$;*
- *remove from the remaining rules all default literals.*

Since $\frac{P}{I}$ is a definite $\mathcal{L}$-parametrized program, it has an unique least model J. We define $\Gamma(I) = J$.

Definition 9. *A 2-valued interpretation I of a $\mathcal{L}$-parametrized logic program P is a stable model of P iff $\Gamma(I) = I$. A formula φ is true under the stable model semantics iff it belongs to all stable models of P.*

Well-Founded Semantics. Several equivalent definitions of well-founded semantics exist in the literature. In this work we follow the iterated fixed-point approach of [6].

First of all it can be readily proved that, as in the non-parametrized case, the operator Γ is antitonic, i.e, for any theories $I, J \in \mathcal{I}$ we have that if $I \subseteq J$ then $\Gamma(J) \subseteq \Gamma(I)$. Therefore, applying Γ twice, denoted by Γ^2, yields a monotone operator on the lattice of theories.

Definition 10. *A 3-valued interpretation $\langle T, F \rangle$ is a partial stable model of a normal $\mathcal{L}$-parametrized logic program P if $T = \Gamma^2(T)$ and $F = L - \Gamma(T)$.*

For defining the well-founded semantics, we first need to prove that the F-least partial stable model of a normal $\mathcal{L}$-parametrized logic program always exits.

Theorem 2. *Every normal $\mathcal{L}$-parametrized logic program has a unique F-least partial stable model.*

Proof. Let P be a normal $\mathcal{L}$-parametrized logic program. Since Γ^2 is a monotone operator we can conclude by the Knaster-Tarski thereom that Γ^2 has a least fixed-point which we denote by T^*. Consider the 3-valued interpretation $I^* = \langle T^*, L - \Gamma(T^*) \rangle$. We now prove that I^* is in fact the F-least partial stable model of P. By definition I^* is a partial stable model of P. We still have to prove that it is the F-least. Let $I = \langle T, F \rangle$ be a partial stable model of P, i.e., $\Gamma^2(T) = T$ and $F = L - \Gamma(T)$. We have that $T^* \subseteq T$ since T^* is the least fixed-point of Γ^2. Using the fact that Γ is antitonic we have that $\Gamma(T) \subseteq \Gamma(T^*)$. Therefore, $L - \Gamma(T^*) \subseteq L - \Gamma(T)$, thus proving that $I^* \leq_F I$.

We can now define the well-founded semantics of a normal $\mathcal{L}$-parametrized logic program.

Definition 11. *The well-founded semantics of a normal $\mathcal{L}$-parametrized logic program P is defined by its unique F-least partial stable model.*

2.3 Particular Cases

For soundness reasons it is important to show that the non-parametrized case is a special case of our approach, using an appropriate choice of the parameter logic. This is indeed the case for normal and extended logic programs.

Normal Logic Programs. Suppose that we fix an alphabet $\mathcal{A}$ for building the language of normal logic programs. Since we want the class of $\mathcal{L}$-parametrized logic programs to coincide with the class of normal logic programs over $\mathcal{A}$, then the language L of the parameter logic $\mathcal{L}$ must be the set of atoms over $\mathcal{A}$. We then have that $L = \mathcal{H}$, where $\mathcal{H}$ is the Herbrand base of $\mathcal{A}$. Now that we have fixed the language L of $\mathcal{L}$ we still have to define the Tarskian consequence relation $\vdash_{\mathcal{L}}$ over L. At first sight it seems that we have a large range of possibilities for defining $\vdash_{\mathcal{L}}$. Nevertheless, the low expressivity of the language enforces that the only reasonable consequence relation we can choose is the trivial one, i.e. for every $T \cup \{\varphi\} \subseteq L$ we have that $T \vdash_{\mathcal{L}} \varphi$ iff $\varphi \in T$. Therefore, $Th(\mathcal{L}) = \mathcal{P}(L)$, i.e. the theories of $\mathcal{L}$, are precisely the sets of atoms over $\mathcal{A}$. Recall that the notion of parametrized interpretation, both 2 or 3-valued, is based on theories of $\mathcal{L}$. Therefore, by construction, the parametrized versions of stable model semantics and well-founded semantics coincide with the classical ones for normal logic programs.

Extended Logic Programs. Suppose that we fix an alphabet $\mathcal{A}$ for building the language of extended logic programs. This language is enriched with a second negation $\neg$, usually called explicit negation, which is allowed to appear in front of the atoms. An objective literal is either an atom or the explicit negation of an atom.

Recall that we want the class of $\mathcal{L}$-parametrized logic programs to coincide with the class of extended logic programs over $\mathcal{A}$. Therefore, we need to take the language L of the parameter logic $\mathcal{L} = \langle L, \vdash \rangle$ as $L = \mathcal{H} \cup \{\neg p : p \in \mathcal{H}\}$. The set $\mathcal{H} \cup \{\neg p : p \in \mathcal{H}\}$ is usually called the extended Herbrand base of P.

In the case of programs with explicit negation there is no a general consensus with respect to the semantics. In fact, there are two main approaches: the classical one that assumes the explosion principle of negation; and the paraconsistent approach that rejects the explosion principle of negation.

It is very interesting that our parametrized approach allows to easily explain why two different approaches to the semantics of extended logic programs exist. In fact, this can be explained by the fact that only two consequence relations can naturally be defined over the language L.

The first consequence relation over L, denoted by $\vdash_1$, assumes the explosion principle and is such that, for every $T \cup \varphi$, we have $T \vdash_1 \varphi$ if $\varphi \in T$ or $\{p, \neg p\} \subseteq T$

for some atom p. It can be readily proved that, taking $\mathcal{L}_1 = \langle L, \vdash_1 \rangle$, the $\mathcal{L}_1$-parametrized stable model semantics of an extended logic program coincides with its answer set semantics. The second consequence relation over L, denoted by $\vdash_2$, does not assume the explosion principle and, therefore, is such that, for every $T \cup \varphi$, we have $T \vdash_2 \varphi$ if $\varphi \in T$. The consequence relation $\vdash_2$ is in fact the 4-valued Belnap paraconsistent logic *Four* restricted to this more restricted language. It can be readily proved that, taking $\mathcal{L}_2 = \langle L, \vdash_2 \rangle$, the $\mathcal{L}_2$-parametrized stable model semantics of an extended logic program coincides with its paraconsistent answer set semantics.

Beyond Extended Logic Programs. Let us now consider a full classical language L built over a set $\mathcal{P}$ of propositional symbols using the usual connectives $(\neg, \vee, \wedge, \Rightarrow)$. Of course, many consequence relations can be defined over this language. Here we only focus on classical logic, Belnap's paraconsistent logic and intuitionistic logic. Consider the following programs:

$$P_1 \begin{cases} p \leftarrow \neg q \\ p \leftarrow q \end{cases} \qquad P_2 \ \{ p \leftarrow \neg q \vee q \qquad P_3 \begin{cases} q \leftarrow \\ (q \vee s) \Rightarrow p \leftarrow \\ r \leftarrow p \end{cases}$$

$$P_4 \begin{cases} r \leftarrow \\ \neg p \leftarrow \\ (p \vee q) \leftarrow r \\ s \leftarrow q \end{cases} \qquad P_5 \ \{ p \leftarrow not\ q, not\ \neg q \qquad P_6 \ \{ p \leftarrow not\ (q \vee \neg q)$$

Example 1 (Classical logic)
Let $\mathcal{L} = \langle L, \vdash_{CPL} \rangle$ be Classical Propositional Logic (CPL) over the language L. Let us study in detail the semantics of P_1. Note that every theory of CPL that does not contain neither p nor $\neg p$ satisfies P_1. In particular, the set $Theo$ of theorems of CPL is a model of P_1. So, $S^{CPL}_{P_1} = Theo$. This means that $p, \neg p, q, \neg q \notin S^{CPL}_{P_1}$.

Using the same kind of reasoning we can conclude that $S^{CPL}_{P_2} = \{p\}^{\vdash}$. So, in the case of P_2 we have that $p \in S^{CPL}_{P_2}$. We can also easily conclude that $r \in S^{CPL}_{P_3}$ and $s \in S^{CPL}_{P_4}$.

In the case of P_5 its stable models are the theories of CPL that contain p and do not contain q and $\neg q$. Therefore, we can conclude that $p \in S^{CPL}_{P_5}$. In the case of P_6, since $(p \vee \neg p) \in T$ for every theory T of CPL we can conclude that the only stable model of P_6 is the set $Theo$ of theorems of CPL. Therefore $p \notin S^{CPL}_{P_6}$.

Example 2 (Paraconsistent logic)
Consider now $\mathcal{L} = \langle L, \vdash_4 \rangle$ the 4-valued Belnap paraconsistent logic *Four*. Consider the program P_4. Contrarily to the case of CPL, in *Four* it is not the case that $\neg p, (p \vee q) \vdash_4 q$. Therefore we have that $q, s \notin S^{Four}_{P_4}$.

Example 3 (Intuitionistic logic)
Let now $\mathcal{L} = \langle L, \vdash_{IPL} \rangle$ be the propositional intuitionistic logic IPL. It is well-known that $q \vee \neg q$ is not a theorem of IPL. Therefore, considering program

P_2 we have $S_{P_2}^{IPL} = \emptyset^{\vdash_{IPL}}$. So, contrarily to the case of CPL, we have that $p \notin S_{P_2}^{IPL}$. Using the same idea for program P_6 we can conclude, contrarily to the case of CPL, that $p \in S_{P_6}^{IPL}$.

3 MKNF

In this section we show that our approach is general enough to capture the semantics of MKNF hybrid knowledge bases [12,14,10], which tightly combines normal logic programs with description logic (DL) based ontologies. More precisely, the goal of this section is to prove that, taking first-order logic as the parameter logic, and translating every DL formula in the ontology into a fact of the corresponding parametrized program, we exactly obtain the MKNF semantics. Moreover, by doing so, the expressiveness of the language can naturally be extended by allowing complex DL formulas to appear anywhere in rules, as it happens e.g. in [11] but in this latter one only for definite programs.

Before we come to this main result, we recall some basic notions of MKNF, and then prove some required intermediate results.

Let $\mathcal{O}$ be a DL database. Consider a first-order signature Σ such that Σ contains the equality predicate $\approx$, all atomic concepts from $\mathcal{O}$ as unary predicates, all atomic roles from $\mathcal{O}$ as binary predicates and all individuals of $\mathcal{O}$ as constants. Let $FOL_\Sigma = \langle L_\Sigma, \vdash_{FOL_\Sigma} \rangle$ denote first-oder logic over the first-order language L_Σ obtained from Σ. As it is usual in the MKNF approach, we assume that $\mathcal{O}$ can be translated into a formula $\pi(\mathcal{O})$ of function free first-order logic over Σ. Given a MKNF knowledge base $\mathcal{K} = \langle \mathcal{O}, P \rangle$ we denote by $P_\mathcal{K}$ the FOL_Σ-parametrized logic program obtained from $\mathcal{K}$ such that $P_\mathcal{K} = P \cup \{\pi(\mathcal{O}) \leftarrow\}$. We denote by $\mathcal{I}$ be the set of all Herbrand interpretations over Σ and by Δ the Herbrand universe of Σ. Following [14] we also assume that in every member of $\mathcal{I}$ the equality predicate $\approx$ is interpreted as a congruence relation. Recall that the language of MKNF, here denoted by $L_\Sigma^{\mathbf{K}}$, is obtained by extending the first-order language over Σ with the modal operators **K** and **not**. A MKNF structure is a triple $\langle I, M, N \rangle$ where $M \cup N \cup \{I\} \subseteq \mathcal{I}$ and M and N are non-empty. Satisfiability of MKNF formulas in a MKNF structure $\langle I, M, N \rangle$ is defined as follows:

$\langle I, M, N \rangle \Vdash p(t_1, \ldots, t_n)$ iff $p(t_1, \ldots, t_n)$ is true in I
$\langle I, M, N \rangle \Vdash \neg\varphi$ iff $\langle I, M, N \rangle \nVdash \varphi$
$\langle I, M, N \rangle \Vdash \varphi \wedge \psi$ iff $\langle I, M, N \rangle \Vdash \varphi$ and $\langle I, M, N \rangle \Vdash \psi$
$\langle I, M, N \rangle \Vdash \exists x\varphi$ iff $\langle I, M, N \rangle \Vdash \varphi[\alpha/x]$ for some $\alpha \in \Delta$
$\langle I, M, N \rangle \Vdash \mathbf{K}\varphi$ iff $\langle J, M, N \rangle \Vdash \varphi$ for all $J \in M$
$\langle I, M, N \rangle \Vdash \mathbf{not}\varphi$ iff $\langle J, M, N \rangle \nVdash \varphi$ for some $J \in N$

A MKNF interpretation is a nonempty set $M \subseteq \mathcal{I}$. A MKNF interpretation M is a model of a formula φ, denoted by $M \Vdash \varphi$, if $\langle I, M, M \rangle \Vdash \varphi$ for every $I \in M$, and for each MKNF interpretation M' such that $M \subset M'$, we have that $\langle I', M', M \rangle \nVdash \varphi$ for some $I' \in M'$.

We write $M \Vdash T$ to denote that for every $I \in M$, we have that $I \Vdash \varphi$ for every $\varphi \in T$.

For $M \subseteq \mathcal{I}$ consider $T_M = \{\varphi \in L_\Sigma : I \Vdash \varphi \text{ for every } I \in M\}$. Conversely, given $T \subseteq L_\Sigma$ consider $M_T = \{I \in \mathcal{I} : I \Vdash \varphi \text{ for every } \varphi \in T\}$. It is easy to

see that if $M \subseteq M'$ then $T_{M'} \subseteq T_M$. Conversely, if $T \subseteq T'$ then $M_{T'} \subseteq M_T$. It is also easy to see that the inclusions $M \subseteq M_{T_M}$ and $T \subseteq T_{M_T}$ always hold, whereas the respective converse inclusions do not always hold. When the first inclusion holds, that is $M = M_{T_M}$, the set M is called *complete*. Let $\mathcal{M}$ denote the set of all complete subsets of $\mathcal{I}$. It is easy to see that the second converse inclusion, $T_{M_T} \subseteq T$, holds for every $T \in Th(FOL_\Sigma)$.

Lemma 1. *The function* $\phi : \mathcal{M} \rightarrow Th(FOL_\Sigma)$ *such that* $M \mapsto T_M$ *is a bijection.*

Proof. We start by proving that ϕ is well-defined, that is, that T_M is a FOL_Σ-theory. Suppose that $T_M \vdash_{FOL_\Sigma} \varphi$. Then, using the fact that M is complete, we have the following sequence of equivalent sentences:

for every $I \in \mathcal{M}$, if $I \Vdash \delta$ for every $\delta \in T_M$ then $I \Vdash \varphi$ iff for every $I \in \mathcal{M}$, if $I \in M_{T_M}$ then $I \Vdash \varphi$ iff for every $I \in \mathcal{M}$, if $I \in M$ then $I \Vdash \varphi$. Therefore, we have that $\varphi \in T_M$ and we can conclude that T_M is a FOL_Σ-theory.

Let us now prove that ϕ is injective. Let $M_1, M_2 \in \mathcal{M}$ such that $T_{M_1} = T_{M_2}$. Clearly we have that $M_{T_{M_1}} = M_{T_{M_2}}$. Since M_1, M_2 are complete we have that $M_1 = M_{T_{M_1}} = M_{T_{M_2}} = M_2$.

It remains to be proved that ϕ is surjective. Let T be a FOL_Σ-theory. First we prove that M_T is complete. Since the inclusion $M_T \subseteq M_{T_{M_T}}$ always holds, it remains to prove the converse inclusion. Let $I \in M_{T_{M_T}}$. Then, $I \Vdash T_{M_T}$. Since $T \subseteq T_{M_T}$ we have that $I \Vdash T$. Therefore, $I \in M_T$ and the inclusion $M_{T_{M_T}} \subseteq M_T$ holds. It just remains to be proved that $\phi(M_T) = T$, that is, $T_{M_T} = T$. We prove the inclusion $T_{M_T} \subseteq T$ since the reverse one always holds. Let $\varphi \in T_{M_T}$. Then $M_T \Vdash \varphi$ and by definition $T \vdash_{FOL_\Sigma} \varphi$. Since T is a FOL_Σ-theory we conclude that $\varphi \in T$.

Lemma 2. *Let* $M, M' \in \mathcal{M}$ *and* $T, T' \in Th(FOL_\Sigma)$. *Then*

1. $T \subset T'$ *iff* $M_{T'} \subset M_T$;
2. $M \subset M'$ *iff* $T_{M'} \subset T_M$.

Proof. 1. Suppose $T \subset T'$. Then, if $I \in M_{T'}$ then $I \in M_T$. Suppose now that $M_{T'} \subset M_T$. Using Lemma 1 we have that $T_{M_T} = T \subset T' = T_{M_{T'}}$. 2. Suppose $M \subset M'$. Then $T_{M'} \subset T_M$. Suppose now that $T_{M'} \subset T_M$. Using Lemma 1 we have that $M_{T_M} = M \subset M' = M_{T_{M'}}$

Lemma 3. *Let* $M \cup M' \cup \{I\} \subseteq \mathcal{I}$, $T \in Th(FOL_\Sigma)$ *and* $\varphi \in L_\Sigma$. *Then,*

1. $\langle I, M_T, M' \rangle \Vdash \mathbf{K}\varphi$ *iff* $\varphi \in T$;
2. $\langle I, M, M' \rangle \Vdash \mathbf{K}\varphi$ *iff* $\varphi \in T_M$;
3. $\langle I, M', M_T \rangle \Vdash \mathbf{not}\varphi$ *iff* $\varphi \notin T$;
4. $\langle I, M', M \rangle \Vdash \mathbf{not}\varphi$ *iff* $\varphi \notin T_M$.

Proof. 1. $\langle I, M_T, M' \rangle \Vdash \mathbf{K}\varphi$ iff $M_T \Vdash \varphi$ iff $\varphi \in T$; 2. $\langle I, M, M' \rangle \Vdash \mathbf{K}\varphi$ iff $M \Vdash \varphi$ iff $\varphi \in T_M$; 3. $\langle I, M', M_T \rangle \Vdash \mathbf{not}\varphi$ iff there exists $I' \in M_T$ such that $I' \nVdash \varphi$ iff (since $T_{M_T} = T$) $\varphi \notin T$; 4. $\langle I, M', M \rangle \Vdash \mathbf{not}\varphi$ iff there exists $I' \in M$ such that $I' \nVdash \varphi$ iff $\varphi \notin T_M$.

Given $\Phi \subseteq L_{\Sigma}^{\mathbf{K}}$ and $M \subseteq \mathcal{I}$ consider $\Gamma(\Phi, M) = \{M' \subseteq \mathcal{I} : M' \text{ is maximal such that } \langle I, M', M\rangle \Vdash \Phi, \text{ for every } I \in M'\}$. Note that, by definition, M is a MKNF model of Φ iff $M \in \Gamma(\Phi, M)$. Moreover, it can be readily proved that if M is a MKNF model of some $\Phi \subseteq L_{\Sigma}^{\mathbf{K}}$ then M is complete. Given $\Phi \subseteq L_{\Sigma}^{\mathbf{K}}$ and $T \in Th(FOL_{\Sigma})$ consider also the set $\Gamma_0(\Phi, T) = \{T' \in Th(FOL_{\Sigma}) : T' \text{ minimal such that } \langle I, M_{T'}, M_T\rangle \Vdash \Phi, \text{ for every } I \in M_{T'}\}$.

Lemma 4. *Let $M \cup M' \cup \{I\} \subseteq \mathcal{I}$ and P a normal FOL_{Σ}-parametrized logic program. Then, $\langle I, M, M'\rangle \Vdash \pi(P)$ iff $\langle I, M_{T_M}, M'\rangle \Vdash \pi(P)$.*

Proof. Recall that $\pi(P) = \bigwedge_{r\in P}\pi(r)$, and that if $r = \varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \varphi_1, \ldots, not\ \varphi_m$ then $\pi(r) = (\mathbf{K}\psi_1 \wedge \ldots \wedge \mathbf{K}\psi_n \wedge \mathbf{not}\varphi_1 \wedge \ldots \wedge \mathbf{not}\varphi_m) \Rightarrow \mathbf{K}\varphi$. Consider the following sequence of equivalent conditions: $\langle I, M, M'\rangle \Vdash \pi(r)$ iff $\langle I, M, M'\rangle \Vdash \mathbf{K}\varphi$ whenever $\langle I, M, M'\rangle \Vdash \mathbf{not}\varphi_i$ for every $i \in \{1, \ldots, m\}$ and $\langle I, M, M'\rangle \Vdash \mathbf{K}\psi_j$ for every $j \in \{1, \ldots, n\}$ iff (Lemma 3) $\varphi \in T_M$ whenever $\varphi_i \notin T_{M'}$ for every $i \in \{1, \ldots, m\}$ and $\psi_j \in T_M$ for every $j \in \{1, \ldots, n\}$ iff (Lemma 3) $\langle I, M_{T_M}, M'\rangle \Vdash \mathbf{K}\varphi$ whenever $\langle I, M_{T_M}, M'\rangle \Vdash \mathbf{not}\varphi_i$ for every $i \in \{1, \ldots, m\}$ and $\langle I, M_{T_M}, M'\rangle \Vdash \mathbf{K}\psi_j$ for every $j \in \{1, \ldots, n\}$ iff $\langle I, M_{T_M}, M'\rangle \Vdash \pi(r)$.

Proposition 1. *Let $M \in \mathcal{M}$, $I \in \mathcal{I}$ and P a normal FOL_{Σ}-parametrized logic program. Then, $M \in \Gamma(\pi(P), M)$ iff $T_M \in \Gamma_0(\pi(P), T_M)$.*

Proof. Let us first assume that $M \in \Gamma(\pi(P), M)$. Then M is maximal such that $\langle I, M, M\rangle \Vdash \pi(P)$ for every $I \in M$. Therefore, $\langle I, M_{T_M}, M_{T_M}\rangle \Vdash \pi(P)$. We still have to prove the minimality condition for T_M. Suppose there exists $T' \subset T_M$ such that $\langle I, M_{T'}, M_{T_M}\rangle \Vdash \pi(P)$. Then, using Lemma 2 we have $M_{T_M} = M \subset M_{T'}$, which contradicts the maximality of M. Therefore, we can conclude that $T_M \in \Gamma_0(\pi(P), T_M)$.

Suppose now that $T_M \in \Gamma_0(\pi(P), T_M)$. Then T_M is minimal such that $\langle I, M_{T_M}, M_{T_M}\rangle \Vdash \pi(P)$. Since $M \in \mathcal{M}$, i.e., $M = M_{T_M}$, we have that $\langle I, M, M\rangle \Vdash \pi(P)$. We still have to prove the maximality condition for M. Suppose there exists $M \subset M'$ such that $\langle I, M', M\rangle \Vdash \pi(P)$. Using Lemma 4 we have that $\langle I, M_{T_{M'}}, M_{T_M}\rangle \Vdash \pi(P)$. Using Lemma 2 we have that $T_{M'} \subset T_M$, which contradicts the minimality condition of T_M. Therefore, we can conclude that $M \in \Gamma(\pi(P), M)$.

Lemma 5. *Let $T \in Th(FOL_{\Sigma})$ and P a normal FOL_{Σ}-parametrized logic program. Then, $\Gamma_0(\pi(\frac{P}{T}), T) = \Gamma_0(\pi(P), T)$.*

Proof. Let $T \in Th(FOL_{\Sigma})$. It suffices to prove that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(P)$ iff $\langle I, M_{T'}, M_T\rangle \Vdash \pi(\frac{P}{T})$. Assume first that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(P)$. Recall that every rule $r = \varphi \leftarrow \psi_1, \ldots, \psi_n$ of $\frac{P}{T}$ is obtained from a rule $\overline{r} = \varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \varphi_1, \ldots, not\ \varphi_m$ of P such that $\varphi_i \notin T$ for every $i \in \{1, \ldots, m\}$. Then, using Lemma 3 we have that $\langle I, M_{T'}, M_T\rangle \Vdash \mathbf{not}\varphi_i$ for every $i \in \{1, \ldots, m\}$. Therefore, $\langle I, M_{T'}, M_T\rangle \Vdash \pi(r)$ iff $\langle I, M_{T'}, M_T\rangle \Vdash \pi(\overline{r})$. Since we are assuming that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(P)$ we can conclude that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(\frac{P}{T})$.

Let us now assume that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(\frac{P}{T})$ and consider the rule $r = \varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \varphi_1, \ldots, not\ \varphi_m$ of P. We have two cases:

Case 1: $\varphi_i \in T$ *for some* $i \in \{1,\ldots,m\}$. Then, by Lemma 3 we have that $\langle I, M_{T'}, M_T\rangle \nVdash \mathbf{not}\varphi_i$. Therefore, $\langle I, M_{T'}, M_T\rangle \Vdash \pi(r)$.

Case 2: $\varphi_i \notin T$ *for every* $i \in \{1,\ldots,m\}$. Then, by Lemma 3 we have that $\langle I, M_{T'}, M_T\rangle \Vdash \mathbf{not}\varphi_i$ for every $i \in \{1,\ldots,m\}$. We then have that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(r)$ iff $\langle I, M_{T'}, M_T\rangle \Vdash \pi(\overline{r})$, where $\overline{r} = \varphi \leftarrow \psi_1,\ldots,\psi_n \in \frac{P}{T}$. Since we are assuming that $\langle I, M_{T'}, M_T\rangle \Vdash r'$ for every rule r' of $\pi(\frac{P}{T})$ we can conclude that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(r)$.

Therefore we can conclude that $\langle I, M_{T'}, M_T\rangle \Vdash \pi(P)$.

Lemma 6. *Let* $T \in Th(FOL_\Sigma)$ *and* P *a definite* FOL_Σ*-parametrized logic program. Then* $\Gamma_0(\pi(P), T)$ *is the set of all* FOL_Σ*-parametrized stable models of* P.

Proof. Let $T \in Th(FOL_\Sigma)$. Then we have the following sequence of equivalent sentences: T is a FOL_Σ-parametrized stable model of P iff T is minimal such that, for every rule $r = \varphi \leftarrow \psi_1,\ldots,\psi_n$, we have that $\varphi \in T$ whenever $\psi_i \in T$ for every $i \in \{1,\ldots,n\}$ iff (Lemma 3) T is minimal such that, for every rule $r = \varphi \leftarrow \psi_1,\ldots,\psi_n$, we have that $\langle I, M_T, M_T\rangle \Vdash \mathbf{K}\varphi$ whenever $\langle I, M_T, M_T\rangle \Vdash \mathbf{K}\psi_i$ for every $i \in \{1,\ldots,n\}$ iff T is minimal such that $\langle I, M_T, M_T\rangle \Vdash \pi(r)$ for every rule r of P iff $T \in \Gamma_0(\pi(P), T)$.

Theorem 3. M *is a MKNF model of* $\mathcal{K}$ *iff* T_M *is a* FOL_Σ*-parametrized stable model of* $P_\mathcal{K}$.

Proof. Just consider the following sequence of equivalent sentences:

M is a MKNF model of $\mathcal{K}$ iff M is a MKNF model of $\pi(P_\mathcal{K})$ iff $M \in \Gamma(\pi(P_\mathcal{K}), M)$ iff (Proposition 1) $T_M \in \Gamma_0(\pi(P_\mathcal{K}), T_M)$ iff (Lemma 5) $T_M \in \Gamma_0(\pi(\frac{P_\mathcal{K}}{T_M}), T_M)$ iff (Lemma 6) T_M is a FOL_Σ-parametrized stable model of $\frac{P_\mathcal{K}}{T_M}$ iff (by definition) T_M is a FOL_Σ-parametrized stable model of $P_\mathcal{K}$.

Using Lemma 1 the following corollary of Theorem 3 is immediate.

Corollary 1. $T \in Th(FOL_\Sigma)$ *is a* FOL_Σ*-parametrized stable model of* $P_\mathcal{K}$ *iff* M_T *is a MKNF model of* $\mathcal{K}$.

Example 4. Let $\mathcal{L}$ be a description logic seen as a fragment of FOL. The following program (P_7) is an adaptation of an example taken from [14].

$$\begin{array}{ll} NotMarried \equiv \neg Married \leftarrow & NotMarried(x) \leftarrow p(x), not\ Married(x) \\ NotMarried \sqsubseteq HighRisk \leftarrow & Discount(x) \leftarrow Spouse(x,y), p(x), p(y) \\ \exists Spouse.\top \sqsubseteq Married \leftarrow & p(Jonh) \leftarrow \end{array}$$

Note that in our approach the combination of an ontology with a rule system can be done in a natural way, simply by adding the ontology elements as facts of the rule system. In fact, we are able to rewrite P_7 in order to remove its first rule, which is nothing but an artificial tool to overcome the impossibility of having complex DL formulas in the head of MKNF rules (in this case, having the classical negation of an atom in a head). Moreover, we may also add bodies to the

facts coming from the ontology. E.g. we can add a non-monotonic condition to the second statement of P_7 above, to state that non married are only considered high-risk in non exceptional periods, obtaining P_8:

$\neg Married \sqsubseteq HighRisk \leftarrow not\ exceptionalPeriod$
$\exists Spouse.\top \sqsubseteq Married \leftarrow$ $\neg Married(x) \leftarrow p(x), not\ Married(x)$
$Discount(x) \leftarrow Spouse(x,y), p(x), p(y)$ $p(Jonh) \leftarrow$

Let us study the stable model semantics of this program. If I is a 2-valued interpretation such that $I(Married(Jonh)) = 1$ then $\Gamma(I)$ is the least model of the following program $\frac{P_8}{I}$:
$\neg Married \sqsubseteq HighRisk \leftarrow$
$\exists Spouse.\top \sqsubseteq Married \leftarrow$
$Discount(x) \leftarrow Spouse(x,y), p(x), p(y)$ $p(Jonh) \leftarrow$

It is clear that the smallest model of $\frac{P_8}{I}$ does not contain $Married(Jonh)$, and so, such interpretation I cannot be a stable model. Therefore, every stable model must satisfy $\neg Married(Jonh)$ and consequently $HighRisk(Jonh)$.

Suppose that we obtain P_9 by adding to P_8 the following facts: $p(Bill) \leftarrow$, $\exists Spouse.\top(Bill) \leftarrow$, and $exceptionalPeriod \leftarrow$. Note that although every stable model now contains $\neg Married(John)$, we no longer conclude $HighRisk(Jonh)$ since we have $exceptionalPeriod$. Every stable model of P_9 contains $Married(Bill)$. So, the Stable Model Semantics of P_9 does not entail $\neg Married(Bill)$ nor $HighRisk(Bill)$.

Suppose now that instead we add to P_8 the facts: $Spouse(Bob, Ann) \leftarrow$, $p(Bob) \leftarrow$, and $p(Ann) \leftarrow$. Every stable model now contains $Discount(Bob)$, and so the Stable Model Semantics entails $Discount(Bob)$.

4 Conclusions

We have introduced the novel notion of parametrized logic program along with several motivating examples, and showed how some usual approaches to the semantics of logic programs can be obtained as particular choices of the parameter logic. We gave a contribution to the important problem of combining rules and ontologies, by capturing MKNF semantics of hybrid knowledge bases as particular case and, moreover, by extending the expressivity of its language. Note that, as a consequence, we are also able to capture Description Logic Programs (DLP) [9] since we can write arbitrary DL formulas in the heads and bodies of rules. Though here we only explored the stable models semantics of MKNF knowledge bases, our approach also naturally yields a well founded semantics for such knowledge bases.

The work raises several interesting paths for future research. One, which is already ongoing is the generalization to the parametrized case of the here-there theories [15], this way allowing for combining in a general way logic programming connectives with formulas of a parameter logic. This work would also generalise for any parameter logic the general default logic of [17], which is fixed for propositional classical logic. Future work also includes a detailed study of particular

choices of the parameter logic which seem very promising. One such example is the case where the parameter logic is temporal logic, to obtain logic programs expressive enough to reason about temporal logic formulas.

This first paper on parametrized logic programs is focused on the definition of the semantics and on showing its interest for capturing at least one known combination of logic programming with other logics (viz. description logic in MKNF) and, as such, left out a study on decidability and complexity issues, which must also be subject of future work.

References

1. Caleiro, C., Sernadas, C., Sernadas, A.: Parameterisation of logics. In: Fiadeiro, J.L. (ed.) WADT 1998. LNCS, vol. 1589, pp. 48–62. Springer, Heidelberg (1999)
2. Carnielli, W.A., Coniglio, M.E., Gabbay, D., Gouveia, P., Sernadas, C.: Analysis and Synthesis of Logics - How To Cut And Paste Reasoning Systems. Applied Logic, vol. 35. Springer, Heidelberg (2008)
3. Damásio, C.V., Pereira, L.M.: Antitonic logic programs. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 379–392. Springer, Heidelberg (2001)
4. Fitting, M.: Bilattices and the semantics of logic programming. J. Log. Program. 11(2), 91–116 (1991)
5. Gabbay, D.: Fibring logics. Oxford University Press, Oxford (1999)
6. Van Gelder, A.: The alternating fixpoint of logic programs with negation. J. Comput. Syst. Sci. 47(1), 185–221 (1993)
7. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080 (1988)
9. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: WWW, pp. 48–57 (2003)
10. Knorr, M., Alferes, J.J., Hitzler, P.: A well-founded semantics for hybrid mknf knowledge bases. In: Description Logics (2007)
11. Krötzsch, M., Rudolph, S., Hitzler, P.: Elp: Tractable rules for owl 2. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 649–664. Springer, Heidelberg (2008)
12. Lifschitz, V.: Minimal belief and negation as failure. Artif. Intell. 70(1-2), 53–72 (1994)
13. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. 2(4), 526–541 (2001)
14. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: IJCAI, pp. 477–482 (2007)
15. Pearce, D.: Equilibrium logic. Ann. Math. Artif. Intell. 47(1-2), 3–41 (2006)
16. Wójcicki, R.: Theory of Logical Calculi. Synthese Library. Kluwer Academic Publishers, Dordrecht (1988)
17. Zhou, Y., Lin, F., Zhang, Y.: General default logic. Annals of Mathematics and Artificial Intelligence 57(2), 125–160 (2009)

Counterexample Guided Abstraction Refinement Algorithm for Propositional Circumscription*

Mikoláš Janota[1], Radu Grigore[2], and Joao Marques-Silva[3]

[1] INESC-ID, Lisbon, Portugal
[2] Queen Mary, University of London
[3] University College Dublin, Ireland

Abstract. Circumscription is a representative example of a nonmonotonic reasoning inference technique. Circumscription has often been studied for first order theories, but its propositional version has also been the subject of extensive research, having been shown equivalent to extended closed world assumption (ECWA). Moreover, entailment in propositional circumscription is a well-known example of a decision problem in the second level of the polynomial hierarchy. This paper proposes a new Boolean Satisfiability (SAT)-based algorithm for entailment in propositional circumscription that explores the relationship of propositional circumscription to minimal models. The new algorithm is inspired by ideas commonly used in SAT-based model checking, namely counterexample guided abstraction refinement. In addition, the new algorithm is refined to compute the theory closure for generalized close world assumption (GCWA). Experimental results show that the new algorithm can solve problem instances that other solutions are unable to solve.

1 Introduction

Closed world reasoning (CWR) and circumscription (CIRC) are well-known nonmonotonic reasoning techniques, that find a wide range of practical applications. Part of the interest in these techniques is that they bring us closer to how humans reason [16,18,17]. While these techniques have been studied in the context of both first-order and propositional logic, this paper addresses the propositional case. Research directions that have characterized the study of nonmonotonic reasoning techniques include expressiveness, computational complexity, applications and algorithms. The different CWR rules proposed in the late 70s and 80s illustrate the evolution in terms of expressive power in first-order and propositional logics. The computational complexity of propositional CWR rules was studied in the early 90s [1,6] and showed that, with few exceptions, the complexity of CWR deduction problems are in the second level of the polynomial hierarchy, being Π_2^{P}-complete [6]. Nonmonotonic reasoning finds a wide range of applications in Artificial Intelligence (AI), but also in description logics [7]

* This work is partially supported by SFI project BEACON (09/IN.1/I2618) and European projects COCONUT (FP7-ICT-217069) and MANCOOSI (FP7-ICT-214898).

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 195–207, 2010.
© Springer-Verlag Berlin Heidelberg 2010

and in interactive configuration [13], among many others. Finally, different algorithms have been proposed over the years, examples of which include minimal model resolution [21], tableau calculus [19], Quantified Boolean Formula (QBF) solvers [5] and Disjunctive Logic Programming (DLP) [15,12,20].

The main contribution of this paper is to propose a new algorithm for solving the deduction problem for the propositional version of some CWR rules and for propositional circumscription. The new algorithm is based on iterative calls to a SAT solver, and is motivated by the practical success of modern SAT solvers. However, given the complexity class of entailment for CWR rules, a SAT solver can be expected to be called an exponential number of times in the worst case, or be required to process an exponentially large input. To cope with this issue, we utilize a technique inspired in counterexample guided abstraction refinement (CEGAR), widely used in model checking [3]. One of the key ideas of the new algorithm is that we try to prove a stronger formula, which is weakened if it turns out to be too strong. Based on this idea we develop an algorithm that decides entailment in circumscription. Further, we refine the algorithm to compute the closure of a formula defined by one of the variants of CWR, namely GCWA. As a result, the main contributions of the paper can be summarized as follows: (i) A novel algorithm for propositional circumscription that does not require an enumeration of all minimal models or prime implicates; (ii) Specialization of this algorithm to compute variables that are 0 in all minimal models; and (iii) Computing the closure of GCWA.

2 Preliminaries

All variables are propositional, and represented by a finite set V. A *Conjunctive Normal Form* (CNF) formula ϕ is a conjunction of *clauses*, which are disjunctions of *literals*, which are possibly negated variables. A formula ϕ can also be viewed as a set of sets of literals. The two representations are used interchangeably in this paper. A clause is called *positive*, if it contains only positive literals. Arbitrary Boolean formulas will also be considered, for which the standard definitions apply. A *variable assignment* ν is a total function from V to $\{0,1\}$. In the text, a variable assignment is represented as $\{x_1^{v_1}, \ldots, x_n^{v_n}\}$ where $V = \{x_1, \ldots, x_n\}$ and $v_i \in \{0,1\}$, $i \in 1..n$. For a variable assignment ν and a formula ϕ we write $\nu \models \phi$ to denote that ν satisfies ϕ. In this case, ν is called a *model* of ϕ. We write $\phi \models \psi$ if the models of ϕ are also models of ψ. Given a set of variables $S \subseteq V$ and $v \in \{0,1\}$, the expression $\phi[S \mapsto v]$ denotes the formula ϕ with all variables in S replaced with v.

2.1 Minimal Models

Minimal models are widely used in nonmonotonic reasoning and AI in general. To introduce minimal models, we consider the bitwise ordering on variable assignments. For variable assignments ν and μ we write $\nu \leq \mu$ and say that ν is *smaller* than μ iff $(\forall x \in V)(\nu(x) \leq \mu(x))$. We write $\nu < \mu$ and say that ν is

strictly smaller than μ iff $\nu \leq \mu$ and $\nu \neq \mu$. A model ν of ϕ is a *minimal model* iff there is no model of ϕ strictly smaller than ν. Finally, we write $\phi \models_{\min} \psi$ if ψ holds in all minimal models of ϕ.

Proposition 1. *The models of formula ϕ that are strictly smaller than some variable assignment ν are the models of the formula*

$$\phi \wedge \bigwedge_{\nu(x)=0} \neg x \wedge \bigvee_{\nu(x)=1} \neg x \tag{1}$$

2.2 Closed World Reasoning

The intuition behind closed world assumption (CWA) reasoning is that facts are not considered to be true unless they were specifically stated. This is motivated by the type of reasoning humans use on an everyday basis. For instance, if Alice asks Bob to buy eggs, Bob will clearly buy eggs. However, he will not buy bread even though Alice has not specified that the bread should not be bought. Traditional mathematical logic behaves differently in this respect: the fact *buy-eggs* trivially entails *buy-eggs* but does not entail the fact $\neg$*buy-bread*.

This intuition has been realized by several different formalisms. Here we present only a small portion of these formalisms and the interested reader is referred to appropriate publications for further reference [1,6,4].

The standard formulation of CWA rules partitions set V into three sets: P, Q and Z, where P denotes the variables to be minimized, Z are the variables that can change when minimizing the variables in P, and Q represents all other (fixed) variables. For any set R, R^+ and R^- denote, respectively, the sets of positive and negative literals from variables in R. Following [1,6], a closure operation is defined for CWR rules as follows:

Definition 1. *Let ϕ be a propositional formula, $\langle P; Q; Z\rangle$ a partition of V, and α a CWR-rule. Then, the* closure *of ϕ with respect to α is defined by,*

$$\alpha(\phi; P; Q; Z) = \phi \cup \{\neg K \mid K \textit{ is free for negation in } \phi \textit{ w.r.t. } \alpha\} \tag{2}$$

Each CWR rule considers a different set of formulas that are free for negation. For each CWR rule below, a formula K is free for negation if and only if the corresponding condition holds:

GCWA (Generalized CWA [18]): K is a positive literal and for every positive clause B such that $\phi \not\models B$ it holds that $\phi \not\models B \vee K$.

EGCWA (Extended GCWA [25]): K is a conjunction of positive literals and for every positive clause B such that $\phi \not\models B$ it holds that $\phi \not\models B \vee K$.

ECWA (Extended CWA [25]): K is an arbitrary formula not involving literals from Z, and for every positive clause B whose literals belong to $P^+ \cup Q^+ \cup Q^-$, such that $\phi \not\models B$, it holds that $\phi \not\models B \vee K$.

We consider only a subset of existing CWR rules. A detailed characterization for existing CWR rules can be found elsewhere [1,6,4].

Observe that a single positive literal is free for negation in both GCWA and EGCWA under the same conditions. Since a positive literal corresponds to some variable, we extend the terminology for variables accordingly.

Definition 2. *A variable x is* free for negation in ϕ *iff for every positive clause B such that $\phi \not\models B$ it holds that $\phi \not\models B \vee v$.*

Another concept closely related to closed world assumption is circumscription. Originally, McCarthy defined circumscription in the context of first order logic as a closure of the given theory that considers only predicates with minimal extension [16]. In propositional logic, circumscription of a formula yields a formula whose models are the minimal models of the original one.

Definition 3. *Consider the sets of variables P, Q and Z introduced above. The circumscription of a formula ϕ is defined as follows:*

$$CIRC(\phi; P; Q; Z) = \phi \wedge (\forall_{P',Z'})((\phi(P'; Q; Z') \wedge (P' \Rightarrow P)) \Rightarrow (P \Rightarrow P')) \quad (3)$$

Where P', Z' are sets of variables s.t. $X' = \{x' \mid x \in X\}$; $\phi(P', Q, Z')$ is obtained from $\phi(P, Q, Z)$ by replacing the variables in P and Z by the corresponding variables in P' and Z'; finally, $P' \Rightarrow P$ stands for $\bigwedge_{x \in P}(x' \Rightarrow x)$.

In the remainder of the paper the sets Z and Q are assumed to be *empty*. The extension to the general case where these sets are not empty is simple and is outlined in an extended version of the article [14].

It is well-known that for the propositional case, circumscription is equivalent to ECWA [9]. Another well-known relationship is the one of both CWR rules and circumscription to minimal models (e.g. [18,1,6]). In particular variables free for negation take value 0 in all minimal models. And, both EGCWA and circumscription entail the same set of facts as the set of minimal models. These relations are captured by the following propositions (adapted from [18,1,6]):

Proposition 2. *A variable x is free for negation in a formula ϕ iff x is assigned value 0 in all minimal models of ϕ.*

Proposition 3. *Let ϕ and ψ be formulas. It holds that $EGCWA(\phi) \models \psi$ iff $\phi \models_{\min} \psi$. And, it holds that $CIRC(\phi) \models \psi$ iff $\phi \models_{\min} \psi$.*

3 Problems

The CWR rules yield the two following problems. The first problem consists of computing the closure of the theory, as defined by the CWR rule. The second problem is that of computing whether a certain fact is entailed by that closure.

If the closure has been computed, standard satisfiability algorithms can be used to solve the entailment problem. However, whereas the closure of GCWA increases the size of the formula by at most a linear number of literals, the

closure of both ECWA and EGCWA may increase the size of the formula by an exponential number of conjuncts of literals. The circumscription of a formula can be constructed easily but gives rise to a QBF formula and our objective is to stay within propositional logic with the ultimate goal of developing purely SAT-based solutions. Hence, this paper focuses on the following problems.

ENTAILS-MIN
instance: formulas ϕ and ψ
question: Does the formula ψ hold in all minimal models of ϕ?

FREE-FOR-NEGATION
instance: formula ϕ and variable $x \in V$
question: Does x take value 0 in all minimal models of ψ?

FREE-FOR-NEGATION-ALL
instance: formula ϕ and a variable $v \in V$
question: What is the set of variables with value 0 in all minimal models of ϕ?

Note that solving ENTAILS-MIN enables answering whether a fact is entailed by ECWA or by circumscription due to Proposition 3. Clearly, the problem FREE-FOR-NEGATION is a special case of ENTAILS-MIN with ψ set to $\neg x$. Solving FREE-FOR-NEGATION-ALL gives us the closure of GCWA.

Interestingly, in terms of complexity, the problem FREE-FOR-NEGATION is not easier than the problem ENTAILS-MIN. Both ENTAILS-MIN and FREE-FOR-NEGATION are Π_2^{P}-complete [6, Lemma 3.1].

4 Computing ENTAILS-MIN

The algorithm we wish to develop will be using a SAT solver. This gives us two objectives. One objective is to construct a propositional formula that corresponds to the validity of $\phi \models_{\min} \psi$. The second objective is to avoid constructing an exponentially large formula. We begin by observing that if $\phi \models_{\min} \psi$ is to hold, then any model of ϕ that violates ψ must *not* be a minimal model.

Proposition 4. *ψ holds in all minimal models of ϕ iff any model ν of ϕ where $\neg\psi$ holds is not a minimal model of ϕ.*

$$[\phi \models_{\min} \psi] \Leftrightarrow [(\forall \nu)\,((\nu \models \phi \wedge \neg\psi) \Rightarrow (\exists \nu')(\nu' < \nu \wedge \nu' \models \phi))]$$

Proposition 4 tells us that whether $\phi \models_{\min} \psi$ holds or not can be decided by deciding whether the following formula is valid:

$$(\forall \nu)\,((\nu \models \phi \wedge \neg\psi) \Rightarrow (\exists \nu')(\nu' < \nu \wedge \nu' \models \phi)) \tag{4}$$

Since our first objective is to find a propositional formula, we need to eliminate $\cdot \models \cdot$ and quantifiers from (4). First, let us focus on the subformula $(\exists \nu')(\nu' < \nu \wedge \nu' \models \phi)$, which expresses that ν is not a minimal model.

Proposition 5. *A model ν of ϕ is* not *minimal iff there exists a set S of variables such that ν is a model of $\phi[S \mapsto 0]$, and $\nu(x) = 1$ for some $x \in S$.*

$$(\exists \nu')(\nu' < \nu \wedge \nu' \models \phi) \Leftrightarrow (\exists S \subseteq V)\,(\nu \models \phi[S \mapsto 0] \wedge (\exists x \in S)(\nu(x) = 1)) \quad (5)$$

Example 1. Let $\phi = \neg x \vee y$. The model $\mu = \{x^0, y^0\}$ is minimal and the right-hand side of (5) is invalid since there is no set S satisfying the condition $(\exists x \in S)(\nu(x) = 1)$. Let $\nu = \{x^0, y^1\}$ and let us choose $S = \{x, y\}$, which yields $\phi[S \mapsto 0] = 1$. ν is *not* minimal and the right-hand side of (5) is valid since $\nu \models 1$ and $\nu(y) = 1$.

Replacing the left-hand side of (5) with the right-hand side of (5) in (4) yields the following formula:

$$(\forall \nu)\,((\nu \models \phi \wedge \neg\psi) \Rightarrow (\exists S \subseteq V)\,(\nu \models \phi[S \mapsto 0] \wedge (\exists x \in S)(\nu(x) = 1))) \quad (6)$$

Removing the universal quantifier and replacing existential quantifiers with the Boolean operator $\vee$ in (6), gives us that (6) holds iff the following formula is a tautology:

$$(\phi \wedge \neg\psi) \Rightarrow \bigvee_{S \in \mathcal{P}(V)} \left(\phi[S \mapsto 0] \wedge \bigvee_{x \in S} x \right) \quad (7)$$

Intuitively, (7) expresses that if ψ is violated in a model of ϕ, then a different model of ϕ is obtained by flipping a set of variables to 0. That this model is indeed different is guaranteed by the condition $\bigvee_{x \in S} x$. The model obtained by the flipping serves as a *witness* of that the model violating ψ is *not* minimal.

If (7) is constructed, its validity can be decided by calling a SAT solver on its negation. However, the formula is too large to construct since it requires considering all subsets of V. Therefore, we construct a stronger version of it that considers only *some* subsets of V. This stronger version is referred to as the *abstraction* of (7) and always has the following form:

$$(\phi \wedge \neg\psi) \Rightarrow \bigvee_{S \in W} \left(\phi[S \mapsto 0] \wedge \bigvee_{x \in S} x \right) \qquad \text{where } W \subseteq \mathcal{P}(V) \quad (8)$$

Each abstraction is determined by a set of sets of variables W. For any W, if the abstraction (8) is shown to be a tautology, then (7) is also a tautology and we are done because we have shown that $\phi \models_{\min} \psi$. If the abstraction is not a tautology, it is either because $\phi \models_{\min} \psi$ does not hold or the abstraction is overly strong—it is too coarse. If the abstraction is shown to be too coarse, a different abstraction must be considered.

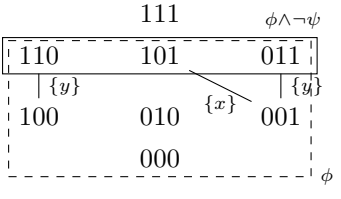
Example 2. Let us show that $\neg x \vee y \models_{\min} \neg y$. First, let us try $W_1 = \{\{y\}\}$, which yields the abstraction $((\neg x \vee y) \wedge y) \Rightarrow \neg x$. This abstraction is not a tautology. In particular, it is violated by the assignment $\{x^1, y^1\}$, which means that flipping y to value 0 in this assignment does not yield a model. Now, let us try $W_2 = \{\{x, y\}\}$, which yields the abstraction $((\neg x \vee y) \wedge y) \Rightarrow 1$. This abstraction is a tautology, which means that any model where y is 1 can be turned into another model by flipping both x and y to 0. Therefore, $\neg x \vee y \models_{\min} \neg y$.

input : formulas ϕ and ψ
output: true iff $\phi \models_{\min} \psi$

1 $\omega \leftarrow \phi \wedge \neg\psi$
2 **while** true **do**
3 $\quad (\mathsf{outc}_1, \nu) \leftarrow \mathtt{SAT}(\omega)$
4 $\quad$ **if** $\mathsf{outc}_1 = \mathsf{false}$ **then**
5 $\quad\quad$ **return** true // no counterexample was found
6 $\quad (\mathsf{outc}_2, \nu') \leftarrow \mathtt{SAT}\left(\phi \wedge \bigwedge_{\nu(x)=0} \neg x \wedge \bigvee_{\nu(x)=1} \neg x\right)$ // find $\nu' < \nu$
7 $\quad$ **if** $\mathsf{outc}_2 = \mathsf{false}$ **then** // ν is minimal
8 $\quad\quad$ **return** false // abstraction cannot be refined
9 $\quad S \leftarrow \{x \in V \mid \nu(x) = 1 \wedge \nu'(x) = 0\}$
10 $\quad \omega \leftarrow \omega \wedge (\neg\phi[S \mapsto 0] \vee \bigwedge_{x \in S} \neg x)$ // refine

Algorithm 1. Refining

Example 3. Let $\phi = \neg x \vee \neg y \vee \neg z$ and $\psi = (\neg x \vee \neg y) \wedge (\neg x \vee \neg z) \wedge (\neg z \vee \neg y)$ Let us show that $\phi \models_{\min} \psi$. Let us choose the abstraction defined by the set $W = \{\{x\}, \{y\}\}$. The following diagram demonstrates that each model violating ψ has a witness corresponding to one of the sets in W.



Each triple represents a variable assignment where the elements represent the values of x, y, and z, respectively. Models and their pertaining witnesses are connected by an edge, which is labeled by the set of variables S whose values are flipped to 0 *to obtain the witness.*

The approach of searching for the right abstraction follows the Counter-Example Guided Abstract Refinement (CEGAR) loop [3]. If the abstraction is a tautology, the search terminates. If the abstraction is not a tautology, it is weakened by adding some set of variables S to the set W. This weakening is referred to as *refinement* and is done by investigating the counterexample that shows that the current abstraction is not a tautology. If it cannot be refined, (7) is not a tautology and $\phi \models_{\min} \psi$ does not hold.

Algorithm 1 realizes the idea outlined above. The algorithm maintains the negation of the abstraction in variable ω and starts with W being the empty set. Therefore the initial abstraction is $(\phi \wedge \neg\psi) \Rightarrow 0$ with the negation being $\phi \wedge \neg\psi$ (line 1). The test whether the abstraction is a tautology or not is done by calling a SAT solver on its negation (line 3). If the negation is unsatisfiable—the abstraction is a tautology—then the algorithm terminates and returns true (line 4). If a model ν is found showing that the abstraction is not a tautology, it means that for any assignment that is obtained from ν by flipping some set of variables in $S \in W$ to 0 is not a model of ϕ. The algorithm looks for a model ν' that is strictly smaller than ν applying Proposition 1 (line 6). If there is no model strictly smaller than ν then the algorithm terminates and returns false since ν is a minimal model and violates ψ (line 7). If there is a model ν' that is strictly smaller than ν, there is some set of

variables that are 1 in ν but are 0 in ν'. This set of variables is added to the sets determining the abstraction (line 10). Observe that a set S will be used at most once to refine the abstraction since once the set is added to W, an assignment for which flipping 1 to 0 for variables in S yields a model cannot satisfy the negation of the abstraction. Consequently, the algorithm is terminating and will perform at most as many iterations as there are subsets of the set V.

5 Computing Free-For-Negation

This section specializes Algorithm 1 to compute variables free for negation—variables that take value 0 in all minimal models. As mentioned earlier, this problem is a special case of the problem Entails-Min, studied in the previous section: x is free for negation in ϕ iff $\phi \models_{\min} \neg x$. However, focusing on this type of formulas enables a more efficient implementation of the algorithm.

The abstractions used in the previous section have to contain the condition that at least one of the variables being flipped to 0 is 1 to guarantee the corresponding witnesses is strictly smaller (see (7)). For variables free for negation these conditions will not be needed thanks to the following proposition.

Proposition 6. *Let ν be a model of ϕ s.t. $\nu(x) = 1$ for a variable x. If x is free for negation, then there exists a model ν' of ϕ s.t. $\nu' < \nu$ and $\nu'(x) = 0$.*

Proposition 6 tells us that if $\nu(x) = 1$ and x is free for negation, there must be a witness ν' that flips x to 0 (and possibly some other variables). This ensures that ν and ν' are different. This observation enables us to compute $\phi \models_{\min} \neg x$ by determining the validity of a stronger and more concise formula than before.

Proposition 7. *A variable x is free for negation in ϕ iff the following formula is a tautology.*

$$(\phi \wedge x) \Rightarrow \bigvee_{S \subseteq V \wedge x \in S} \phi[S \mapsto 0] \tag{9}$$

The abstraction of (9) is analogous to the one used in the previous section with the difference that only sets of variables containing x are considered. Hence, the abstraction always has the following form.

$$(\phi \wedge x) \Rightarrow \bigvee_{S \in W} \phi[S \mapsto 0], \text{ where } W \subseteq \mathcal{P}(V) \text{ and } (\forall S \in W)(x \in S) \tag{10}$$

5.1 Constructing and Refining Abstraction

Whenever the abstraction is being refined (weakened) the size of the formula representing the negation of the abstraction increases. Since the abstraction is refined in the worst case exponentially many times, it is warranted to pay attention to the size of the formula representing the negation of the abstraction.

The negation of an abstraction is a conjunct of the left-hand side of the implication and formulas capturing the substitutions.

input : CNF formula ϕ and a variable x
output: true iff $\phi \models_{\min} \neg x$

1 $\phi_0 \leftarrow \phi[x \mapsto 0]$
2 $\phi_0' \leftarrow \{\neg r_c \vee c \mid c \in \phi_0\} \cup \{\neg l \vee r_c \mid c \in \phi_0, l \in c\} \cup \left\{\bigvee_{c \in \phi_0} \neg r_c\right\}$
3 $\omega \leftarrow \phi \wedge x \wedge \phi_0'$
4 **while** true **do**
5 $\quad (\mathsf{outc}_1, \nu) \leftarrow \mathtt{SAT}(\omega)$
6 $\quad$ **if** $\mathsf{outc}_1 =$ false **then**
7 $\quad\quad$ **return** true // no counterexample was found
8 $\quad (\mathsf{outc}_2, \nu') \leftarrow \mathtt{SAT}\left(\phi \wedge \neg x \wedge \bigwedge_{\nu(z)=0} \neg z\right)$ // find $\nu' < \nu$ and $\nu'(x) = 0$
9 $\quad$ **if** $\mathsf{outc}_2 =$ false **then**
10 $\quad\quad$ **return** false // abstraction cannot be refined
11 $\quad S \leftarrow \{z \in V \mid \nu(z) = 1 \wedge \nu'(z) = 0\}$
12 $\quad C_p \leftarrow \{c \in \phi_0 \mid (c \cap S) \neq \emptyset\}$ // clauses with some $y \in S$
13 $\quad C_n \leftarrow \{c \in \phi_0 \mid (c \cap \neg S) \neq \emptyset\}$ // clauses with some $\neg y \in S$
14 $\quad C \leftarrow \{c' \mid c \in (C_p \setminus C_n) \wedge c' = c[S \mapsto 0]\}$ // new clauses
15 $\quad \omega \leftarrow \omega \cup \{\neg r_c \vee c \mid c \in C\} \cup \{\neg l \vee r_c \mid c \in C, l \in c\}$ // representation
16 $\quad \omega \leftarrow \omega \cup \left\{\bigvee_{c \in \phi \setminus (C_n \cup C_p)} \neg r_c \vee \bigvee_{c \in C} \neg r_c\right\}$ // negation of clauses

Algorithm 2. Deciding whether a variable is free for negation

$$(\phi \wedge x) \wedge \bigwedge_{S \in W} \neg\phi[S \mapsto 0], \text{ where } W \subseteq \mathcal{P}(V) \text{ and } (\forall S \in W)(x \in S) \quad (11)$$

When the abstraction is being refined, a new set of variables S is added to the set W, therefore, the negation of the abstraction is strengthened by conjoining it with $\neg\phi[S \mapsto 0]$. We aim to implement this strengthening without duplicating those parts of the formula that are already present.

Algorithm 2 outlines this procedure. Since all the sets S must contain x, the algorithm starts with the abstraction determined by $W = \{\{x\}\}$. In the initialization phase, the negation of this abstraction is $\phi \wedge x \wedge \neg\phi[x \mapsto 0]$ and is computed using the Tseitin transformation [24]. Each clause c in $\phi[x \mapsto 0]$ is represented by a fresh variable r_c and a clause is added that expresses that at least one of these variables must be 0 (line 2). As in the previous section, variable ω represents the negation of the abstraction (see (11)).

When the abstraction is being refined, the formula in variable ω is conjoined with $\neg\phi[S \mapsto 0]$. Since ω already contains clauses from $\neg\phi[x \mapsto 0]$, we need to consider only those clauses that contain literals on the variables in S. Clauses containing negative literals on variables from S are skipped, positive literals are removed. Each of the affected clauses is represented by a fresh Tseitin variable. Finally, a clause is added to express that one of the clauses in $\phi[S \mapsto 0]$ is 0. Note that this clause is referring to the original Tseitin variables for the clauses that are not affected by the substitution besides the freshly created ones. Note that when looking for a model $\nu' < \nu$, the algorithm requires that x has value 0 in ν' since the set S must contain x (line 8).

5.2 Finding Models

An abstraction is refined according to two responses from the underlying SAT solver (ν and ν'). This enables us to devise heuristics that prefer some responses of the solver to another. The motivation for these heuristics is to find abstractions where the set W determining the abstraction contains few sets S. Dually, this means that each of $S \in W$ yields a witness for many models. The heuristics used in the current implementation are motivated by the two following examples.

Example 4. Let $\phi = (x \Rightarrow y) \wedge (w \vee z)$. The abstraction defined by $W = \{\{x, y\}\}$ shows that $\phi \models_{\min} \neg y$ since flipping both x and y in any model yields a model (a witness). The abstraction determined by $W = \{\{x, y, z\}\}$ is not sufficient. This abstraction provides a witness for models with w having value 1 but not for the others. Intuitively, variable z is irrelevant to the relation $x \Rightarrow y$ and therefore it is better to choose a small S.

Example 5. Let $\phi = x \Rightarrow (y \vee w_1 \vee \ldots w_n)$ and let us prove that $\phi \models_{\min} \neg y$. The abstraction determined by $W = \{\{x, y\}\}$ is sufficient. However, if ν is not minimal, it may be that $\nu = x^1, y^1, w_1^1, \ldots w_n^1$ which gives us an exponential number of possibilities for ν' while only one of them is desirable. Intuitively, if ν is not minimal and there is some set S that yields a witness for both ν and some $\nu_1 < \nu$, then the set S is more likely to be found when ν_1 is inspected.

Based on this last observation, the model ν is required to be minimal. To make the difference between ν and ν' small, and therefore make this set S small, the solution ν' is required to be a maximal model.

To obtain a minimal, respectively maximal, model from a SAT solver is done by specifying the *phase*—the value that the solver prefers when making decisions when traversing the search space. Namely, preferring 0 yields a minimal model while preferring 1 yields a maximal model [10,22].

6 Computing Free-For-Negation-All

To calculate the set of variables that are free for negation, we invoke the algorithm described in the previous section for each variable. This procedure is optimized by conjoining the negations of the variables that have already been shown to be free for negation, which is justified by the following proposition.

Proposition 8. *Let ϕ and ψ be formulas such that $\phi \models_{\min} \psi$. The formula $\phi \wedge \psi$ has the same set of minimal models as ϕ. In particular, if $\phi \models_{\min} \neg x$ then $(\phi \wedge \neg x) \models_{\min} \neg y$ iff $\phi \models_{\min} \neg y$.*

The motivation for conjoining negations of variables free for negation is to give more information to subsequent inferences. The effectiveness of this technique, however, depends on the ordering of the variables. Hence, the approach we use is to set timeouts for testing a single variable and if a test times out, the variable is tested again but with information gained from the other tests.

```
input  : CNF formula φ and a set of variables V
output: subset of V that are free for negation
1  F ← ∅
2  X ← V
3  timeout ← initial-timeout
4  while X ≠ ∅ do
5      G ← ∅
6      foreach x in X do
7          (success, outc) ← Free-For-Negation(φ, x, timeout)
8          if success = true then
9              G ← G ∪ {x}
10             if outc = true then
11                 F = F ∪ {x}
12                 φ = φ ∧ ¬x
13     X ← X ∖ G
14     timeout ← k × timeout
15     return F
```

Algorithm 3. Computing the set of variables that are free for negation

Algorithm 3 summarizes these ideas in pseudocode. The algorithm described in the previous section is represented by the function Free-For-Negation, which returns a pair of values. The first value in the pair indicates whether the algorithm terminated before the given timeout or not. The second value of the pair indicates whether the given variable is free for negation or not. The timeout is gradually multiplied by some constant coefficient k. In the actual implementation there is a maximum timeout for which the algorithm stops and returns an approximation of the set of variables free for negation.

7 Evaluation

Algorithm 3 was implemented in Java using SAT4J as the underlying SAT solver while availing of its incremental interface [23]. The implementation was evaluated on a benchmark of 260 tests[1]. A majority of these are valid software configurations (motivated by [13]). A few tests are from the SAT '09 competition—relatively easy instances were chosen as the computed problem is significantly harder than satisfiability. The results appear in Table 1. An instance is considered solved if the answer is given in less than 30 s. The time given in the table is the average for the solved instances.

The alternative we tried was based on the tool `circ2dlp` [20], which transforms circumscription into a disjunctive logic program, and `gnt` [11], which lists all models of that program. From the list of models it is easy and fast to construct the set of variables that are free for negation. We also tried using a QBF solver along with (3), but that implementation solved *none* of the 260 tests.

[1] Available at `http://logos.ucd.ie/confs/jelia10/jelia10-bench.tgz`

Table 1. Experimental evaluation

	tests	Algorithm 3 solved	Algorithm 3 time[s]	circ2dlp+gnt solved	circ2dlp+gnt time[s]
e-shop	174	174	2.1	95	2.4
BerkeleyDB	30	30	0.9	30	< 0.1
model-transf	41	41	1.1	35	2.8
SAT2009	15	3	7.6	2	2.5

8 Summary and Future Work

This paper proposes an algorithm for deduction under the set of minimal models of a propositional formula. This algorithm enables us to reason under the propositional versions of close world assumption or circumscription. The algorithm hinges on an application of a SAT solver but more importantly on counterexample guided abstraction refinement (CEGAR). While CEGAR has been amply used in software verification [3,8],we are not aware of its application in nonmonotonic reasoning.

The deduction problem under the set of minimal models can be formulated as QBF [5] or as a DLP [15,12]. The experimental results suggest that current QBF solvers are not practical for this problem. The comparison to the DLP-based solution indicates that our dedicated algorithm enables solving more instances. Nevertheless, the DLP-based solution was faster for some instances.

The promising experimental results indicate that the ideas behind the presented algorithms have potential for further work. The evaluation was performed for the computation of variables free for negation defining the closure of a theory in GCWA, hence, further evaluations should be performed on other types of problems in this domain. On a more general scale, it is well known that minimal models can be seen as optima with respect to the pertaining ordering [2,22]. This opens possibilities to investigate generalizations of the presented algorithms for different orderings than the one used for minimal models. Last but not least, the comparison with the DLP-based solution indicates that it would be beneficial to investigate approaches tackling the problem with hybrid techniques.

References

1. Cadoli, M., Lenzerini, M.: The complexity of closed world reasoning and circumscription. In: AAAI Conference on Artificial Intelligence, pp. 550–555 (1990)
2. Castell, T., Cayrol, C., Cayrol, M., Berre, D.L.: Using the Davis and Putnam procedure for an efficient computation of preferred models. In: European Conference on Artificial Intelligence, pp. 350–354 (1996)
3. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, ch. 1855, pp. 154–169. Springer, Heidelberg (2000)
4. Dix, J., Furbach, U., Niemelä, I.: Nonmonotonic reasoning: Towards efficient calculi and implementations. In: Voronkov, A., Robinson, A. (eds.) Handbook of Automated Reasoning, vol. 19, pp. 1241–1354. North-Holland, Amsterdam (2001)

5. Egly, U., Eiter, T., Tompits, H., Woltran, S.: Solving advanced reasoning tasks using quantified boolean formulas. In: AAAI Conference on Artificial Intelligence, pp. 417–422 (2000)
6. Eiter, T., Gottlob, G.: Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. Theor. Comput. Sci. 114(2), 231–245 (1993)
7. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. Artif. Intell. 172(12-13), 1495–1539 (2008)
8. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: Principles of programming languages (POPL), pp. 191–202. ACM, New York (2002)
9. Gelfond, M., Przymusinska, H., Przymusinski, T.C.: On the relationship between circumscription and negation as failure. Artif. Intell. 38(1), 75–94 (1989)
10. Giunchiglia, E., Maratea, M.: Solving optimization problems with DLL. In: European Conference on Artificial Intelligence, pp. 377–381 (2006)
11. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding partiality and disjunctions in stable model semantics. ACM Trans. Comput. Log. 7(1), 1–37 (2006)
12. Janhunen, T., Oikarinen, E.: Capturing parallel circumscription with disjunctive logic programs. In: European Conf. on Logics in Artif. Intell., pp. 134–146 (2004)
13. Janota, M., Botterweck, G., Grigore, R., Marques-Silva, J.: How to complete an interactive configuration process? In: Conference on Current Trends in Theory and Practice of Computer Science, pp. 528–539 (2010)
14. Janota, M., Grigore, R., Marques-Silva, J.: Counterexample guided abstraction refinement algorithm for propositional circumscription. Tech. Rep. TR-32-2010, INESC-ID Lisboa (2010)
15. Lifschitz, V.: Foundations of logic programming. In: Principles of Knowledge Representation, pp. 69–127 (1996)
16. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artif. Intell. 13(1-2), 27–39 (1980)
17. McCarthy, J.: Applications of circumscription to formalizing common-sense knowledge. Artif. Intell. 28(1), 89–116 (1986)
18. Minker, J.: On indefinite databases and the closed world assumption. In: Loveland, D.W. (ed.) CADE 1982. LNCS, vol. 138, pp. 292–308. Springer, Heidelberg (1982)
19. Niemelä, I.: Implementing circumscription using a tableau method. In: European Conference on Artificial Intelligence, pp. 80–84 (1996)
20. Oikarinen, E., Janhunen, T.: circ2dlp — translating circumscription into disjunctive logic programming. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 405–409. Springer, Heidelberg (2005)
21. Przymusinski, T.C.: An algorithm to compute circumscription. Artif. Intell. 38(1), 49–73 (1989)
22. Rosa, E.D., Giunchiglia, E., Maratea, M.: Solving satisfiability problems with preferences. Constraints. An International Journal (2010) (in press)
23. SAT4j, http://www.sat4j.org
24. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic 2(115-125), 10–13 (1968)
25. Yahya, A.H., Henschen, L.J.: Deduction in non-Horn databases. Journal of Automated Reasoning 1(2), 141–160 (1985)

$\mathcal{ALC}_{\mathcal{ALC}}$: A Context Description Logic

Szymon Klarman[1] and Víctor Gutiérrez-Basulto[2]

[1] Department of Computer Science, Vrije Universiteit Amsterdam
sklarman@few.vu.nl
[2] Department of Computer Science, Universität Bremen
victor@informatik.uni-bremen.de

Abstract. We develop a novel description logic (DL) for representing and reasoning with contextual knowledge. Our approach descends from McCarthy's tradition of treating contexts as formal objects over which one can quantify and express first-order properties. As a foundation we consider several common product-like combinations of DLs with multi-modal logics and adopt the prominent $(\mathbf{K}_n)_{\mathcal{ALC}}$. We then extend it with a second sort of vocabulary for describing contexts, i.e., objects of the second dimension. In this way, we obtain a *two-sorted, two-dimensional combination of a pair of DLs* $\mathcal{ALC}$, called $\mathcal{ALC}_{\mathcal{ALC}}$. As our main technical result, we show that the satisfiability problem in this logic, as well as in its proper fragment $(\mathbf{K}_n)_{\mathcal{ALC}}$ with global TBoxes and local roles, is 2ExpTime-complete. Hence, the surprising conclusion is that the significant increase in the expressiveness of $\mathcal{ALC}_{\mathcal{ALC}}$ due to adding the vocabulary comes for no substantial price in terms of its worst-case complexity.

1 Introduction

Over two decades ago John McCarthy introduced the AI community to a new paradigm of formalizing contexts in logic-based knowledge systems. This idea, presented in his Turing Award Lecture [1], was quickly picked up by others and by now has led to a significant body of work studying different implementations of the approach in a variety of formal frameworks and applications [2,3,4,5,6,7,8]. The great appeal of McCarthy's paradigm stems from the simplicity and intuitiveness of the three major postulates it is based on:

1. Contexts are formal objects. More precisely, a *context* is anything that can be denoted by a first-order term and used meaningfully in a statement of the form $ist(c, p)$, saying that proposition p *is true* in context c [1,5,6,2], e.g., $ist(Hamlet,$ 'Hamlet is a prince.'). By adopting a strictly formal view on contexts, one can bypass unproductive debates on what they really are and instead take them as primitives underlying practical models of contextual reasoning.

2. Contexts are organized in relational structures. In the commonsense reasoning, contextual assumptions are dynamically and directionally altered [8,2]. Contexts are entered and then exited, accessed from other contexts or transcended to broader ones. Formally, we want to allow nestings of the form $ist(c, ist(c', p))$, e.g., $ist(France, ist(capital,$ 'The city river is Seine.')).

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 208–220, 2010.
© Springer-Verlag Berlin Heidelberg 2010

3. Contexts have properties and can be described. As first-order objects, contexts can be in a natural way described in a first-order language [4,6]. This allows for addressing them generically through quantified formulas such as $\forall x(P(x) \rightarrow ist(x, p))$, expressing that p is true in every context of type P, e.g., $\forall x(barbershop(x) \rightarrow ist(x,$ 'Main service is a haircut.')).

The goal of this work is to import McCarthy's paradigm into the framework of Description Logics (DLs), a popular family of knowledge representation formalisms, with many successful applications [9]. Although the importance of contexts in DLs has been generally acknowledged, the framework is still not supported with a dedicated, generic theory of accommodating contextual knowledge. The most common perspectives considered in this area are limited to: 1) integration of local ontologies [10,11], 2) modeling levels of abstraction as subsets of DL models [12,13], and 3) capturing dynamics of knowledge across a fixed modal dimension, most typically a temporal one [14,15,16].

The DL $\mathcal{ALC}_{\mathcal{ALC}}$, which we develop here, is a novel formalism for representing and reasoning with context-dependent knowledge. On the one hand, we systematically incorporate the three postulates of McCarthy, and thus, ground our proposal in a longstanding tradition of formalizing contexts in AI. On the other, we build on top of two-dimensional DLs [17], which provide $\mathcal{ALC}_{\mathcal{ALC}}$ with well-understood formal foundations. In this paper we present a thorough study of the formal properties of $\mathcal{ALC}_{\mathcal{ALC}}$, including its expressiveness, computational complexity and relationships to other formalisms. As our main technical result, we show that the satisfiability problem in $\mathcal{ALC}_{\mathcal{ALC}}$, as well as in its proper fragment $(\mathbf{K}_n)_{\mathcal{ALC}}$ with global TBoxes and local roles, is 2ExpTime-complete. This reveals that the jump in the complexity from ExpTime is essentially caused by the interaction of multiple $\mathbf{K}$-modalities with global TBoxes.

2 Overview

We start with an outline of the milestones for constructing and studying the logic $\mathcal{ALC}_{\mathcal{ALC}}$. Then, we recap the basic notions concerning the DL $\mathcal{ALC}$.

2.1 Roadmap

We introduce $\mathcal{ALC}_{\mathcal{ALC}}$ in a gradual way. First, in Section 3, we elaborate on some well-studied combinations of the DL $\mathcal{ALC}$ with modal logics, known as two-dimensional or modal DLs [18,17,19]. From our perspective, the two-dimensional semantics of such logics is very well suited for representing context objects and the relational structures they form. After some conceptual and computational evaluation we then adopt $(\mathbf{K}_n)_{\mathcal{ALC}}$ as the foundation for our context DL. Finally, we show that the migration from $\mathcal{ALC}$ to $(\mathbf{K}_n)_{\mathcal{ALC}}$ with global TBoxes and local roles rises the complexity from ExpTime to 2ExpTime.

Next, in Section 4, we extend $(\mathbf{K}_n)_{\mathcal{ALC}}$ with a second sort of vocabulary, which serves for describing contexts. Formally, we can see this extension as a shift from $(\mathbf{K}_n)_{\mathcal{ALC}}$ to $\mathcal{ALC}_{\mathcal{ALC}}$, i.e., a *two-sorted*, *two-dimensional* combination

of a pair of DLs $\mathcal{ALC}$. Each sort in $\mathcal{ALC}_{\mathcal{ALC}}$ applies to its corresponding dimension and the two are allowed to interact in a controlled manner. Since such an extension is relatively uncommon, we then relate $\mathcal{ALC}_{\mathcal{ALC}}$ to the standard framework of products of modal logics and show that the departure is not radical. More interestingly, we also prove that the extension, although offering a lot of expressive flexibility, is not to be paid for in yet another increase of the worst-case complexity. Satisfiability in $\mathcal{ALC}_{\mathcal{ALC}}$ remains 2ExpTime-complete.

In Section 5, we present an example application of $\mathcal{ALC}_{\mathcal{ALC}}$. Finally, in Section 6, we conclude the paper and point to directions for future research.

2.2 Preliminaries: DL $\mathcal{ALC}$

A DL language is specified by a vocabulary $\Sigma = (N_I, N_C, N_R)$, where N_I is a set of *individual names*, N_C a set of *concept names*, N_R a set of *role names*, and a number of operators for constructing complex concept descriptions [9]. The $\mathcal{ALC}$ concept language L over Σ is the smallest set of concepts containing $\top$, all concept names from N_C and closed under the constructors:

$$\neg C \;\mid\; C \sqcap D \;\mid\; \exists r.C$$

where $C, D \in L$ and $r \in N_R$. Conventionally, we abbreviate $\neg\top$ with $\bot$, $\neg(\neg C \sqcap \neg D)$ with $C \sqcup D$ and $\neg\exists r.\neg C$ with $\forall r.C$. The semantics of L is given through *interpretations* of the form $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$, where Δ is a non-empty *domain* of individuals, and $\cdot^{\mathcal{I}}$ is an *interpretation function*. The meaning of the vocabulary is fixed via mappings: $a^{\mathcal{I}} \in \Delta$ for every $a \in N_I$, $A^{\mathcal{I}} \subseteq \Delta$ for every $A \in N_C$ and $r^{\mathcal{I}} \subseteq \Delta \times \Delta$ for every $r \in N_R$, and $\top^{\mathcal{I}} = \Delta$. Then the function is inductively extended over L according to the fixed semantics of the constructors:

$$\begin{aligned}(\neg C)^{\mathcal{I}} &= \{x \in \Delta \mid x \notin C^{\mathcal{I}}\},\\ (C \sqcap D)^{\mathcal{I}} &= \{x \in \Delta \mid x \in C^{\mathcal{I}} \cap D^{\mathcal{I}}\},\\ (\exists r.C)^{\mathcal{I}} &= \{x \in \Delta \mid \exists y:\ \langle x, y\rangle \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}.\end{aligned}$$

A *knowledge base* (or an *ontology*) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. The TBox contains general concept inclusion axioms (GCIs) $C \sqsubseteq D$, for arbitrary concepts $C, D \in L$. We write $C \equiv D$ whenever both $C \sqsubseteq D$ and $D \sqsubseteq C$ are in $\mathcal{T}$. The ABox consists of concept assertions $C(a)$ and role assertions $r(a, b)$, where $a, b \in N_I$, $C \in L$ and $r \in N_R$. An interpretation $\mathcal{I}$ *satisfies* an axiom in either of the following cases:

- $\mathcal{I} \models C \sqsubseteq D$ *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $\mathcal{I} \models C(a)$ *iff* $a^{\mathcal{I}} \in C^{\mathcal{I}}$,
- $\mathcal{I} \models r(a, b)$ *iff* $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in r^{\mathcal{I}}$.

Finally, $\mathcal{I}$ is a *model* of a DL knowledge base whenever it satisfies all its axioms.

3 Adding Context Structures: From $\mathcal{ALC}$ to $(\mathbf{K}_n)_{\mathcal{ALC}}$

In order to introduce *context structures* into the DL semantics, and thus account for the first two postulates of McCarthy, we move from $\mathcal{ALC}$ to its two-dimensional, multi-modal extensions.

3.1 Syntax and Semantics

A two-dimensional, multi-modal concept language $L_{\mathcal{ALC}}$ over vocabulary Σ is the smallest set of concepts containing $\top$, concept names from N_C and closed under the $\mathcal{ALC}$ and the two new constructors:

$$\Diamond_i C \;\mid\; \Box_i C$$

where $C \in L_{\mathcal{ALC}}$ and $1 \leq i \leq n$ for some fixed $n \in \mathbb{N}$. It is assumed that $\Box_i$ abbreviates $\neg\Diamond_i\neg$. In our framework, every i is interpreted as a distinguished *contextualization operation*. The modal *context operators* associated with i enable a transition to the state of affairs holding in some ($\Diamond_i$) or all ($\Box_i$) contexts accessible from the current one through i. An interpretation of $L_{\mathcal{ALC}}$ is defined as a tuple $\mathfrak{M} = (\mathfrak{C}, \{R_i\}_{1\leq i\leq n}, \Delta, \{\cdot^{\mathcal{I}(c)}\}_{c\in\mathfrak{C}})$, where:

- $\mathfrak{C}$ is a non-empty *context domain*,
- $R_i \subseteq \mathfrak{C} \times \mathfrak{C}$ is an *accessibility relation* on $\mathfrak{C}$, associated with $\Diamond_i$ and $\Box_i$,
- Δ is a non-empty *object domain*,
- $\cdot^{\mathcal{I}(c)}$ is an *interpretation function* in context c.

For every $c \in \mathfrak{C}$, the interpretation function $\mathcal{I}(c)$ fixes the meaning of the language by extending the basic $\mathcal{ALC}$ interpretation rules with the additional:

$$(\Diamond_i C)^{\mathcal{I}(c)} = \{x \in \Delta \mid \exists d \in \mathfrak{C} :\ cR_i d \wedge x \in C^{\mathcal{I}(d)}\},$$
$$(\Box_i C)^{\mathcal{I}(c)} = \{x \in \Delta \mid \forall d \in \mathfrak{C} :\ cR_i d \rightarrow x \in C^{\mathcal{I}(d)}\}.$$

In what follows, we loosely refer to $\mathfrak{C}$ as the *context dimension* and to Δ as the *object dimension* of the combination (see example in Fig. 1). Generally, the semantic setup for multi-dimensional DLs allows several degrees of freedom regarding rigidity of names and domain assumptions [17]. Here, we pose the natural, rigid interpretation of individual names, i.e., $a^{\mathcal{I}(c)} = a^{\mathcal{I}(d)}$ for every $c, d \in \mathfrak{C}$, and local (non-rigid) interpretation of concepts. The interpretation of roles is discussed in the next paragraphs. We also assume that all contexts share the same object domain. Even if not suiting all applications, the constant domain assumption is known to be most universal, in the sense that the expanding/varying case can be always reduced to the constant one.

For a fixed language $L_{\mathcal{ALC}}$ the knowledge about the object dimension, now relative to contexts, can be expressed by means of usual axioms. In particular, a TBox $\mathcal{T}$ is a set of GCIs over concepts from $L_{\mathcal{ALC}}$. In this section it suffices to consider only the basic problem of *concept satisfiability* with respect to a global $\mathcal{T}$. The satisfaction relation for GCIs is defined with respect to an interpretation $\mathfrak{M}$ and a context $c \in \mathfrak{C}$:

- $(\mathfrak{M}, c) \models C \sqsubseteq D$ *iff* $C^{\mathcal{I}(c)} \subseteq D^{\mathcal{I}(c)}$.

We call $\mathfrak{M}$ a model of a global $\mathcal{T}$ whenever it satisfies all axioms in $\mathcal{T}$ in every $c \in \mathfrak{C}$. A concept C is satisfiable w.r.t. $\mathcal{T}$ *iff* there exists a model of $\mathcal{T}$ such that for some $c \in \mathfrak{C}$ and $d \in \Delta$ it is the case that $d \in C^{\mathcal{I}(c)}$.

It is not hard to see that without further constraints the resulting logic corresponds to the well-known product of multi-modal $\mathbf{K}_n$ with $\mathcal{ALC}$, denoted shortly

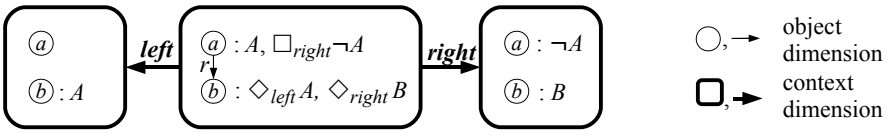


Fig. 1. A context structure modeling concept $A \sqcap \Box_{right} \neg A \sqcap \exists r.(\Diamond_{left} A \sqcap \Diamond_{right} B)$.

as $(\mathbf{K}_n)_{\mathcal{ALC}}$ [18,20,17,19]. As for many other applications, also in the case of context DLs $(\mathbf{K}_n)_{\mathcal{ALC}}$ seems to provide the most natural and flexible foundation. Obviously, it is not difficult to further constrain accessibility relations in order to obtain context structures with more specific properties. Leaving a broader study of this subject for future research, let us just consider two such restrictions, sometimes evoked in the literature on contexts:

(quasi-functionality) $\forall c, d, e \in \mathfrak{C}\ (cRd \wedge cRe \rightarrow d = e)$,
(seriality) $\forall c \in \mathfrak{C}\ \exists d \in \mathfrak{C}\ (cRd)$.

Buvač's propositional logic of contexts [2,3] is a notational variant of $\mathbf{K}_n$, with $\Box_i \varphi$ written as $ist(i, \varphi)$. In Buvač's setting $\Box_i$ quantifies over possible interpretations of the context i. In our framework, where contexts are not modality indices but first-order objects, $\Box_i$ would quantify over possible contexts instead, which clearly distorts the intended behavior of *ist*. To avoid this, one might rather use $\Box_i$ of the logic $\mathbf{Alt}_n$, characterized by all quasi-functional Kripke frames [19]. In $\mathbf{Alt}_n$ there is at most one context accessible through each contextualization operation. Thus, $\Diamond_i \varphi \wedge \Diamond_i \psi$ semantically implies $ist(c, \varphi \wedge \psi)$ for some unique c. Nossum [8] pursues similar intuitions and advocates even stronger $\mathbf{DAlt}_n$, which is Kripke-complete w.r.t. all quasi-functional and serial frames. Such a semantics ensures that it is always possible to reach exactly one context through each accessibility relation. Since formally the two frame properties boil down to the functionality condition, it follows that the two operators $\Diamond_i, \Box_i$ collapse into a single $\bigcirc_i$. Finally $\mathbf{D}_n$, characterized by all serial frames, is used by Buvač [2,3] for verifying consistency of contextual knowledge. Since the seriality condition enforces existence of all potential contexts, the knowledge attributed to these contexts cannot be self-contradictory.

3.2 Complexity

As it turns out, the choice between any of the characterizations discussed above is quite irrelevant from the computational perspective. In most cases the complexity results apply to all logics $L_{\mathcal{ALC}}$, for $L \in \{\mathbf{DAlt}_n, \mathbf{D}_n, \mathbf{Alt}_n, \mathbf{K}_n\}$. To ease the transfer of some of the observations we make below, we use the following reductions:

Proposition 1. *Concept satisfiability w.r.t. global TBoxes is polynomially reducible between the following logics* (*where* $\mapsto$ *means* reduces to)*:*

$$(\mathbf{DAlt}_n)_{\mathcal{ALC}} \mapsto \{(\mathbf{D}_n)_{\mathcal{ALC}}, (\mathbf{Alt}_n)_{\mathcal{ALC}}\} \mapsto (\mathbf{K}_n)_{\mathcal{ALC}}.$$

To see that the reductions hold indeed, it is enough to notice that if $(C, \mathcal{T})$ is a problem of deciding whether a concept C is satisfiable w.r.t. a global TBox $\mathcal{T}$, then by simple transformations of C and $\mathcal{T}$ one can enforce only models that are bisimilar to those characterizing the respective frame conditions:

(quasi-functionality) W.l.o.g. assume that $C = \mathrm{NNF}(C)$, where NNF stands for Negation Normal Form, and $\mathcal{T} = \{\top \sqsubseteq C_{\mathcal{T}}\}$, for some $C_{\mathcal{T}} = \mathrm{NNF}(C_{\mathcal{T}})$. Let C' and $C'_{\mathcal{T}}$ be the result of replacing every subconcept $\Diamond_i B$ occurring in C and $C_{\mathcal{T}}$, respectively, with $(\Diamond_i \top) \sqcap (\Box_i B)$. Then, $(C, \mathcal{T})$ is satisfiable on a quasi-functional frame *iff* $(C', \{\top \sqsubseteq C'_{\mathcal{T}}\})$ is satisfiable.

(seriality) Let $\mathcal{T}' = \mathcal{T} \cup \{\top \sqsubseteq \Diamond_i \top \mid 1 \leq i \leq n\}$, where n is the number of all modalities occurring in $\mathcal{T}$ and C. Then, $(C, \mathcal{T})$ is satisfiable on a serial frame *iff* $(C, \mathcal{T}')$ is satisfiable.

Our first result is a negative one. It closes the option of using rigid roles, i.e., such that $r^{\mathcal{I}(c)} = r^{\mathcal{I}(d)}$ for every $c, d \in \mathfrak{C}$, or applying context operators to roles. Unfortunately, adding rigid roles leads to undecidability already for the strongest of the logics with just a single context operator.

Theorem 1. *Concept satisfiability in* $\mathbf{DAlt}_{\mathcal{ALC}}$ *w.r.t. global TBoxes and with a single rigid role is undecidable.*

The full proof, along the others from this paper, is included in the appendix of the accompanying technical report [21]. We notice that $\mathbf{DAlt}_{\mathcal{ALC}}$ corresponds to a fragment of $\mathrm{LTL}_{\mathcal{ALC}}$ with the *next-time* operator, which is enough to construct a usual encoding of the undecidable $\mathbb{N} \times \mathbb{N}$ *tiling problem* [14]. Together with Proposition 1, the theorem immediately entails the following:

Theorem 2. *For any* $L \in \{\mathbf{DAlt}_n, \mathbf{D}_n, \mathbf{Alt}_n, \mathbf{K}_n\}$, *concept satisfiability in* $L_{\mathcal{ALC}}$ *w.r.t. global TBoxes with a single rigid role is undecidable.*

This result reveals an obvious limitation to the formalism, but a limitation one has to live with, considering that combinations of rigid roles with global TBoxes are rarely decidable unless the expressive power of the modal or the DL component is significantly reduced [19,14]. In the rest of this paper, we almost exclusively address the case of local (non-rigid) roles. To show decidability and the upper bound of the concept satisfiability problem in this setup,[1] we devise a quasistate elimination algorithm for $(\mathbf{K}_n)_{\mathcal{ALC}}$, similar to [19, Theorem 6.61]. As usual, the idea is to abstract from the domains $\mathfrak{C}$ and Δ and consider only a finite, in fact double exponential, number of quasistates which represent possible contexts inhabited by a finite number of possible types of individuals. Then, we iteratively eliminate all those that do not satisfy necessary conditions.

[1] Mind that the NExpTime-completeness result for concept satisfiability in $\mathbf{K}_{\mathcal{ALC}}$ [19, Theorem 15.15] applies to $\mathcal{ALC}$ with a single pair of $\mathbf{K}$ operators, full booleans on modalized formulas and no global TBoxes.

Theorem 3. *Deciding concept satisfiability in* $(\mathbf{K}_n)_{\mathcal{ALC}}$ *w.r.t. global TBoxes and only with local roles is in* 2ExpTime.

One could hope that at least some of the considered logics could be less complex than that. However, as the next theorem shows, this is not the case.

Theorem 4. *Deciding concept satisfiability in* $(\mathbf{DAlt}_n)_{\mathcal{ALC}}$ *w.r.t. global TBoxes and only with local roles is* 2ExpTime*-hard.*

For the proof we use a reduction of the word problem for exponentially bounded Alternating Turing Machines, which is known to be 2ExpTime-hard [22]. The increase in the complexity by one exponential, as compared to $\mathcal{ALC}$ alone (for which the problem is ExpTime-complete [9]), is notable and quite surprising. It could be expected that without rigid roles the satisfiability problem can be straightforwardly reduced to satisfiability in fusion models. This in turn should yield ExpTime upper bound by means of the standard techniques. However, as the following example for $(\mathbf{K}_n)_{\mathcal{ALC}}$ demonstrates, this strategy fails.

$$(\dagger)\ \Diamond_i C \sqcap \exists r.\Box_i \bot \qquad (\ddagger)\ \exists succ_i.C \sqcap \exists r.\forall succ_i.\bot$$

Although (†) clearly does not have a model, its reduction (‡) to a fusion language, where context operators are translated to restrictions on fresh $\mathcal{ALC}$ roles, is satisfiable. The reason is that while in the former case the information about the structure of the **K**-frame is global for all individuals, in the latter it becomes local. The r-successor in (‡) is simply not 'aware' that it should actually have a $succ_i$-successor.[2] This effect, amplified by presence of multiple modalities and global TBoxes (which can enforce infinite **K**-trees), makes the reasoning harder.

The two complexity bounds from Theorem 3 and 4, together with the reductions established in Proposition 1, provide us with the completeness result.

Theorem 5. *For any* $L \in \{\mathbf{DAlt}_n, \mathbf{D}_n, \mathbf{Alt}_n, \mathbf{K}_n\}$*, deciding concept satisfiability in* $L_{\mathcal{ALC}}$ *w.r.t. global TBoxes and only with local roles is* 2ExpTime*-complete.*

The theorem is quite robust under changes of domain assumptions and holds already in the case of expanding/varying domains in $(\mathbf{Alt}_n)_{\mathcal{ALC}}$. The only exception applies to $(\mathbf{DAlt}_n)_{\mathcal{ALC}}$ and $(\mathbf{D}_n)_{\mathcal{ALC}}$ with expanding/varying domains, where reduction to $\mathcal{ALC}$ is still possible.

What follows from this analysis, is that by sacrificing the generality of $\mathbf{K}_n$-frames one does not immediately obtain a better computational behavior as long as multiple context operators are permitted. For this reason, we adopt $(\mathbf{K}_n)_{\mathcal{ALC}}$ as the baseline for $\mathcal{ALC}_{\mathcal{ALC}}$, leaving for now the option of restricting context structures as an open problem.

[2] Demonstrating the corresponding phenomenon in $(\mathbf{DAlt}_n)_{\mathcal{ALC}}$ is not that straightforward due to the seriality condition, as then the global information concerns only the existence of $succ_i$-predecessors. Thus, one needs role inverses in the fusion language to observe the loss of such information.

4 Describing Contexts: From $(\mathbf{K}_n)_{\mathcal{ALC}}$ to $\mathcal{ALC}_{\mathcal{ALC}}$

We are now ready to define the target logic $\mathcal{ALC}_{\mathcal{ALC}}$, which additionally to $(\mathbf{K}_n)_{\mathcal{ALC}}$ offers a second sort of vocabulary for directly describing contexts. This extension addresses the third postulate of McCarthy.

4.1 Syntax and Semantics

We start by introducing the context component of the language and then suitably revise the object component.

The *context language* L_C is an $\mathcal{ALC}$ concept language over vocabulary $\Gamma = (M_I, M_C, M_R)$, where M_I is a set of (*context*) *individual names*, M_C is a set of (*context*) *concept names*, and M_R is a set of (*context*) *role names*. For disambiguation, we use ***bold font*** when writing names from the context vocabulary and we denote the elements of L_C as *c-concepts*. The semantics is defined in the usual manner (as presented in Section 2.2), in terms of an interpretation function $\cdot^{\mathcal{J}}$ ranging over the context domain $\mathfrak{C}$. The *context knowledge base* $\mathcal{C}$ consists of TBox and ABox axioms over Γ and L_C, also with the usual satisfaction conditions. Thus, $\mathcal{C}$ is in fact a standard $\mathcal{ALC}$ ontology with standard models of the form $(\mathfrak{C}, \cdot^{\mathcal{J}})$.

The interpretations of the context language are incorporated in the full $\mathcal{ALC}_{\mathcal{ALC}}$ interpretations of the form $\mathfrak{M} = (\mathfrak{C}, \cdot^{\mathcal{J}}, \Delta, \{\cdot^{\mathcal{I}(c)}\}_{c \in \mathfrak{C}})$, where:

- $\mathfrak{C}$ is a non-empty *context domain*,
- $\cdot^{\mathcal{J}}$ is an *interpretation function* of the context language,
- Δ is a non-empty *object domain*,
- $\cdot^{\mathcal{I}(c)}$ is an *interpretation function* of the object language in c.

The divergence from the original $(\mathbf{K}_n)_{\mathcal{ALC}}$ interpretations is minor. Basically, the accessibility relations over $\mathfrak{C}$ become now redundant, as their function can be taken over by context roles. For every contextualization operation i we can assume an implicit correspondence $R_i = \boldsymbol{r}_i^{\mathcal{J}}$, for some $\boldsymbol{r}_i \in M_R$. Note that given the broadened take on the context dimension, we might be now less strict about the informal reading of some of the components of the framework. Arguably, not all context roles have to be necessarily seen as 'contextualization operations' and not all elements of $\mathfrak{C}$ as genuine 'contexts'. Sometimes they can be just entities needed for describing contexts. Nevertheless, we keep using the context-object nomenclature to avoid potential confusions.

Although one can already express rich knowledge about contexts, such knowledge remains 'invisible' from the object level. In order to render it more accessible, and so gain better control over the interaction between the dimensions, we need to suitably internalize context descriptions in the object language.

Let $\Sigma = (N_I, N_C, N_R)$ be the *object vocabulary* disjoint from Γ. The *object language* L_O over Σ and the context language L_C is the smallest set of concepts, called *o-concepts*, containing $\top$, concept names from N_C and closed under the $\mathcal{ALC}$ and the following two constructors:

$$\langle \boldsymbol{C} \rangle_{\boldsymbol{r}} D \quad | \quad [\boldsymbol{C}]_{\boldsymbol{r}} D$$

where $\boldsymbol{C} \in L_C$ and $\boldsymbol{r} \in M_R$. Again, $[\cdot]_{\boldsymbol{r}}$ abbreviates $\neg\langle\cdot\rangle_{\boldsymbol{r}}\neg$. Intuitively, $\langle \boldsymbol{C}\rangle_{\boldsymbol{r}} D$ denotes all objects which are D in *some* context which is $\boldsymbol{C}$ and is accessible through $\boldsymbol{r}$. Similarly, $[\boldsymbol{C}]_{\boldsymbol{r}} D$ denotes all objects which are D in *every* context which is $\boldsymbol{C}$ and is accessible through $\boldsymbol{r}$. Overall, the syntax of the object language diverges from the one of $(\mathbf{K}_n)_{\mathcal{ALC}}$ only in that the indices appearing by $\Diamond_i, \Box_i$ are now replaced with context roles, while both operators embrace a single c-concept, which additionally qualifies the accessed contexts. Consequently, the changes in the semantics affect only the contextualized concepts:

$$(\langle \boldsymbol{C}\rangle_{\boldsymbol{r}} D)^{\mathcal{I}(c)} = \{x \in \Delta \mid \exists d \in \mathfrak{C} : \ \langle c, d\rangle \in \boldsymbol{r}^{\mathcal{J}} \wedge d \in \boldsymbol{C}^{\mathcal{J}} \wedge x \in D^{\mathcal{I}(d)}\},$$
$$([\boldsymbol{C}]_{\boldsymbol{r}} D)^{\mathcal{I}(c)} = \{x \in \Delta \mid \forall d \in \mathfrak{C} : \ \langle c, d\rangle \in \boldsymbol{r}^{\mathcal{J}} \wedge d \in \boldsymbol{C}^{\mathcal{J}} \rightarrow x \in D^{\mathcal{I}(d)}\}.$$

To grant maximum flexibility in expressing the knowledge about the object dimension we first define the set of possible *object formulas*, i.e., formulas which can meaningfully hold in individual contexts:

$$B \sqsubseteq D \ \mid \ a : D \ \mid \ s(a, b) \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \psi \ \mid \ \langle \boldsymbol{C}\rangle_{\boldsymbol{r}} \varphi \ \mid \ [\boldsymbol{C}]_{\boldsymbol{r}} \varphi$$

where B, D are o-concepts, $a, b \in N_I$, $s \in N_R$, $\boldsymbol{C}$ is a c-concept and $\boldsymbol{r} \in M_R$. Object formulas are satisfied by $\mathfrak{M}$ in context $c \in \mathfrak{C}$ in the following cases:

- $(\mathfrak{M}, c) \models B \sqsubseteq D$ *iff* $B^{\mathcal{I}(c)} \subseteq D^{\mathcal{I}(c)}$,
- $(\mathfrak{M}, c) \models a : D$ *iff* $a^{\mathcal{I}(c)} \in D^{\mathcal{I}(c)}$,
- $(\mathfrak{M}, c) \models s(a, b)$ *iff* $\langle a^{\mathcal{I}(c)}, b^{\mathcal{I}(c)}\rangle \in s^{\mathcal{I}(c)}$,
- $(\mathfrak{M}, c) \models \neg\varphi$ *iff* $(\mathfrak{M}, c) \not\models \varphi$,
- $(\mathfrak{M}, c) \models \varphi \wedge \psi$ *iff* $(\mathfrak{M}, c) \models \varphi$ and $(\mathfrak{M}, c) \models \psi$,
- $(\mathfrak{M}, c) \models \langle \boldsymbol{C}\rangle_{\boldsymbol{r}} \varphi$ *iff* $(\mathfrak{M}, d) \models \varphi$ for some $d \in \mathfrak{C}$ s.t. $\langle c, d\rangle \in \boldsymbol{r}^{\mathcal{J}}$ and $d \in \boldsymbol{C}^{\mathcal{J}}$,
- $(\mathfrak{M}, c) \models [\boldsymbol{C}]_{\boldsymbol{r}} \varphi$ *iff* $(\mathfrak{M}, d) \models \varphi$ for every $d \in \mathfrak{C}$ s.t. $\langle c, d\rangle \in \boldsymbol{r}^{\mathcal{J}}$ and $d \in \boldsymbol{C}^{\mathcal{J}}$.

Then we define an *object knowledge base* $\mathcal{O}$ as a set of axioms of two forms:

$$\boldsymbol{a} : \varphi \ \mid \ \boldsymbol{C} : \varphi$$

where $\boldsymbol{a} \in M_I$, $\boldsymbol{C}$ is a c-concept and φ is an object formula. Such axioms have a straightforward reading: φ is true in context $\boldsymbol{a}$; and φ is true in every context which is $\boldsymbol{C}$. Formally, we specify those conditions as follows:

- $\mathfrak{M} \models \boldsymbol{a} : \varphi$ *iff* $(\mathfrak{M}, c) \models \varphi$ for $c = \boldsymbol{a}^{\mathcal{J}}$,
- $\mathfrak{M} \models \boldsymbol{C} : \varphi$ *iff* $(\mathfrak{M}, c) \models \varphi$ for every $c \in \boldsymbol{C}^{\mathcal{J}}$.

A pair $\mathcal{K} = (\mathcal{C}, \mathcal{O})$ is called an $\mathcal{ALC}_{\mathcal{ALC}}$ *knowledge base*. An interpretation $\mathfrak{M}$ is a *model* of $\mathcal{K}$ whenever all axioms in $\mathcal{K}$ are satisfied. A small example of an $\mathcal{ALC}_{\mathcal{ALC}}$ knowledge base is presented in Section 5.

4.2 Complexity and Expressiveness

Obviously, the expressiveness of $\mathcal{ALC}_{\mathcal{ALC}}$ properly subsumes that of $(\mathbf{K}_n)_{\mathcal{ALC}}$. In particular, the following relationship holds:

Proposition 2. *Concept satisfiability problem in* $(\mathbf{K}_n)_{\mathcal{ALC}}$ *w.r.t. global TBoxes is polynomially reducible to knowledge base satisfiability in* $\mathcal{ALC}_{\mathcal{ALC}}$.

To see this is indeed the case suppose $(C, \mathcal{T})$ is the problem of deciding whether concept C is satisfiable w.r.t. global TBox $\mathcal{T}$. Let C' and $\mathcal{T}'$ be the results of

replacing every $\Diamond_i$ with $\langle\top\rangle_{\boldsymbol{r}_i}$ and every $\Box_i$ with $[\top]_{\boldsymbol{r}_i}$ in C and $\mathcal{T}$, respectively, where for $i \neq j$ we have $\boldsymbol{r}_i \neq \boldsymbol{r}_j$. Further define $\mathcal{C} = \emptyset$ and $\mathcal{O} = \{\boldsymbol{c} : a : C'\} \cup \{\top : C \sqsubseteq D \mid C \sqsubseteq D \in \mathcal{T}'\}$. It clearly follows that C is satisfiable w.r.t. $\mathcal{T}$ in $(\mathbf{K}_n)_{\mathcal{ALC}}$ *iff* the knowledge base $\mathcal{K} = (\mathcal{C}, \mathcal{O})$ is satisfiable in $\mathcal{ALC}_{\mathcal{ALC}}$. Note, that the reduction holds even when object roles are interpreted rigidly.

This naturally means that the 2ExpTime lower bound established in Theorem 5 transfers immediately to $\mathcal{ALC}_{\mathcal{ALC}}$. But can it get even higher? Quite surprisingly, the answer is negative. Despite the increase of expressiveness, satisfiability problem in $\mathcal{ALC}_{\mathcal{ALC}}$ remains in 2ExpTime.

Theorem 6. *Deciding satisfiability of an $\mathcal{ALC}_{\mathcal{ALC}}$ knowledge base in which object roles are interpreted locally is* 2ExpTime*-complete.*

The proof of the upper bound is based on quasimodel elimination technique, which extends the one used for Theorem 3. In particular, every quasistate has to carry now also the type of the context which it represents and the set of object formulas which are satisfied in it.

To give a final insight into the expressiveness of the formalism, in more traditional terms of products of modal logics, we show that $\mathcal{ALC}_{\mathcal{ALC}}$ (with rigid roles) is equally expressive to the full $\mathcal{ALC}$ language over the union of two vocabularies interpreted in product models.

Let L_1 and L_2 be two $\mathcal{ALC}$ concept languages over disjoint vocabularies $\Gamma = (M_C, M_R, \emptyset)$ and $\Sigma = (N_C, N_R, \emptyset)$, respectively. Now, let $L_{1\times 2}$ be the $\mathcal{ALC}$ concept language over vocabulary $\Theta = (M_C \cup N_C, M_R \cup N_R, \emptyset)$. The semantics for $L_{1\times 2}$ is given through *product interpretations* $\mathcal{P} = (\mathfrak{C} \times \Delta, \cdot^{\mathcal{P}})$, which align every $r \in N_R$ along the 'vertical' dimension and every $\boldsymbol{p} \in M_R$ along the 'horizontal' one. Thus, $r^{\mathcal{P}}, \boldsymbol{p}^{\mathcal{P}} \subseteq (\mathfrak{C} \times \Delta) \times (\mathfrak{C} \times \Delta)$ and for every $u, v, w \in \mathfrak{C}$ and $x, y, z \in \Delta$:

$$\langle (u, x), (v, y)\rangle \in r^{\mathcal{P}} \rightarrow u = v \;\&\; \langle (w, x), (w, y)\rangle \in r^{\mathcal{P}},$$
$$\langle (u, x), (v, y)\rangle \in \boldsymbol{p}^{\mathcal{P}} \rightarrow x = y \;\&\; \langle (u, z), (v, z)\rangle \in \boldsymbol{p}^{\mathcal{P}}.$$

All concepts are interpreted as subsets of $\mathfrak{C} \times \Delta$. Additionally, we force every $\boldsymbol{A} \in M_C$ to be interpreted rigidly across the 'vertical' dimension, i.e., for every $v \in \mathfrak{C}$ and $x, y \in \Delta$ we assume:

$$(*) \quad (v, x) \in \boldsymbol{A}^{\mathcal{I}} \rightarrow (v, y) \in \boldsymbol{A}^{\mathcal{I}}$$

Finally, $\cdot^{\mathcal{P}}$ is extended inductively as usual. A concept $C \in L_{1\times 2}$ is satisfiable *iff* for some product model $\mathcal{P} = (\mathfrak{C} \times \Delta, \cdot^{\mathcal{P}})$ it is the case that $C^{\mathcal{P}} \neq \emptyset$. On the contrary to the others, the condition $(*)$ is rather uncommon in the realm of products of modal logics. Nevertheless, it captures precisely the difference between the semantics of the two sorts of concepts. Without it the sorts collapse into one, while the whole logic turns into a notational variant of $(\mathbf{K}_n)_{\mathcal{ALC}}$. It turns out that the following claim holds:

Theorem 7. *The language $L_{1\times 2}$ interpreted in product models is exactly as expressive as the concept language of $\mathcal{ALC}_{\mathcal{ALC}}$ interpreted in models with rigid interpretations of object roles.*

What follows from Theorem 7 is that the syntactic constraints of $\mathcal{ALC}_{\mathcal{ALC}}$, which make the logic more intuitive and well-behaved, by no means lead to

loss of expressiveness. Moreover, it shows that $\mathcal{ALC}_{\mathcal{ALC}}$ (at least in its concept component) does not seriously deviate from the usual products of modal logics. In principle, the only feature distinguishing it from $(\mathbf{K}_n)_{\mathcal{ALC}}$ (both with and without rigid roles) is the condition (*) imposed on the interpretations of selected concepts, which in $\mathcal{ALC}_{\mathcal{ALC}}$ we simply happen to call context concepts.

5 Contextual Ontologies — Example

One of the designated applications of $\mathcal{ALC}_{\mathcal{ALC}}$ is construction of *contextual ontologies*. The distinguishing feature of such ontologies is that they allow for varying the characterization of concepts according to contexts. Hence, $\mathcal{ALC}_{\mathcal{ALC}}$ can provide a good formal support for exchanging and integrating information in DL. Moreover, as the context knowledge base can be created independently from the object component, the framework encourages reuse of existing ontologies.

As an example of a contextual ontology, we present a simple representation of knowledge about the food domain contextualized with respect to geographic locations. Consider the (context) geographic knowledge base $\mathcal{C} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox.

$\mathcal{T} = \{$ (1) $\mathbf{Country} \sqsubseteq \exists \mathbf{location}.\mathbf{Europe} \sqcup \exists \mathbf{location}.\mathbf{America}$,
(2) $\mathbf{Region} \sqsubseteq \exists \mathbf{part\_of}.\mathbf{Country}$,
(3) $\mathbf{City} \sqsubseteq \exists \mathbf{has\_part}.\mathbf{Neighborhood}$ $\}$
$\mathcal{A} = \{$ (4) $\mathit{US} : \mathbf{Country}$,
(5) $\mathit{SanFrancisco} : \mathbf{City}$,
(6) $\mathit{California} : \mathbf{Region}$,
(7) $\mathbf{part\_of}(\mathit{California}, \mathit{US})$,
(8) $\mathit{France} : \mathbf{Country} \sqcap \exists \mathbf{location}.\mathbf{Europe}$ $\}$

Now, we define an (object) food ontology $\mathcal{O}$, contextualized with $\mathcal{C}$.

$\mathcal{O} = \{$ (a) $\top : \mathsf{Food} \equiv \mathsf{Meat} \sqcup \mathsf{Beverages} \sqcup \mathsf{Sea\_Food} \sqcup \mathsf{Grains}$
(b) $\top : \mathsf{Wine} \equiv \mathsf{WhiteWine} \sqcup \mathsf{RedWine}$
(c) $\top : (\mathtt{SauvignonBlanc} : \mathsf{WhiteWine})$
(d) $\mathbf{Country} : [\mathbf{Europe}]_{\mathbf{location}}(\mathsf{WhiteWine} \sqsubseteq \mathsf{Popular\_Beverage})$
(e) $\mathit{California} : \mathsf{WhiteWine} \sqsubseteq [\mathbf{Country}]_{\mathbf{part\_of}}\mathsf{Popular\_Wine}$
(f) $\mathit{US} : \mathsf{Popular\_Wine} \sqsubseteq \neg\mathsf{Popular\_Beverage}$
(g) $\mathit{SanFrancisco} : [\top]_{\mathbf{has\_part}}(\mathsf{WhiteWine} \sqsubseteq \neg\mathsf{Popular\_Wine})$ $\}$

Let us shortly highlight the intuition behind $\mathcal{O}$ by explaining some of the axiom definitions and the inferences they sanction. First, axioms (a)-(c) present geographic-independent terminology of the food domain. For example, by (c), `SauvignonBlanc` is a WhiteWine in any part of the world. Then, (d)-(g) characterize WhiteWine as Popular_Wine or Popular_Beverage according to different territories. We explain (d)-(g) in terms of `SauvignonBlanc`. By (d), in any **Country** that has as a **location Europe** (e.g., *France*) `SauvignonBlanc` is a Popular_Beverage. However, by (e)-(f), `SauvignonBlanc` is *not* a Popular_Beverage in *US*. This, is explained as follows: (e) establishes

that SauvignonBlanc is a Popular_Wine in any **Country** of which *California* is part of, namely *US*. Then, by (f), in the *US* any Popular_Wine is not a Popular_Beverage. Hence, SauvignonBlanc is not a Popular_Beverage in *US*. Although SauvignonBlanc is a Popular_Wine in *US*, this is does not necessarily transfer to more specific contexts. For instance, by (g), in every part of *SanFrancisco*, SauvignonBlanc is not in fact a Popular_Wine. In particular, by (3), there is at least one such **Neighborhood** in which this happens.

6 Conclusions and Future Work

We have presented a novel DL $\mathcal{ALC}_{\mathcal{ALC}}$ for representing and reasoning with contextual knowledge. Our approach is derived from McCarthy's conception of contexts as first-order objects which are describable in a first-order language. Formally, the logic extends the well-known $(\mathbf{K}_n)_{\mathcal{ALC}}$ with another sort of 'context' vocabulary interpreted over the **K**-dimension. The surprising conclusion is that the increase of the expressiveness of the logic due to this addition comes for no substantial price in terms of the worst-case complexity. The jump to 2ExpTime-completeness stems from the interaction of multiple modalities with global TBoxes and is inherent already to the underlying two-dimensional DLs.

We believe that with this work we have set the stage for a promising future research on similar combinations of DLs. Clearly, there are three major determinants of such formalisms which deserve a careful study: 1) *the expressiveness of the context language*, 2) *the expressiveness of the object language*, 3) *the level of interaction between the two*. Finding a proper balance between them is the key to identifying well-behaved and potentially useful fragments. One of the first directions, which we want to investigate, is to reduce the interaction between the languages by employing only **S5**-like operators. Such operators, e.g., $\langle \boldsymbol{C} \rangle \varphi$, would state that there exists a context of type C in which φ holds, without involving context roles. This modification should result in a better computational behavior and a somewhat simpler conceptual design of the language.

On the applied side, it could be interesting to consider a restricted fragment of the framework (a finite number of named contexts) for the task of ontology integration on the Semantic Web. Arguably, such fragment is sufficient to provide a logical underpinning for the ongoing endeavor of describing and linking OWL/RDFS knowledge sources in a context-sensitive manner.

Acknowledgements. We want to thank Carsten Lutz and Stefan Schlobach for many helpful discussions and suggestions on the ideas presented in this paper.

References

1. McCarthy, J.: Generality in artificial intelligence. Communications of the ACM 30, 1030–1035 (1987)
2. Buvač, S., Mason, I.A.: Propositional logic of context. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 412–419 (1993)
3. Buvač, S., Buvac, V., Mason, I.A.: Metamathematics of contexts. Fundamenta Informaticae 23, 412–419

4. Buvač, S.: Quantificational logic of context. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 412–419 (1996)
5. McCarthy, J.: Notes on formalizing context. In: Proc. of International Joint Conference on Artificial Intelligence, IJCAI 1993, pp. 555–560. Morgan Kaufmann, San Francisco (1993)
6. Guha, R.: Contexts: a formalization and some applications. PhD thesis, Stanford University (1991)
7. Guha, R., Mccool, R., Fikes, R.: Contexts for the semantic web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 32–46. Springer, Heidelberg (2004)
8. Nossum, R.: A decidable multi-modal logic of context. Journal of Applied Logic 1(1-2), 119–133 (2003)
9. Baader, F., Calvanese, D., Mcguinness, D.L., Nardi, D., Patel-Schneider, P.F.: The description logic handbook: theory, implementation, and applications. Cambridge University Press, Cambridge (2003)
10. Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. Journal of Data Semantics 1, 2003 (2003)
11. Cuenca Grau, B., Kutz, O.: Modular ontology languages revisited. In: Proc. of the Workshop on Semantic Web for Collaborative Knowledge Acquisition (2007)
12. Goczyla, K., Waloszek, W., Waloszek, A.: Contextualization of a DL knowledge base. In: The Proceedings of the International Workshop on Description Logics, DL 2007 (2007)
13. Grossi, D.: Desigining Invisible Handcuffs. Formal Investigations in Institutions and Organizations for Multi-Agent Systems. PhD thesis, Utrecht University (2007)
14. Lutz, C., Wolter, F., Zakharyaschev, M.: Temporal description logics: A survey. In: Proceedings of the Fourteenth International Symposium on Temporal Representation and Reasoning. IEEE Computer Society Press, Los Alamitos (2008)
15. Artale, A., Lutz, C., Toman, D.: A description logic of change. In: Veloso, M. (ed.) Proceedings of IJCAI 2007, pp. 218–223 (2007)
16. Artale, A., Kontchakov, R., Lutz, C., Wolter, F., Zakharyaschev, M.: Temporalising tractable description logics. In: Proceedings of the Fourteenth International Symposium on Temporal Representation and Reasoning (2007)
17. Wolter, F., Zakharyaschev, M.: Multi-dimensional description logics. In: IJCAI 1999: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, San Francisco, CA, USA, pp. 104–109 (1999)
18. Baader, F., Laux, A.: Terminological logics with modal operators. In: Mellish, C. (ed.) Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montréal, Canada, pp. 808–814. Morgan Kaufmann, San Francisco (1995)
19. Kurucz, A., Wolter, F., Zakharyaschev, M., Gabbay, D.M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and the Foundations of Mathematics, vol. 148. Elsevier, Amsterdam (2003)
20. Wolter, F., Zakharyaschev, M.: Satisfiability problem in description logics with modal operators. In: Proceedings of the Sixth Conference on Principles of Knowledge Representation and Reasoning, pp. 512–523. Morgan Kaufman, San Francisco (1998)
21. Klarman, S., Gutiérrez-Basulto, V.: $\mathcal{ALC}_{\mathcal{ALC}}$: a context description logic. Technical report, Vrije Universiteit Amsterdam (2010), http://klarman.synthasite.com/resources/JELIA2010TechRep.pdf
22. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)

Stable Belief Sets Revisited

Costas D. Koutras[1] and Yorgos Zikos[2,⋆]

[1] Department of Computer Science and Technology
University of Peloponnese
end of Karaiskaki Street, 22 100 Tripolis, Greece
ckoutras@uop.gr
[2] Graduate Programme in Logic, Algorithms and Computation (MPLA)
Department of Mathematics, University of Athens
Panepistimioupolis, 157 84 Ilissia, Greece
zikos@sch.gr

Abstract. *Stable belief sets* were introduced by R. Stalnaker in the early '80s, as a formal representation of the epistemic state for an ideal introspective agent. This notion motivated Moore's *autoepistemic logic* and greatly influenced *modal nonmonotonic reasoning*. Stalnaker stable sets possess an undoubtly simple and intuitive definition and can be elegantly characterized in terms of **S5** universal models or **KD45** situations. However, they do model an extremely perfect introspective reasoner and suffer from a Knowledge Representation (KR) version of the *logical omniscience problem*. In this paper, we vary the context rules underlying the positive and/or negative introspection conditions in the original definition of R. Stalnaker, to obtain variant notions of a stable epistemic state, which appear to be more plausible under the epistemic viewpoint. For these alternative notions of stable belief set, we obtain *representation theorems* using *possible world models* with *non-normal (impossible) worlds* and *neighborhood modal models*.

Keywords: modal epistemic logic, stable belief sets, nonmonotonic logics.

1 Introduction

Classical epistemic reasoning has been born and bred within the realm of Philosophical Logic and always had a modal flavour, already from its early inception in Hintikka's seminal work [12]. The *epistemic/doxastic logic* stream of research was very active for more than two decades and mainly revolved around constructing and discussing axiomatic systems which accurately describe the phenomena of *knowledge* and *belief*, from the perspective of a philosopher 'externally' reasoning about other entities' knowledge [18]. Many axiomatic systems have been proposed and several problems around this axiomatic approach to knowledge and belief have been identified and discussed (see [17,9]); in more recent years, epistemic and doxastic modal logics have found important applications in Knowledge Representation and Computer Science [5].

AI has created a completely new battlefield for epistemic reasoning, through the attempts to construct *nonmonotonic logics* in Knowledge Representation. The perspective

⋆ The second author gratefully acknowledges financial support by the Greek Ministry of Education, Lifelong Learning and Religious Affairs under the scheme of educational leave.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 221–233, 2010.
© Springer-Verlag Berlin Heidelberg 2010

of KR is much different, as the objective now is to describe 'internally' the epistemic capabilities of an intelligent agent reasoning on his/her own beliefs; see [2] for a recent, interesting discussion on the 'internal' vs 'external' viewpoint in modeling epistemic states. The use of modal languages and the import of techniques from classical epistemic reasoning have been employed from as early as the beginning of the '80s, when nonmonotonic logics have been announced. Modal nonmonotonic reasoning has been introduced through the work of D. McDermott and J. Doyle [22], with the use of a fixpoint construction which has been seriously criticized initially. *Stable belief sets* were introduced by R. Stalnaker at the same time; the short note [25] was written as a commentary on modal nonmonotonic logic and proposed the notion of a *stable set of beliefs* as a formal representation of the epistemic state of an ideally rational agent, with full introspective capabilities. Assuming a propositional language, endowed with a modal operator $\Box\varphi$, interpreted as 'φ *is believed*', a set of formulas S is a stable set if it is 'stable' under classical inference and epistemic introspection:

(i) $Cn_{\mathbf{PC}}(S) \subseteq S$

(ii) $\varphi \in S$ implies $\Box\varphi \in S$

(iii) $\varphi \notin S$ implies $\neg\Box\varphi \in S$

This notion proved to be of major importance in nonmonotonic modal logics. According to [25], R. Moore has written that this notion '.. *was a very important influence on the development of autoepistemic logic*' [23]; it also played a role in the logical investigations of Marek, Schwarz and Truszczyński on the McDermott & Doyle family of modal nonmonotonic modal logics [21]. Actually, the definition of stable sets was the first important step towards the idea of constructing epistemic logic(s) in nonmonotonic reasoning, without any appeal to classical modal logic (known as the '*Modality Si, Modal Logic No!*' motto of J. McCarthy).

The syntactic definition of stable sets is very natural and intuitive. Further research quickly revealed that they possess interesting properties while they do also admit simple and elegant semantic characterizations: they can be represented as the theories of universal ($\mathbf{S5}$) Kripke models, or alternatively, as the set of beliefs of an agent residing in a $\mathbf{KD45}$ situation (see [21, Chapt.8], [8]). It is not hard to see however, that Stalnaker's stable sets model an extremely perfect reasoner. In a sense, the situation is reminiscent of the '*logical omniscience*' problem in classical epistemic logic: normal modal logics of knowledge describe a reasoner who knows all the logical consequences of his/her beliefs; more on this, in section 2. Actually, the situation in Stalnaker's stable sets is a bit more uncomfortable: all tautologies are known and a stable set is a theory maximally consistent with provability in $\mathbf{S5}$. This raises some important philosophical and technical questions in modal non-monotonic reasoning, observed in [8] and addressed from a fine viewpoint in the work of Marek, Schwarz and Truszczyński [20].

So, stable sets are defined by calling for closure under (classical propositional logic and) suitable context rules, intended to capture positive and negative introspection on self beliefs. They are characterized by (and represented as theories of) well-known epistemic possible-worlds models, which have emerged in logics of classical epistemic reasoning ($\mathbf{S5}$, $\mathbf{KD45}$). It is absolutely natural to investigate whether one can define in

a natural way, variants of this notion which represent a less ideal and less omniscient agent, while retaining some of their interesting and useful properties; in this direction it is interesting from the KR viewpoint to work on the following two questions, related to the interplay between syntax and semantics of stable epistemic states:

- can we weaken the positive and/or negative introspection conditions (seen henceforth as context-dependent rules) in Stalnaker's original definition and still obtain a plausible (and perhaps, more pragmatic) notion of stable epistemic state? For such an emerging notion, does there exist a good model-theoretic representation?
- can we suitably replace **S5** and **KD45** in the semantic characterization of stable sets, with a possible-worlds model (possibly with *non-normal worlds* or a *neighborhood model*) determining some other classical modal logic and prove that the emerging notion of an epistemic state admits a syntactic definition in terms of (closure under) natural positive and negative introspection conditions?

In this paper, we work on the first of these two questions, actually the most important from the KR viewpoint. We vary conditions (ii) and (iii) in Stalnaker's definition to obtain three weaker notions of an epistemic state. We obtain semantic characterizations for the notions of stable sets we define; not surprisingly, we have to employ *impossible worlds* and *neighborhood modal models*. In Section 2 we gather the necessary technical background needed for our results, establishing notation and terminology. In Section 3 we very briefly mention some results we have obtained on the *determination* of *classical* and *regular modal logics*, with a notion of *strong provability* from premises. Sections 4 and 5 form the core of our results: we define, examine and characterize weaker notions of a stable epistemic state. In Section 6 we comment on related work and discuss open questions for future research. Due to space limitations, many proofs are omitted or sketched; the details can be found in the full paper (draft version in [16]).

2 Background Material

In this section we gather the necessary background material and results. For the basics of Modal Logic the reader is refered to the books [3,4,13] and for the essentials of modal nonmonotonic logics to [21]. We assume a modal propositional language $\mathcal{L}_{\Box}$, endowed with an epistemic operator $\Box\varphi$, read as '*it is believed that* φ *holds*'. Sentence symbols include $\top$ (for *truth*) and $\bot$ (for *falsity*). Some of the important axioms in epistemic/doxastic logic are:

K. $(\Box\varphi \wedge \Box(\varphi \supset \psi)) \supset \Box\psi$

T. $\Box\varphi \supset \varphi$ (axiom of true, justified knowledge)

D. $\Box\varphi \supset \neg\Box\neg\varphi$ or $\neg(\Box\varphi \wedge \Box\neg\varphi)$ (consistent belief)

4. $\Box\varphi \supset \Box\Box\varphi$ (positive introspection)

5. $\neg\Box\varphi \supset \Box\neg\Box\varphi$ (negative introspection)

w5. $(\varphi \wedge \neg\Box\varphi) \supset \Box\neg\Box\varphi$ (negative introspection limited to true facts)

p5. $(\neg\Box\varphi \wedge \neg\Box\neg\varphi) \supset \Box\neg\Box\varphi$ (negative introspection limited to possible facts)

Modal logics are sets of modal formulas containing classical propositional logic (i.e. containing all tautologies in the augmented language $\mathcal{L}_{\Box}$) and closed under rule $\mathbf{MP}.\frac{\varphi,\varphi\supset\psi}{\psi}$. The smallest modal logic is denoted as $\mathbf{PC}$ (propositional calculus in the augmented language). *Normal* are called those modal logics, which contain all instances of axiom $\mathbf{K}$ and are closed under rule

$$\mathbf{RN}.\ \frac{\varphi}{\Box\varphi}$$

By $\mathbf{KA_1}\ldots\mathbf{A_n}$ we denote the normal modal logic axiomatized by axioms $\mathbf{A_1}$ to $\mathbf{A_n}$. Well-known epistemic logics comprise $\mathbf{KT45}$ ($\mathbf{S5}$) (a *strong logic of knowledge*) and $\mathbf{KD45}$ (a *logic of consistent belief*). Throughout this paper we use the notion of strong provability from a theory I. In the case of a normal modal logic Λ we write $I \vdash^{\mathrm{RN}}_{\Lambda} \varphi$ iff there is a Hilbert-style proof, where each step of the proof is a formula, which is a tautology in $\mathcal{L}_{\Box}$, or an instance of $\mathbf{K}$, or an instance of an axiom of Λ, or a member of I, or a result of applying $\mathbf{MP}$ or $\mathbf{RN}$ to formulas of previous steps.

Normal modal logics are interpreted over Kripke models: a *Kripke model* $\mathfrak{M} = \langle W, R, V\rangle$ consists of a set of possible worlds W and a binary relation between them $R \subseteq W \times W$: whenever wRv, we say that world w '*sees*' world v. The valuation V determines which propositional variables are true inside each possible world. Within a world w, the propositional connectives ($\neg$, $\supset$, $\wedge$, $\vee$) are interpreted classically, while $\Box\varphi$ is true at w iff it is true in every world '*seen*' by w, notation: $(\mathfrak{M}, w \Vdash \Box\varphi$ iff $(\forall v \in W)(wRv \Rightarrow \mathfrak{M}, v \Vdash \varphi))$. A logic Λ is *determined* by a class of models iff it is *sound* and *complete* with respect to this class; it is known that $\mathbf{S5}$ is determined by the class of Kripke models with a *universal* relation, while $\mathbf{KD45}$ is determined by the class of models consisting of a world which '*sees*' a '*cluster*' (i.e. a universally connected subset) of worlds and which does not 'see' itself; every model of this class has the form $\langle\{w\} \cup W, (\{w\} \cup W) \times W, V\rangle$).

Normal modal epistemic logics suffer from the so-called *logical omniscience* problem, which can be attributed to axiom $\mathbf{K}$ and rule $\mathbf{RN}$. Because of the latter, all tautologies are known. Also, because of the axiom $\mathbf{K}$, logical consequences of knowledge constitute knowledge, something unreasonable in realistic situations. Note however that axiom $\mathbf{K}$ and axioms as simple as $\mathbf{N}.\Box\top$ are unavoidable in Kripke models and ubiquitous in normal modal logics. A first step towards solving the logical omniscience problem is by defining *regular* modal logics which contain $\mathbf{K}$, but substitute rule $\mathbf{RN}$ for rule

$$\mathbf{RM}.\ \frac{\varphi \supset \psi}{\Box\varphi \supset \Box\psi}$$

We denote by $\mathbf{KA_1}\ldots\mathbf{A_{nR}}$ the regular modal logic axiomatized by axioms $\mathbf{A_1}$ to $\mathbf{A_n}$. In this case (of a regular modal logic Λ) we use again a notion of strong provability, where the application of $\mathbf{RN}$ in any step of the proof is replaced by $\mathbf{RM}$, and we write $I \vdash^{\mathrm{RM}}_{\Lambda} \varphi$. Regular modal logics are interpreted on a strange species of possible world models, introduced by Kripke too; we will call them *q-models* here ($\mathfrak{M} = \langle W, N, R, V\rangle$). We now have two kinds of worlds: *normal* worlds (N), which behave in the way we described above and *non-normal* (also called *queer* or *impossible*) worlds ($W \setminus N$), where nothing is known/believed ($\Box\varphi$ is *never* true there) and

everything is consistent to our state of affairs ($\neg\Box\neg\varphi$ is *always* true there). Within a world w, the propositional connectives are interpreted classically and $\Box\varphi$ is true at w iff $w \in N$ and $(\forall v \in W)(wRv \Rightarrow \mathfrak{M}, v \Vdash \varphi)$).

This however does not avoid the effect of $\mathbf{K}$: to be able to eliminate $\mathbf{K}$ we have to resort to *neighborhood* (also called *Montague* or *minimal* in [4]) semantics. Neighborhood structures are very flexible and they are considered to be the standard semantic tool used to reason about non-normal modal logics; see [11] for some recent results. In this kind of models, which we will call *n-models*, each world does not 'see' other worlds but it is associated to possible '*neighborhoods*' (subsets) of possible worlds: an n-model is a triple $\mathfrak{N} = \langle W, E, V\rangle$, where W is any set of worlds, E is any function assigning to any world, its sets of 'neighboring' worlds (i.e. $E : W \to \mathcal{P}(\mathcal{P}(W))$) and V is again a valuation. The interpretation of any formula is exactly as in Kripke models, except of the formulas of the form $\Box\varphi$; such a formula is true at w iff the set of worlds where φ holds, belong to the possible neighborhoods of w: $\overline{V}(\varphi) = \{v \in W \mid \mathfrak{N}, v \Vdash \varphi\} \in E(w)$. *Theory* of a (Kripke, q- or n-) model $\mathfrak{M}$ (denoted as $Th(\mathfrak{M})$) is the set of all formulae being true in every world of $\mathfrak{M}$. Having a q-model, we can define a pointwise equivalent n-model:

Definition 1. *Let* $\mathfrak{M} = \langle W, N, R, V\rangle$ *be a q-model and* $\mathfrak{N}_\mathfrak{M} = \langle W, E, V\rangle$ *the n-model, where* $E(w) = \{X \subseteq W \mid R_w \subseteq X\}$[1]*, if* $w \in N$*, and* $E(w) = \varnothing$*, if* $w \in W \setminus N$*.* $\mathfrak{N}_\mathfrak{M}$ *is called* the equivalent n-model produced by $\mathfrak{M}$.

This notion of 'equivalence' seems to be appropriate, because of the following result:

Proposition 1. *Let* $\mathfrak{M} = \langle W, N, R, V\rangle$ *be a Kripke q-model. Then* $(\forall\varphi \in \mathcal{L}_\Box)\ (\forall w \in W)\ (\mathfrak{M}, w \Vdash \varphi \iff \mathfrak{N}_\mathfrak{M}, w \Vdash \varphi)$

The directed graph $\mathfrak{F} = \langle W, R\rangle$, underlying a (Kripke, q-, or n-) model, is called a *frame*. A modal logic Λ is called *classical* iff it is closed under the rule

$$\mathbf{RE}.\ \frac{\varphi \equiv \psi}{\Box\varphi \equiv \Box\psi}$$

See [4] for results on the characterization of classical modal logics in terms of Montague semantics. By $\mathbf{A_1} \ldots \mathbf{A_n C}$ we denote the classical modal logic axiomatized by axioms $\mathbf{A_1}$ to $\mathbf{A_n}$. Again, in this case (of a classical modal logic Λ) we use a notion of strong provability, where in any step of the proof the use of $\mathbf{K}$-instances is prohibited and the application of $\mathbf{RN}$ is replaced by $\mathbf{RE}$; we write $I \vdash^{\mathrm{RE}}_{\Lambda} \varphi$.[2] It is convenient in our paper to consider the following context-dependent versions of the modal rules mentioned up to this point: assuming a set S of modal formulae, we denote the rules

$$\mathbf{RN_c}.\ \frac{\varphi \in S}{\Box\varphi \in S} \qquad \mathbf{NI_c}.\ \frac{\varphi \notin S}{\neg\Box\varphi \in S}$$

$$\mathbf{RM_c}.\ \frac{\varphi \supset \psi \in S}{\Box\varphi \supset \Box\psi \in S} \qquad \mathbf{RE_c}.\ \frac{\varphi \equiv \psi \in S}{\Box\varphi \equiv \Box\psi \in S}$$

[1] $R_w = \{v \in W \mid wRv\}$.

[2] Usually, we just use the notion $\vdash_\Lambda$ instead of $\vdash^{\mathrm{RN}}_{\Lambda}$, $\vdash^{\mathrm{RM}}_{\Lambda}$ or $\vdash^{\mathrm{RE}}_{\Lambda}$, if it is clear by the context whether Λ is normal, regular or classical respectively.

Stalnaker stable sets are closed under propositional reasoning (i), under rule $\mathbf{RN_c}$ (ii) and rule $\mathbf{NI_c}$ (iii). The following theorem gathers some of their useful properties; see [21] for a proof.

Theorem 1. *a) If a set S is stable, then it is closed under strong* $\mathbf{S5}$ *provability*[3]*. In particular, it contains every instance of* $\mathbf{K}$*,* $\mathbf{T}$*,* $\mathbf{4}$*, and* $\mathbf{5}$*.*
b) A set S is stable iff it is the theory of a Kripke model with a universal accessibility relation.
c) A set S is stable iff it is the set of formulae believed in a world w of a $\mathbf{KD45}$*-model, i.e. S is stable iff there is a* $\mathbf{KD45}$*-model $\mathfrak{M} = \langle W, R, V\rangle$ and $(\exists w \in W)S = \{\varphi \in \mathcal{L}_\Box \mid \mathfrak{M}, w \Vdash \Box\varphi\}$.*

3 A Digression: Regular and Classical Modal Logics

To be able to characterize the stable sets introduced in the subsequent sections, we have to work on the proof theory of regular and classical modal logics with a notion of strong provability from premises. The results are original, in the sense that they have not been developed elsewhere, yet they are quite lengthy to be included in this extended abstract and they are left for the full paper; a preliminary version can be found at [16]

For regular modal logics, we employ the axioms:

$\mathbf{4}_\top$. $\Box\varphi \supset \Box(\Box\top \supset \Box\varphi)$

$\mathbf{B}_\top$. $(\varphi \wedge \Box\top) \supset \Box\neg\Box\neg\varphi$

$\mathbf{5}_\top$. $(\neg\Box\varphi \wedge \Box\top) \supset \Box\neg\Box\varphi$

The first of them appears in [24] and all of them seem useful in our KR investigations. Furthermore, for a q-frame $\mathfrak{F} = \langle W, N, R\rangle$, we employ following property:

$(\mathsf{U_q})$ $(\forall w \in N)(\forall v \in W)wRv$

We also use the following definitions for a regular modal logic Λ: a theory I is called *consistent with* Λ ($c\Lambda$-theory) iff $I \nvdash_\Lambda \bot$, and T is called *I-consistent with* Λ ($Ic\Lambda$-theory) iff $(\forall n \in \mathbb{N}, \forall\varphi_0, \ldots, \varphi_n \in T)\ I \nvdash_\Lambda \varphi_0 \wedge \ldots \wedge \varphi_n \supset \bot$ - otherwise, T is called $Iinc\Lambda$. T is called *maximal I-consistent with* Λ ($mIc\Lambda$-theory) iff T is $Ic\Lambda$ and $(\forall\psi \notin T)\ T \cup \{\psi\}$ is $Iinc\Lambda$. Then, we can prove following facts:

Proposition 2. *Let I be a $c\Lambda$-theory and Γ a $mIc\Lambda$-theory. Then*
(i) *Γ is closed under* $\mathbf{MP}$.
(ii) $(\forall\varphi \in \mathcal{L}_\Box)(\varphi \in \Gamma$ *or* $\neg\varphi \in \Gamma)$
(iii) $(\forall\varphi \in \mathcal{L}_\Box)(I \vdash_\Lambda \varphi \Rightarrow \varphi \in \Gamma)$
(iv) $(\forall\varphi \in \mathcal{L}_\Box)(\varphi \wedge \psi \in \Gamma \Leftrightarrow (\varphi \in \Gamma$ *and* $\psi \in \Gamma))$

Furthermore, using a typical *Lindenbaum construction* and the (appropriate version of the) canonical model method, we can define the *canonical model* $\mathfrak{M}^{\Lambda,I}$ (for a regular modal logic Λ and for a $c\Lambda$-theory I).

[3] i.e. $S = \{\varphi \in \mathcal{L}_\Box \mid S \vdash_{\mathbf{S5}} \varphi\}$.

Definition 2. *Let Λ be any regular modal logic and I be any $c\Lambda$-theory. The* canonical model $\mathfrak{M}^{\Lambda,I}$ *for Λ and I is the Kripke q-model, which is defined as the quadruple $\langle W^{\Lambda,I}, N^{\Lambda,I}, R^{\Lambda,I}, V^{\Lambda,I}\rangle$, where:*
(i) $W^{\Lambda,I} = \{\Gamma \subseteq \mathcal{L}_\Box \mid \Gamma : mIc\Lambda\}$
(ii) $N^{\Lambda,I} = \{\Gamma \in W^{\Lambda,I} \mid \Box\top \in \Gamma\}$
(iii) $(\forall \Gamma, \Delta \in W^{\Lambda,I})(\Gamma R^{\Lambda,I} \Delta$ *iff* $(\forall \varphi \in \mathcal{L}_\Box)(\Box\varphi \in \Gamma \Rightarrow \varphi \in \Delta))$
(iv) $(\forall p \in \Phi)(V^{\Lambda,I}(p) = \{\Gamma \in W^{\Lambda,I} \mid p \in \Gamma\})$

We can prove the following characterization of a useful regular modal logic, namely $\mathbf{S5'_R} = \mathbf{KT4_\top B_{\top R}}$

Proposition 3. $\mathbf{S5'_R}$ *is strongly complete with respect to all q-frames, for which* $(\mathsf{U_q})$ *holds.*

Actually, a more general result can be proved, which will be useful in subsequent sections.

Proposition 4. *Let I be a $c\Lambda$-theory. Then,* $(\forall \varphi \in \mathcal{L}_\Box)$

$$\mathfrak{M}^{\Lambda,I} \Vdash \varphi \iff I \vdash_\Lambda \varphi$$

About classical modal logics, due to space limitations, we can only mention that a similar result can be proved for a classical logic Λ and for a corresponding strong notion of RE-provability, having defined the appropriate canonical n-model:

Definition 3. *Let Λ be a classical modal logic and I be a $c\Lambda$-theory. The* canonical model $\mathfrak{N}^{\Lambda,I}$ *for Λ and I is the n-model, which is defined as the triple $\langle W^{\Lambda,I}, E^{\Lambda,I}, V^{\Lambda,I}\rangle$, where:*
(i) $W^{\Lambda,I} = \{\Gamma \subseteq \mathcal{L}_\Box \mid \Gamma : mIc\Lambda\}$
(ii) $(\forall \Gamma \in W^{\Lambda,I})(\forall \varphi \in \mathcal{L}_\Box)(|\varphi|_{\Lambda,I} \in E^{\Lambda,I}(\Gamma) \iff \Box\varphi \in \Gamma)$
where $|\varphi|_{\Lambda,I} = \{\Gamma \in W^{\Lambda,I} \mid \varphi \in \Gamma\}$
(iii) $(\forall p \in \Phi)(V^{\Lambda,I}(p) = \{\Gamma \in W^{\Lambda,I} \mid p \in \Gamma\})$

4 RM-Stable Theories

Having set the appropriate background, we proceed to define our first variant of a stable belief set by taking the most obvious road: substituting $\mathbf{RM_c}$ for $\mathbf{RN_c}$ in Stalnaker's definition.

Definition 4. *A theory $S \subseteq \mathcal{L}_\Box$ is called* RM-stable *iff*

(i) $\mathbf{PC} \subseteq S$ *and S is closed under* $\mathbf{MP}$
(ii) S *is closed under rule* $\mathbf{RM_c}$. $\frac{\varphi \supset \psi \in S}{\Box\varphi \supset \Box\psi \in S}$
(iii) S *is closed under rule* $\mathbf{NI_c}$. $\frac{\varphi \notin S}{\neg\Box\varphi \in S}$

The first observation is that the axiom $\Box\top$ plays here a role similar to the one encountered in non-normal modal logics, where $\Box\top$ eliminates queer worlds and leads to the realm of normal modal logics. Addition of $\Box\top$ to an RM-stable set leads to the classical Stalnaker notion.

Fact 2. *A theory S is a Stalnaker stable set iff it is an RM-stable set containing* $\Box\top$.

From the proof-theoretic viewpoint, the following result shows that RM-stable sets stand to the regular logic $\mathbf{S5'_R}$, as classical Stalnaker (RN-)stable sets stand to $\mathbf{S5}$. The following theorem should be compared to theorem 1a), and it is stated without a proof.

Theorem 3. *a) If a set S is RM-stable, then it is closed under strong $\mathbf{S5'_R}$ provability. In particular, it contains every instance of $\mathbf{K}$, $\mathbf{T}$, $\mathbf{4}_\top$, and $\mathbf{B}_\top$.*
b) If a set S is RM-stable and consistent, then it is a consistent with $\mathbf{S5'_R}$ theory ($c\mathbf{S5'_R}$-theory)

Representation theory for RM-stable sets. We can provide *model-theoretic characterizations* of RM-stable theories in terms of q-models and n-models. We can set RM-stable theories in an one-to-one-correspondence to theories of q-models consisting of a cluster of normal worlds 'seeing' every non-normal world (if any). We can also characterize RM-stable sets as the set of beliefs held within a normal world in such a q-model.

Theorem 4. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is RM-stable iff there is a q-model $\mathfrak{M} = \langle W, N, R, V\rangle$ satisfying property $(\mathsf{U_q})$ s.t. $Th(\mathfrak{M}) = S$.*

PROOF. ($\Rightarrow$) Since S is RM-stable, by Theor.3a), S contains any formula of the form of $\mathbf{T}$, $\mathbf{4}_\top$ or $\mathbf{B}_\top$. Hence, since S is consistent, by Theor.3b), S is a $c\mathbf{S5'}_R$-theory. So, model $\mathfrak{M}^{\mathbf{S5'}_R,S}$ does exist and, by Prop.4, $Th(\mathfrak{M}^{\mathbf{S5'}_R,S}) = \{\varphi \in \mathcal{L}_\Box \mid S \vdash_{\mathbf{S5'}_R} \varphi\}$. Consequently, by Theor.3a), $Th(\mathfrak{M}^{\mathbf{S5'}_R,S}) = S$.

Now, consider any $\Gamma \in N^{\mathbf{S5'}_R,S}$ and $\Delta \in W^{\mathbf{S5'}_R,S}$. For any $\psi \in \mathcal{L}_\Box$ s.t. $\Box\psi \in \Gamma$, since Γ is $Sc\mathbf{S5'}_R$, $\neg\Box\psi \notin \Gamma$. Suppose now that $\neg\Box\psi$ were in S. Then, $S \vdash_{\mathbf{S5'}_R} \neg\Box\psi$, hence, by Prop.2(iii), $\neg\Box\psi \in \Gamma$, which is a contradiction. So $\neg\Box\psi \notin S$. But, S is RM-stable, so, by $\mathbf{NI_c}$, $\psi \in S$, hence, $S \vdash_{\mathbf{S5'}_R} \psi$, consequently, again by Prop.2(iii), $\psi \in \Delta$. So, by Def.2, $\Gamma R^{\mathbf{S5'}_R,S}\Delta$.
($\Leftarrow$) (i) Since $Th(\mathfrak{M})$ contains every tautology in $\mathcal{L}_\Box$ and is closed under (MP), Def.4(i) is trivially established.
(ii)-$\mathbf{RM_c}$ Let $\varphi, \psi \in \mathcal{L}_\Box$ s.t. $\varphi \supset \psi \in Th(\mathfrak{M})$ and $w \in W$ s.t. $\mathfrak{M}, w \Vdash \Box\varphi$. Then, $w \in N$ and $(\forall v \in W)\ wRv \Rightarrow \mathfrak{M}, v \Vdash \varphi$. Therefore, since $\varphi \supset \psi \in Th(\mathfrak{M})$, $\mathfrak{M}, v \Vdash \psi$, hence, $\mathfrak{M}, w \Vdash \Box\psi$. So, $\Box\varphi \supset \Box\psi \in Th(\mathfrak{M})$.
(iii)-$\mathbf{NI_c}$ Let $\varphi \in \mathcal{L}_\Box$ s.t. $\varphi \notin Th(\mathfrak{M})$ i.e. there is $v \in W$ s.t. $\mathfrak{M}, v \nVdash \varphi$. Let now be any $w \in W$. If $w \in W \setminus N$, then, by definition of q-models, $\mathfrak{M}, w \Vdash \neg\Box\varphi$. If $w \in N$, then again, since wRv and $\mathfrak{M}, v \nVdash \varphi$, $\mathfrak{M}, w \Vdash \neg\Box\varphi$.
So, $\neg\Box\varphi \in Th(\mathfrak{M})$. ∎

The following characterization is the parallel to the characterization of Stalnaker stable sets in terms of beliefs held 'inside' a $\mathbf{KD45}$ situation, and as such, seems amenable to generalization in multi-agent situations (as argued convincingly in [10]). Its proof can be found in [16].

Proposition 5. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is RM-stable iff there is a q-model $\mathfrak{M} = \langle W, N, R, V\rangle$ and $u \in N$ s.t. $S = \{\varphi \in \mathcal{L}_\Box \mid \mathfrak{M}, u \Vdash \Box\varphi\}$ and $(\forall w \in N)(\forall v \in W \setminus \{u\})wRv$.*

By using Theorem 4, we also obtain a representation for RM-stable sets, in terms of neighborhood semantics.

Proposition 6. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is RM-stable iff there is an n-model $\mathfrak{N} = \langle W, E, V\rangle$ s.t. $Th(\mathfrak{N}) = S$ and $(\forall w \in W)(E(w) = \varnothing$ or $E(w) = \{W\})$.*

PROOF. ($\Rightarrow$) By Theor.4, there is a q-model $\mathfrak{M} = \langle W, N, R, V\rangle$ s.t. $Th(\mathfrak{M}) = S$ and $(\forall w \in N)(\forall v \in W)wRv$. Consider now $\mathfrak{N}_\mathfrak{M} = \langle W, E, V\rangle$, the equivalent n-model produced by $\mathfrak{M}$ (see Def.1). By Prop.1 follows immediately that $Th(\mathfrak{N}_\mathfrak{M}) = Th(\mathfrak{M}) = S$. Furthermore, if $w \in W \setminus N$, then $E(w) = \varnothing$ and if $w \in N$, then $E(w) = \{X \subseteq W \mid R_w \subseteq X\} = \{W\}$, since $(\forall v \in W)wRv$.

($\Leftarrow$) (i) Since $Th(\mathfrak{N})$ contains every tautology in $\mathcal{L}_\Box$ and is closed under (MP), Def.4(i) is established. (ii)-$\mathbf{RM_c}$ Let $\varphi, \psi \in \mathcal{L}_\Box$ s.t. $\varphi \supset \psi \in Th(\mathfrak{N})$ and $w \in W$ s.t. $\mathfrak{N}, w \Vdash \Box\varphi$. Then, $\overline{V}(\varphi) \in E(w)$, hence, $E(w) = \{W\}$ and $\overline{V}(\varphi) = W$. Therefore, since $\varphi \supset \psi \in Th(\mathfrak{N})$, $(\forall w \in W)\mathfrak{N}, w \Vdash \psi$, i.e. $\overline{V}(\psi) = W$, so, $\overline{V}(\psi) \in E(w)$, hence, $\mathfrak{N}, w \Vdash \Box\psi$. So, $\Box\varphi \supset \Box\psi \in Th(\mathfrak{N})$. (iii)-$\mathbf{NI_c}$ Let $\varphi \in \mathcal{L}_\Box$ s.t. $\varphi \notin Th(\mathfrak{N})$ i.e. $\overline{V}(\varphi) \neq W$. Let now be any $w \in W$. $E(w) = \varnothing$ or $E(w) = \{W\}$, so in both cases, $\overline{V}(\varphi) \notin E(w)$. Hence, $\mathfrak{N}, w \Vdash \neg\Box\varphi$. So, $\neg\Box\varphi \in Th(\mathfrak{N})$. ∎

Finally, we can obtain the following representation of Stalnaker stable sets, in terms of n-models, given for the first time.

Proposition 7. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is stable iff there is an n-model $\mathfrak{N} = \langle W, E, V\rangle$ s.t. $Th(\mathfrak{N}) = S$ and $(\forall w \in W)E(w) = \{W\}$.*

5 RE-Stable Theories

Following a typical route, it is tempting to attempt weakening further the positive introspection condition. Rule $\mathbf{RE_c}$ seems the obvious candidate, but we have soon to face the obvious problem that the introspective reasoner should be able to distinguish tautologies as equivalent formulae. We have then to consider the addition of $\Box\top$ and this leads us to the following generic notion:

Definition 5. *A theory $S \subseteq \mathcal{L}_\Box$ is called* RE-stable *iff*

(i) $\mathbf{PC} \subseteq S$ *and S is closed under* $\mathbf{MP}$

(ii) $\Box\top \in S$

(iii) S is closed under rule $\mathbf{RE_c}$. $\frac{\varphi \equiv \psi \in S}{\Box\varphi \equiv \Box\psi \in S}$

We can prove that RE-stable theories are consistent with strong provability in classical modal logics:

Lemma 1. *Let S be an RE-stable theory containing axioms $\mathbf{A_1}, \ldots, \mathbf{A_n}$. Then*
a) S is closed under strong $\mathbf{A_1} \ldots \mathbf{A_nC}$ provability.
b) If S is also consistent, then it is a consistent with $\mathbf{A_1} \ldots \mathbf{A_nC}$ theory ($c\mathbf{A_1} \ldots \mathbf{A_nC}$-theory)

But, it comes that by adding $\Box\top$, we get nothing less than $\mathbf{RN_c}$, as in the original definition.

Lemma 2. *Any RE-stable theory is closed under* $\mathbf{RN_c}$.

PROOF. Let S be an RE-stable theory and $\varphi \in S$. Since $\varphi \supset (\top \supset \varphi) \in S$, by Def.5(i), $\top \supset \varphi \in S$. Furthermore, $\varphi \supset \top \in S$, so, by Def.5(i), $\top \equiv \varphi \in S$, hence, by $\mathbf{RE_c}$, $\Box\top \equiv \Box\varphi \in S$, and, by Def.5(i), $\Box\top \supset \Box\varphi \in S$, and finally, by Def.5(ii) and (i), $\Box\varphi \in S$. ∎

This means we have to proceed to different notions of negative introspection and by doing so, we obtain two different notions of RE-stable sets.

5.1 REw-Stable Theories

We introduce the following context rule[4] for negative introspection:

$$\mathbf{NI_{c-w}}\cdot \frac{\neg\varphi \notin S}{\Box\varphi \in S \quad \text{or} \quad \neg\Box\varphi \in S}$$

which 'says' that *if φ is consistent with what is believed, something is known about it*.

Definition 6. *An RE-stable theory S is called* REw-stable *iff it is closed under* $\mathbf{NI_{c-w}}$.

We readily prove the presence of axiom $\mathbf{w5}$ and then, we can obtain a representation theorem for REw-stable theories in terms of n-models.

Lemma 3. *Every instance of axiomatic scheme* $\mathbf{w5}$ *is contained in any REw-stable theory.*

PROOF. Let S be an REw-stable theory and $\varphi \in \mathcal{L}_\Box$.
If $\neg\varphi \in S$ or $\Box\varphi \in S$, then, by Def.5(i), $\mathbf{w5} \in S$.
If $\neg\varphi \notin S$ and $\Box\varphi \notin S$, then, by $\mathbf{NI_{c-w}}$, $\neg\Box\varphi \in S$, and , by Lem.2, $\Box\neg\Box\varphi \in S$, hence again, $\mathbf{w5} \in S$. ∎

Theorem 5. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is REw-stable iff there is an n-model $\mathfrak{N} = \langle W, E, V\rangle$ s.t. $Th(\mathfrak{N}) = S$ and $(\forall w \in W)\ W \in E(w)$ (1) and $(\forall v \in W)(E(v) \setminus E(w) \subseteq \{\varnothing\})$ (2)*

PROOF. ($\Rightarrow$) Since S is REw-stable, by Lem.3, S contains $\mathbf{w5}$, hence, since S is RE-stable and consistent, by Lem.1b), S is a $c\mathbf{w5}_C$-theory. So, model $\mathfrak{N}^{\mathbf{w5}_C,S}$ does exist. For simplicity, let us denote $\mathfrak{N}^{\mathbf{w5}_C,S}$ as $\mathfrak{N} = \langle W, E, V\rangle$. Then, by Prop.4 (for classical logics and RE-proofs), $Th(\mathfrak{N}) = \{\varphi \in \mathcal{L}_\Box \mid S \vdash_{\mathbf{w5}_C} \varphi\}$. Consequently, by Lem.1a), $Th(\mathfrak{N}) = S$. Now, fix any $\mathsf{r} \in W$.
(1) By Def.5(i), $\top \in S$, so, by Prop.2(iii), $(\forall\Delta \in W)\top \in \Delta$, hence, since every Δ is a $mSc\mathbf{w5}_C$-theory, $|\top|_{\mathbf{w5}_C,S} = W$. But, by Def.5(ii), $\Box\top \in S$, i.e., by Prop.2(iii),

[4] Perhaps, it would be more appropriate to call this a *postulate*, rather than a (context) rule. We wish however to stick with the tradition of modal nonmonotonic logic which stresses the presence of a *context* in the involved inferences.

$\Box\top \in \mathsf{r}$, hence, by Def.3(ii), $|\top|_{\mathbf{w5}_C,S} \in E(\mathsf{r})$. Consequently, $W \in E(\mathsf{r})$.
(2) Consider any $\Delta \in W$ and let $Y \subseteq W$ s.t. $Y \in E(\Delta)$ but $Y \notin E(\mathsf{r})$. Then, by Def.3(ii), there must be a $\varphi \in \mathcal{L}_\Box$ s.t. $Y = |\varphi|_{\mathbf{w5}_C,S}$ and $\Box\varphi \in \Delta$ (I)
But, since $Y \notin E(\mathsf{r})$, $\Box\varphi \notin \mathsf{r}$, hence, by Prop.2(iii), $\Box\varphi \notin S$ (II)

Suppose now, for the sake of contradiction, that $Y \neq \varnothing$. Then, there is a $\Xi \in Y$. Since $Y = |\varphi|_{\mathbf{w5}_C,S}$, $\varphi \in \Xi$, and since Ξ is consistent, $\neg\varphi \notin \Xi$, so, by Prop.2(iii), $\neg\varphi \notin S$ (III)

Now, (II) and (III) imply by $\mathbf{NI_{c-w}}$, $\neg\Box\varphi \in S$, therefore, again by Prop.2(iii), $\neg\Box\varphi \in \Delta$, hence, by (I), Δ is inconsistent, which is a contradiction. So, $Y = \varnothing$.

$(\Leftarrow)$(i) Since $Th(\mathfrak{N})$ contains every tautology in $\mathcal{L}_\Box$ and is closed under (MP), Def.5(i) is trivially established.(ii) Since $\overline{V}(\top) = W$ and, by (1), $(\forall w \in W)W \in E(w)$, $\Box\top \in Th(\mathfrak{N})$.(iii)-$\mathbf{RE_c}$ Let $\varphi, \psi \in \mathcal{L}_\Box$ s.t. $\varphi \equiv \psi \in Th(\mathfrak{N})$. Then, $\overline{V}(\varphi) = \overline{V}(\psi)$, hence, $(\forall w \in W)$ $(\overline{V}(\varphi) \in E(w) \iff \overline{V}(\psi) \in E(w))$, consequently, $\Box\varphi \equiv \Box\psi \in Th(\mathfrak{N})$.$\mathbf{NI_{c-w}}$ Let $\varphi \in \mathcal{L}_\Box$ s.t. $\neg\varphi \notin Th(\mathfrak{N})$ and $\Box\varphi \notin Th(\mathfrak{N})$. Then, $\overline{V}(\neg\varphi) \neq W$ and $(\exists w \in W)$ $\mathfrak{N}, w \nVdash \Box\varphi$, i.e. $\overline{V}(\varphi) \neq \varnothing$ and $(\exists w \in W)\overline{V}(\varphi) \notin E(w)$. Now, suppose for the sake of contradiction, that there is a $v \in W$ s.t. $\overline{V}(\varphi) \in E(v)$. Then, $\overline{V}(\varphi) \in E(v) \setminus E(w)$, hence, by (2), $\overline{V}(\varphi) = \varnothing$, which is a contradiction. So, $(\forall v \in W)$ $\overline{V}(\varphi) \notin E(v)$, i.e. $(\forall v \in W)$ $\mathfrak{N}, v \Vdash \neg\Box\varphi$, hence $\neg\Box\varphi \in Th(\mathfrak{N})$. ∎

5.2 REp-Stable Theories

We can alternatively consider the following rule for negative introspection:

$$\mathbf{NI_{c-p}}.\ \frac{\varphi \notin S \quad \text{and} \quad \neg\varphi \notin S}{\neg\Box\varphi \in S}$$

which 'says' that *if nothing is known to hold about φ, then it is known that φ is not known*.

Definition 7. *An RE-stable theory S is called* REp-stable *iff it is closed under* $\mathbf{NI_{c-p}}$.

This notion is stronger than the previous one and contains every instance of axiom $\mathbf{p5}$, introduced in [15].

Fact 6. *Every REp-stable theory is REw-stable.*

Lemma 4. *Every instance of axiomatic scheme* $\mathbf{p5}$ *(introduced and examined in [15]) is contained in any REp-stable theory.*

Furthermore, we can prove a representation theorem for REp-stable sets.

Theorem 7. *Let $S \subseteq \mathcal{L}_\Box$ be a consistent theory. S is REp-stable iff there is an n-model $\mathfrak{N} = \langle W, E, V\rangle$ s.t. $Th(\mathfrak{N}) = S$ and $(\forall w \in W)(E(w) = \{W\}$ or $E(w) = \{\varnothing, W\})$.*

PROOF. $(\Rightarrow)$ Since S is REp-stable, by Lem.4, S contains $\mathbf{p5}$, hence, since S is RE-stable and consistent, by Lem.1b), S is a $c\mathbf{p5}_C$-theory. So, model $\mathfrak{N}^{\mathbf{p5}_C,S}$ does exist. For simplicity, let us denote $\mathfrak{N}^{\mathbf{p5}_C,S}$ as $\mathfrak{N} = \langle W, E, V\rangle$. Then, by Prop.4 (for classical

logics and RE-proofs), $Th(\mathfrak{N}) = \{\varphi \in \mathcal{L}_{\Box} \mid S \vdash_{\mathbf{p5}_C} \varphi\}$. Consequently, by Lem.1a), $Th(\mathfrak{N}) = S$. Now, let $\mathsf{r} \in W$. Exactly as in Theor.5(1), one can prove that $W \in E(\mathsf{r})$. Consider now any $Y \in E(\mathsf{r})$ s.t. $Y \neq W$. Then, by Def.3(ii), there must be a $\varphi \in \mathcal{L}_{\Box}$ s.t. $Y = |\varphi|_{\mathbf{p5}_C,S}$ and $\Box\varphi \in \mathsf{r}$ $\quad (I)$ But, since $|\varphi|_{\mathbf{p5}_C,S} \subset W$, there is a $mSc\mathbf{p5}_C$-theory Δ s.t. $\Delta \notin |\varphi|_{\mathbf{p5}_C,S}$, hence, $\varphi \notin \Delta$, consequently, by Prop.2(iii), $\varphi \notin S$ $\quad (II)$ Suppose now, for the sake of contradiction, that $Y \neq \varnothing$. Then, there is a $\Xi \in Y$. Since $Y = |\varphi|_{\mathbf{p5}_C,S}$, $\varphi \in \Xi$, and since Ξ is consistent, $\neg\varphi \notin \Xi$, so, by Prop.2(iii), $\neg\varphi \notin S$ $\quad (III)$. Now, (II) and (III) imply by $\mathbf{NI_{c-p}}$, $\neg\Box\varphi \in S$, therefore, again by Prop.2(iii), $\neg\Box\varphi \in \mathsf{r}$, hence, by (I), r is inconsistent, which is a contradiction. So, $Y = \varnothing$.

($\Leftarrow$) (i) – (iii)-$\mathbf{RE_c}$ can be proved exactly as in Theor.5. So, let us prove property $\mathbf{NI_{c-p}}$. Let $\varphi \in \mathcal{L}_{\Box}$ s.t. $\varphi \notin Th(\mathfrak{N})$ and $\neg\varphi \notin Th(\mathfrak{N})$. Then, $\overline{V}(\varphi) \neq W$ and $\overline{V}(\varphi) \neq \varnothing$, hence, for any $w \in W$, since $E(w) = \{W\}$ or $E(w) = \{\varnothing, W\}$, $\overline{V}(\varphi) \notin E(w)$, consequently, $(\forall w \in W)\ \mathfrak{N}, w \Vdash \neg\Box\varphi$, hence $\neg\Box\varphi \in Th(\mathfrak{N})$. ∎

Theorem 7 and Fact 6 allow us to prove that REp-stable (and hence, REw-stable) theories do not suffer from the presence of all known epistemic axioms.

Proposition 8. *There is an REp-stable theory (which is also REw-stable), which does not contain* $\mathbf{K}$, $\mathbf{T}$, $\mathbf{4}$ *and* $\mathbf{5}$.

6 Related Work - Future Research

The notion of a stable belief set has been very useful in modal nonmonotonic reasoning. Investigations on stable sets have mainly focused on identifying their technical properties and representing them with the aid of model-theoretic constructions known from classical modal logic. It seems natural however to investigate, both from the logician's and the KR engineer's viewpoint, what can be obtained by loosening the conditions in the original definition of R. Stalnaker. To the best of our knowledge, it is the first time that notions of stable sets are investigated by varying the positive and negative introspection closure conditions. Up to now, there have been approaches which build belief sets by changing classical logic in condition (i) (replacing Modus Ponens by tautological entailment in [19], or adopting intuitionistic logic in [1]), or generalizing the notion of stability in a way somewhat related to the second question of our introduction [14].

The basic motivation of the research reported in our paper, is to define more plausible notions of an epistemic state and the ultimate goal is to employ these notions in new mechanisms for nonmonotonic modal logics, à la McDermott and Doyle. The latter goal is the first step in the roads of future research, along with the investigation on the assessment of epistemic states which emerge if we adopt even weaker notions of positive introspection, for instance by employing a context-dependent version of *Oscar Becker's rule* which has been employed in the study of modal systems which go some way towards solving the logical omniscience problem [6].

Acknowledgements. The authors wish to thank the three anonymous JELIA 2010 reviewers of the paper for raising important questions, providing valuable advice on the presentation of the results and pointing to interesting recent articles in the bibliography.

References

1. Amati, G., Carlucci Aiello, L., Pirri, F.: Intuitionistic autoepistemic logic. Studia Logica 59 (1997)
2. Aucher, G.: An internal version of epistemic logic. Studia Logica 94(1), 1–22 (2010)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
4. Chellas, B.F.: Modal Logic, an Introduction. Cambridge University Press, Cambridge (1980)
5. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press, Cambridge (2003)
6. Fitting, M.C.: Basic Modal Logic. In: Gabbay, et al. (eds.) [7], vol. 1, pp. 368–448 (1993)
7. Gabbay, D.M., Hogger, C.J., Robinson, J.A.: Handbook of Logic in Artificial Intelligence and Logic Programming. Oxford University Press, Oxford (1993)
8. Halpern, J.: A critical reexamination of default logic, autoepistemic logic and only-knowing. Computational Intelligence 13(1), 144–163 (1997)
9. Halpern, J.Y., Moses, Y.: A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence 54(2), 319–379 (1992)
10. Halpern, J.Y.: A theory of knowledge and ignorance for many agents. Journal of Logic and Computation 7(1), 79–108 (1997)
11. Hansen, H.H., Kupke, C., Pacuit, E.: Neighbourhood Structures: Bisimilarity and Basic Model Theory. Logical Methods in Computer Science 5(2:2), 1–38 (2009)
12. Hintikka, J.: Knowledge and Belief: an Introduction to the Logic of the two notions. Cornell University Press, Ithaca (1962)
13. Hughes, G.E., Cresswell, M.J.: A New Introduction to Modal Logic. Routledge (1996)
14. Jaspars, J.: A generalization of stability and its application to circumscription of positive introspective knowledge. In: Schönfeld, W., Börger, E., Kleine Büning, H., Richter, M.M. (eds.) CSL 1990. LNCS, vol. 533, pp. 289–299. Springer, Heidelberg (1991)
15. Koutras, C.D., Zikos, Y.: On a modal epistemic axiom emerging from McDermott-Doyle logics. Fundamenta Informaticae 96(1–2), 111–125 (2009)
16. Koutras, C. D., Zikos, Y.: Stable belief sets revisited, Technical Report, draft version (May 2010), available through the authors' web pages, in particular, http://users.att.sch.gr/zikos/index/logic/KZ-SBSr-extended.pdf
17. Lenzen, W.: Recent Work in Epistemic Logic. North-Holland, Amsterdam (1978)
18. Lenzen, W.: Epistemologische Betractungen zu [S4,S5]. Erkenntnis 14, 33–56 (1979)
19. Lakemeyer, G., Levesque, H.J.: A Tractable Knowledge Representation Service with Full Introspection. In: Vardi, M. (ed.) Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, TARK 1988, pp. 145–159. Morgan Kaufmann, San Francisco (1988)
20. Marek, V.W., Schwarz, G.F., Truszczyński, M.: Modal non-monotonic logics: Ranges, characterization, computation. Journal of the ACM 40, 963–990 (1993)
21. Marek, V.W., Truszczyński, M.: Non-Monotonic Logic: Context-dependent Reasoning. Springer, Heidelberg (1993)
22. McDermott, D., Doyle, J.: Non-monotonic logic I. Artificial Intelligence 13, 41–72 (1980)
23. Moore, R.C.: Semantical considerations on non-monotonic logics. Artificial Intelligence 25, 75–94 (1985)
24. Segerberg, K.: An essay in Clasical Modal Logic. Filosofiska Studies, Uppsala (1971)
25. Stalnaker, R.: A note on non-monotonic modal logic. Artificial Intelligence 64, 183–196 (1993), Revised version of the unpublished note originally circulated in 1980

Efficient Inferencing for OWL EL

Markus Krötzsch

Institute AIFB, Karlsruhe Institute of Technology, DE
mak@aifb.uni-karlsruhe.de

Abstract. We develop inferencing methods for $\mathcal{SROEL}(\sqcap, \times)$ – a DL that subsumes the main features of the W3C recommendation OWL EL –, and present a framework for studying materialisation calculi based on datalog. The latter is used to investigate the resource requirements for inferencing, and we can show that certain $\mathcal{SROEL}(\sqcap, \times)$ feature combinations must lead to increased space upper bounds in any materialisation calculus, suggesting that efficient implementations are easier to obtain for suitably chosen fragments of $\mathcal{SROEL}(\sqcap, \times)$.

1 Introduction

The recent OWL 2 W3C recommendation includes the lightweight ontology language OWL EL [9] which is semantically based on an extension of the $\mathcal{EL}^{++}$ description logic (DL). It is widely assumed that inferencing in OWL EL is possible in polynomial time, but it is not obvious how to extend existing reasoning procedures for $\mathcal{EL}^{++}$ accordingly [2]. In this paper, we set out to close this gap by developing suitable inferencing calculi for the DL $\mathcal{SROEL}(\sqcap, \times)$ which can be considered as an extension of the tractable DL $\mathcal{EL}^{++}$ with local reflexivity (Self), conjunctions of roles, and concept products. The latter two features generalise role disjointness, the universal (top) role, and admissible range restrictions as introduced in OWL EL. Concrete domains (datatypes) hardly interact with the additional features of $\mathcal{SROEL}(\sqcap, \times)$ and are not considered in this paper, though the according mechanisms used in [2] could be lifted to $\mathcal{SROEL}(\sqcap, \times)$.

Our second main contribution is to assess the *efficiency* of the proposed calculi. Inferencing for $\mathcal{EL}$-type DLs often suggests a materialisation-based (or consequence-driven) implementation, where all deductions are computed simultaneously in a bottom-up fashion. The number of inferable facts is an important measure of efficiency in this case, and we present a formalisation of materialisation calculi to relate it to the space complexity of datalog reasoning. Since upper space bounds for datalog are exponential in the *arity* of inferred predicates, our goal is to find materialisation calculi where these arities are low. We are able to show that there are limits to such optimisation: some inferencing tasks intrinsically require predicates of higher arities than others.

We present four inferencing calculi: a materialisation calculus for instance checking in $\mathcal{SROEL}(\sqcap, \times)$ in Section 3, and three calculi for classification in $\mathcal{SROEL}(\sqcap, \times)$ and two of its fragments in Section 4. Thereafter, in Section 5, we show that the arity of inferred predicates is minimal for each of the presented calculi. We provide extended sketches for some of the more interesting proofs to the extent that space permits. Detailed proofs for all results are found in the accompanying technical report [6].

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 234–246, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Table 1. Syntax and semantics of $\mathcal{SROEL}(\sqcap, \times)$ axioms

Axiom	Syntax	Semantics for an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
concept inclusion (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role inclusion	$R \sqsubseteq T$	$R^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
generalised role inclusion	$R \circ S \sqsubseteq T$	$\{\langle x, z \rangle \mid \langle x, y \rangle \in R^{\mathcal{I}}, \langle y, z \rangle \in S^{\mathcal{I}} \text{ for some } y\} \subseteq T^{\mathcal{I}}$
role conjunction	$S_1 \sqcap S_2 \sqsubseteq T$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
concept product	$C \times D \sqsubseteq T$	$C^{\mathcal{I}} \times D^{\mathcal{I}} \subseteq T^{\mathcal{I}}$
	$R \sqsubseteq C \times D$	$T^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times D^{\mathcal{I}}$
$C, D \in \mathbf{C}, R, S_{(i)}, T \in \mathbf{N_R}, a, b \in \mathbf{N_I}$		

2 Preliminaries

This section summarises the basic notions from DL and datalog that are used in this paper. Readers who are not familiar with these topics may find extended introductory definitions in [6]. The main DL studied herein is $\mathcal{SROEL}(\sqcap, \times)$ which subsumes all semantic features of OWL EL that are not related to datatypes (concrete domains). $\mathcal{SROEL}(\sqcap, \times)$ is based on three disjoint finite sets of *individual names* $\mathbf{N_I}$, *concept names* $\mathbf{N_C}$, and *role names* $\mathbf{N_R}$. The set $\mathbf{C}$ of $\mathcal{SROEL}(\sqcap, \times)$ *concept expressions* then is given as $\mathbf{C} ::= \top \mid \bot \mid \mathbf{N_C} \mid \mathbf{C} \sqcap \mathbf{C} \mid \exists \mathbf{N_R}.\mathbf{C} \mid \exists \mathbf{N_R}.\mathsf{Self} \mid \{\mathbf{N_I}\}$. The set of $\mathcal{SROEL}(\sqcap, \times)$ *axioms* is defined as in Table 1. One may distinguish axioms of *ABox* (assertional axioms), *TBox* (terminological axioms: GCIs), and *RBox* (axioms related to roles).

Knowledge bases are sets of axioms that satisfy some additional properties. Consider a set KB of $\mathcal{SROEL}(\sqcap, \times)$ axioms. We inductively define the set of *non-simple roles* of KB to contain all roles T for which there is an axiom $R \circ S \sqsubseteq T \in \text{KB}$, or an axiom $R \sqsubseteq T$ such that R is non-simple. A role that is not non-simple is called *simple*. Moreover, given a role name R, we define $\mathsf{ran}(R)$ to denote the set of concept expressions $D \in \mathbf{C}$ for which KB contains axioms $R \sqsubseteq S_1, \ldots, S_{n-1} \sqsubseteq S_n$ and $S_n \sqsubseteq C \times D$ for some $S_1, \ldots, S_n \in \mathbf{N_R}$ and $n \geq 0$. The set KB is a $\mathcal{SROEL}(\sqcap, \times)$ *knowledge base* if the following restrictions are satisfied:

- all roles S occurring in expressions $\exists S.\mathsf{Self} \in \text{KB}$ are simple,
- all roles S_1, S_2 occurring in axioms $S_1 \sqcap S_2 \sqsubseteq T \in \text{KB}$ are simple,
- for every axiom $R \circ S \sqsubseteq T \in \text{KB}$ we have $\mathsf{ran}(T) \subseteq \mathsf{ran}(S)$, and
- for every axiom $S_1 \sqcap S_2 \sqsubseteq T \in \text{KB}$ we have $\mathsf{ran}(T) \subseteq \mathsf{ran}(S_1) \cup \mathsf{ran}(S_2)$.

Note that we do not impose the structural restrictions of RBox regularity here [5] which also apply to OWL DL (and hence to OWL EL) ontologies, since these are not needed for efficient reasoning in $\mathcal{SROEL}(\sqcap, \times)$.

The semantics of $\mathcal{SROEL}(\sqcap, \times)$ is specified by defining DL interpretations $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ as usual. Here, we merely recall the semantics of axioms in Table 1; see [6] for a complete definition of $\mathcal{SROEL}(\sqcap, \times)$ semantics and entailment. Note that concept products on the left-hand side allow us to define the universal (top) role U with an axiom $\top \times \top \sqsubseteq U$. Since we can also define the empty (bottom) role N using $\exists N.\top \sqsubseteq \bot$,

conjunctions of (simple) roles are a generalisation of disjointness of (simple) roles: the axiom $R \sqcap S \sqsubseteq N$ declares S and R to be disjoint. In the absence of other role conjunctions, our requirements on concept products in $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases agree with the known admissibility requirements for range restrictions in $\mathcal{EL}^{++}$ [3].

Our formalisation of inferencing calculi is based on the simple rule language *datalog* [1]. A *signature* of datalog is a tuple $\langle \mathbf{C}, \mathbf{P} \rangle$, where $\mathbf{C}$ is a finite set of *constants*, and $\mathbf{P}$ is a finite set of *predicates*, and each predicate $p \in \mathbf{P}$ has a fixed arity $\mathsf{ar}(p) \geq 0$. We assume $\mathbf{P}$ to be a disjoint union $\mathbf{P}_i \cup \mathbf{P}_e$ of *IDB predicates* $\mathbf{P}_i$ and *EDB predicates* $\mathbf{P}_e$.[1] A countably infinite set of *variables* is denoted by $\mathbf{V}$. Elements of $\mathbf{C} \cup \mathbf{V}$ are called *terms*.

A *datalog atom* over a signature $\langle \mathbf{C}, \mathbf{P} \rangle$ is an expression $p(t_1, \ldots, t_n)$ where $p \in \mathbf{P}$ with $\mathsf{ar}(p) = n$, and $t_i \in \mathbf{C} \cup \mathbf{V}$ for $i = 1, \ldots, n$. An IDB (EDB) atom is one that uses an IDB (EDB) predicate. A *datalog rule* is a formula of the form $B_1 \wedge \ldots \wedge B_l \rightarrow H$ where B_i and H are datalog atoms, and H is an IDB atom. The premise of a rule is also called its *body*, and the conclusion is called its *head*. A *datalog program* P is a set of datalog rules. A *fact* is a ground, i.e. variable-free, rule with an empty body.

A *ground substitution* σ for a signature $\langle \mathbf{C}, \mathbf{P} \rangle$ is a function $\sigma : \mathbf{V} \rightarrow \mathbf{C}$. Substitutions are extended to datalog atoms by setting $\sigma(p(t_1, \ldots, t_n)) := p(\sigma(t_1), \ldots, \sigma(t_n))$, and $\sigma(p(t_1, \ldots, t_n))$ is called a *ground instance* of $p(t_1, \ldots, t_n)$ in this case.

A *proof tree* for a datalog program P is a structure $\langle N, E, \lambda \rangle$ where N is a finite set of nodes, $E \subseteq N \times N$ is a set of edges of a directed tree, and λ is a labelling function that assigns a ground datalog atom to each node, where the following holds: for each node $n \in N$, there is a rule $B_1 \wedge \ldots \wedge B_l \rightarrow H \in P$ and a ground substitution σ such that $\lambda(n) = \sigma(H)$ and the set of child nodes $\{m \mid \langle n, m \rangle \in E\}$ is of the form $\{m_1, \ldots, m_l\}$ where $\lambda(m_i) = \sigma(B_i)$ for each $i = 1, \ldots, l$.

A ground atom H is a *consequence* of a datalog program P if there is a proof tree for P that has H as the label $\lambda(r)$ of its root node r.

Definition 1. *Given a datalog signature* $\langle \mathbf{C}, \mathbf{P} \rangle$*, a* renaming ρ *is a function* $\rho : \mathbf{C} \rightarrow \mathbf{C}$. *To extend* ρ *to ground datalog atoms we set* $\rho(p(t_1, \ldots, t_n)) := p(\rho(t_1), \ldots, \rho(t_n))$.

3 Instance Checking for $\mathcal{SROEL}(\sqcap, \times)$

We now introduce a calculus for solving the inference task of instance checking – deciding if $C(a)$ is entailed for any $C \in \mathbf{N_C}$, $a \in \mathbf{N_I}$ – for $\mathcal{SROEL}(\sqcap, \times)$. In Section 5 we show its optimality in the sense that no other materialisation calculus can be better in terms of certain characteristics. To prepare this study of calculi, it makes sense to seek a uniform presentation for deduction calculi that have been proposed for $\mathcal{EL}$-type DLs, e.g., in [2,4]. This motivates our use of datalog in this section.

Intuitively speaking, a materialisation calculus is a system of deduction rules for deriving logical consequences which – as opposed to a complete inferencing algorithm – does not specify a control flow or processing strategy for evaluating these rules. Deduction rules can be denoted in many forms, e.g. using textual if-then descriptions [2], in

[1] This terminology originates from the field of deductive databases where one distinguishes *extensional* and *intensional data base*.

$C(a) \mapsto \{\texttt{subClass}(a, C)\}$	$R(a, b) \mapsto \{\texttt{subEx}(a, R, b, b)\}$	$a \in \mathbf{N_I} \mapsto \{\texttt{nom}(a)\}$
$\top \sqsubseteq C \mapsto \{\texttt{top}(C)\}$	$A \sqsubseteq \bot \mapsto \{\texttt{bot}(A)\}$	$A \in \mathbf{N_C} \mapsto \{\texttt{cls}(A)\}$
$\{a\} \sqsubseteq C \mapsto \{\texttt{subClass}(a, C)\}$	$A \sqsubseteq \{c\} \mapsto \{\texttt{subClass}(A, c)\}$	$R \in \mathbf{N_R} \mapsto \{\texttt{rol}(R)\}$
$A \sqsubseteq C \mapsto \{\texttt{subClass}(A, C)\}$	$A \sqcap B \sqsubseteq C \mapsto \{\texttt{subConj}(A, B, C)\}$	
$\exists R.\mathsf{Self} \sqsubseteq C \mapsto \{\texttt{subSelf}(R, C)\}$	$A \sqsubseteq \exists R.\mathsf{Self} \mapsto \{\texttt{supSelf}(A, R)\}$	
$\exists R.A \sqsubseteq C \mapsto \{\texttt{subEx}(R, A, C)\}$	$A \sqsubseteq \exists R.B \mapsto \{\texttt{supEx}(A, R, B, aux^{A \sqsubseteq \exists R.B})\}$	
$R \sqsubseteq T \mapsto \{\texttt{subRole}(R, T)\}$	$R \circ S \sqsubseteq T \mapsto \{\texttt{subRChain}(R, S, T)\}$	
$R \sqsubseteq C \times D \mapsto \{\texttt{supProd}(R, C, D)\}$	$A \times B \sqsubseteq R \mapsto \{\texttt{subProd}(A, B, R)\}$	
$R \sqcap S \sqsubseteq T \mapsto \{\texttt{subRConj}(R, S, T)\}$		
$A, B, C, D \in \mathbf{N_C}, R, S, T \in \mathbf{N_R}, a, b, c \in \mathbf{N_I}$		

Fig. 1. Input translation P_{inst}

tabular form [9], or as sequent calculus style derivation rules [4]. Premises and conclusions of rules often consist of logical formulae, but may also contain auxiliary expressions that are relevant to the calculus.[2] A deduction rule can then be viewed as a schema for deriving new expressions from a finite set of given expressions. In particular, the applicability of rules is normally not affected by uniform renamings of signature symbols in premise and conclusion.

Deduction rules in this sense can be denoted as datalog rules where concrete logical sentences are represented as ground facts that use signature symbols in term positions. For example, we can represent $A \sqsubseteq B$ as $\texttt{subclass0f}(A, B)$, and introduce a rule $\texttt{subclass0f}(x, y) \wedge \texttt{subclass0f}(y, z) \rightarrow \texttt{subclass0f}(x, z)$. This unifies the presentation of diverse calculi, and allows us to exploit techniques from deductive databases. For connecting datalog to DL, we require an input translation from individual DL axioms to (sets of) datalog EDB facts. This translations is also defined for signature symbols, since symbols must generally be "loaded" into datalog to be able to derive conclusions about them, regardless of whether the symbols occurred in input axioms or not. A formalisation of these ideas is given later in Definition 2.

Calculi in the above sense generally suggest materialisation-based (or consequence-driven) reasoning: after translating a knowledge base to datalog facts, all consequences of these facts under the deduction rules can be computed in a bottom-up fashion, and all supported entailments can then be checked without further recursive computation. This contrasts with other reasoning principles such as the tableaux method where just a single entailment is checked in one run of the algorithm.

It is not hard to formulate the deduction algorithms presented for $\mathcal{EL}$-type logics in [2] and [4] using datalog rules. The calculus we present here, however, is derived from a datalog reduction introduced in [8] for a rule language based on $\mathcal{EL}^{++}$. This approach can be modified to cover $\mathcal{SROEL}(\sqcap, \times)$ and to use a fixed set of datalog rules to yield a materialisation calculus in our sense. For simplicity, the following calculus only considers $\mathcal{SROEL}(\sqcap, \times)$ axioms of the basic forms in Fig. 1. $\mathcal{SROEL}(\sqcap, \times)$ axioms can be translated to such normalised axioms in linear time so that all entailments of the input knowledge base are preserved [6].

[2] For instance, the calculus in [2] uses auxiliary statements $A \rightsquigarrow_R B$ for $A, B \in \mathbf{N_C}, R \in \mathbf{N_R}$.

(1)	$\texttt{nom}(x) \rightarrow \texttt{inst}(x, x)$
(2)	$\texttt{nom}(x) \wedge \texttt{triple}(x, v, x) \rightarrow \texttt{self}(x, v)$
(3)	$\texttt{top}(z) \wedge \texttt{inst}(x, z') \rightarrow \texttt{inst}(x, z)$
(4)	$\texttt{bot}(z) \wedge \texttt{inst}(u, z) \wedge \texttt{inst}(x, z') \wedge \texttt{cls}(y) \rightarrow \texttt{inst}(x, y)$
(5)	$\texttt{subClass}(y, z) \wedge \texttt{inst}(x, y) \rightarrow \texttt{inst}(x, z)$
(6)	$\texttt{subConj}(y_1, y_2, z) \wedge \texttt{inst}(x, y_1) \wedge \texttt{inst}(x, y_2) \rightarrow \texttt{inst}(x, z)$
(7)	$\texttt{subEx}(v, y, z) \wedge \texttt{triple}(x, v, x') \wedge \texttt{inst}(x', y) \rightarrow \texttt{inst}(x, z)$
(8)	$\texttt{subEx}(v, y, z) \wedge \texttt{self}(x, v) \wedge \texttt{inst}(x, y) \rightarrow \texttt{inst}(x, z)$
(9)	$\texttt{supEx}(y, v, z, x') \wedge \texttt{inst}(x, y) \rightarrow \texttt{triple}(x, v, x')$
(10)	$\texttt{supEx}(y, v, z, x') \wedge \texttt{inst}(x, y) \rightarrow \texttt{inst}(x', z)$
(11)	$\texttt{subSelf}(v, z) \wedge \texttt{self}(x, v) \rightarrow \texttt{inst}(x, z)$
(12)	$\texttt{supSelf}(y, v) \wedge \texttt{inst}(x, y) \rightarrow \texttt{self}(x, v)$
(13)	$\texttt{subRole}(v, w) \wedge \texttt{triple}(x, v, x') \rightarrow \texttt{triple}(x, w, x')$
(14)	$\texttt{subRole}(v, w) \wedge \texttt{self}(x, v) \rightarrow \texttt{self}(x, w)$
(15)	$\texttt{subRChain}(u, v, w) \wedge \texttt{triple}(x, u, x') \wedge \texttt{triple}(x', v, x'') \rightarrow \texttt{triple}(x, w, x'')$
(16)	$\texttt{subRChain}(u, v, w) \wedge \texttt{self}(x, u) \wedge \texttt{triple}(x, v, x') \rightarrow \texttt{triple}(x, w, x')$
(17)	$\texttt{subRChain}(u, v, w) \wedge \texttt{triple}(x, u, x') \wedge \texttt{self}(x', v) \rightarrow \texttt{triple}(x, w, x')$
(18)	$\texttt{subRChain}(u, v, w) \wedge \texttt{self}(x, u) \wedge \texttt{self}(x, v) \rightarrow \texttt{triple}(x, w, x)$
(19)	$\texttt{subRConj}(v_1, v_2, w) \wedge \texttt{triple}(x, v_1, x') \wedge \texttt{triple}(x, v_2, x') \rightarrow \texttt{triple}(x, w, x')$
(20)	$\texttt{subRConj}(v_1, v_2, w) \wedge \texttt{self}(x, v_1) \wedge \texttt{self}(x, v_2) \rightarrow \texttt{self}(x, w)$
(21)	$\texttt{subProd}(y_1, y_2, w) \wedge \texttt{inst}(x, y_1) \wedge \texttt{inst}(x', y_2) \rightarrow \texttt{triple}(x, w, x')$
(22)	$\texttt{subProd}(y_1, y_2, w) \wedge \texttt{inst}(x, y_1) \wedge \texttt{inst}(x, y_2) \rightarrow \texttt{self}(x, w)$
(23)	$\texttt{supProd}(v, z_1, z_2) \wedge \texttt{triple}(x, v, x') \rightarrow \texttt{inst}(x, z_1)$
(24)	$\texttt{supProd}(v, z_1, z_2) \wedge \texttt{self}(x, v) \rightarrow \texttt{inst}(x, z_1)$
(25)	$\texttt{supProd}(v, z_1, z_2) \wedge \texttt{triple}(x, v, x') \rightarrow \texttt{inst}(x', z_2)$
(26)	$\texttt{supProd}(v, z_1, z_2) \wedge \texttt{self}(x, v) \rightarrow \texttt{inst}(x, z_2)$
(27)	$\texttt{inst}(x, y) \wedge \texttt{nom}(y) \wedge \texttt{inst}(x, z) \rightarrow \texttt{inst}(y, z)$
(28)	$\texttt{inst}(x, y) \wedge \texttt{nom}(y) \wedge \texttt{inst}(y, z) \rightarrow \texttt{inst}(x, z)$
(29)	$\texttt{inst}(x, y) \wedge \texttt{nom}(y) \wedge \texttt{triple}(z, u, x) \rightarrow \texttt{triple}(z, u, y)$

Fig. 2. Deduction rules P_{inst}

Theorem 1. *Consider the materialisation calculus K_{inst} with input translation I_{inst} as in Fig. 1, and derivation rules P_{inst} as in Fig. 2. For a knowledge base* KB *such that $I_{inst}(\alpha)$ is defined for all $\alpha \in$ KB, set $P(\text{KB}) := P_{inst} \cup \bigcup_{\alpha \in \text{KB}} I_{inst}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{inst}(s)$.*

For all $C \in \mathbf{N_C}$, and $a \in \mathbf{N_I}$, KB *entails $C(a)$ if and only if $P(\text{KB})$ entails $\texttt{inst}(a, C)$, whenever $P(\text{KB})$ is defined. Thus K_{inst} provides a materialisation calculus for instance checking for $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases within which all axioms are normalised.*

The IDB predicates `inst`, `triple`, and `self` in P_{inst} correspond to ABox axioms for atomic concepts, roles, and concepts $\exists R.\text{Self}$, respectively. Rule (1) serves as an initialisation rule that accounts for the first `inst` facts to be derived. Rule (2) specifies the (only) case where reflexive `triple` facts lead to `self` facts. The rules (3) to (26) capture expected derivations for each of the axiom types as encoded by the EDB predicates. Rule (4) checks for global inconsistencies, and would typically not be materialised in implementations since its effect can directly be taken into account during entailment checking. Rules (9) and (10) make use of auxiliary constants $aux^{A \sqsubseteq \exists R.B}$ for handling existentials. Roughly speaking, each such constant represents the class of all

role successors generated by the axiom from which it originates; see [6] for details. The remaining rules (27) to (29) encode equality reasoning that is relevant in the presence of nominals where statements $\texttt{inst}(a, b)$ with $a, b \in \mathbf{N_I}$ encode equality of a and b.

Axiom normalisation and the computation of I_{inst} can be accomplished in linear time, and the time for reasoning in datalog is polynomial w.r.t. the size of the collection of ground facts. Together with the known P-hardness of $\mathcal{EL}^{++}$ [2], we obtain the following result, of which no formal proof seems to have been published so far:

Corollary 1. *Instance checking in $\mathcal{SROEL}(\sqcap, \times)$ and in OWL EL without datatype properties is* P *complete w.r.t. the size of the knowledge base.*

This result can be extended to OWL EL with datatype properties along the lines of datatype reasoning in $\mathcal{EL}^{++}$ [2], but this is not implied by the above theorem. The proof of Theorem 1 is found in [6]. Completeness is obtained by transforming models of datalog programs to corresponding models of DL knowledge bases, part of which is to show that equality reasoning really suffices to establish a congruence between elements of the domain. Soundness is shown by interpreting the meaning of datalog atoms in terms of DL, and showing inductively that each rule application preserves soundness of this interpretation. This is most interesting for rules (19) and (25) where the result hinges upon the restrictions on role conjunction and concept products in $\mathcal{SROEL}(\sqcap, \times)$.

4 Classification of $\mathcal{SROEL}(\sqcap, \times)$ Knowledge Bases

The materialisation calculus K_{inst} of Theorem 1 solves the instance checking problem for $\mathcal{SROEL}(\sqcap, \times)$. A calculus for checking satisfiability is easily derived since a $\mathcal{SROEL}(\sqcap, \times)$ knowledge base is inconsistent if and only if K_{inst} infers a fact $\texttt{inst}(x, z)$ where $\texttt{bot}(z)$ holds. In this section, we ask how to obtain calculi for *classification* – the computation of all subsumptions of atomic classes implied by a knowledge base.

Class subsumption, too, can be reduced to instance retrieval: to check $A \sqsubseteq B$, one introduces a new individual c and adds an assertion $A(c)$; then the subsumption holds if the modified knowledge base entails $B(c)$. This reduction requires the knowledge base to be modified, leading to new entailments, possibly even to global inconsistency. Thus K_{inst} cannot directly be used for classification, since it is not feasible to introduce test individuals c for all (atomic) classes at load time so as to materialise all subsumptions in parallel. Rather, one would have to use a separate run of K_{inst} for each subclass A to compute all entailments of the form $A \sqsubseteq B$.

This approach allows us to derive a sound and complete materialisation calculus for materialisation in $\mathcal{SROEL}(\sqcap, \times)$ by "internalising" the runs of K_{inst} by extending all IDB predicates with an additional parameter to encode the test assumption under which this fact can be inferred. Our assumptions have the form $A(c)$, but the name of c is not essential. So we re-use the datalog constant A as the test instance of class A, such that the additional parameter of IDB atoms can simply be a concept name A. The proof of the following theorem is immediate from this discussion.

Theorem 2. *Consider the materialisation calculus K_{sc} with input translation I_{sc} defined like I_{inst} in Fig. 1 and datalog program P_{sc} containing the following rules:*

- *for each rule* $r \in P_{inst}$ *(Fig. 2), a rule* r' *obtained from* r *by adding a new body atom* $\texttt{cls}(q)$*, and replacing each IDB atom* $\texttt{inst}(x, y)$ *(*$\texttt{triple}(x, y, z)$*,* $\texttt{self}(x, y)$*) by an atom* $\texttt{inst\_sc}(x, y, q)$ *(*$\texttt{triple\_sc}(x, y, z, q)$*,* $\texttt{self\_sc}(x, y, q)$*), where* q *is a variable not occurring in* r*,*
- *the additional rule* $\texttt{cls}(q) \rightarrow \texttt{inst\_sc}(q, q, q)$*.*

For a knowledge base KB *such that* $I_{sc}(\alpha)$ *is defined for all* $\alpha \in$ KB*, set* $P(\text{KB}) := P_{sc} \cup \bigcup_{\alpha \in \text{KB}} I_{sc}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{sc}(s)$*. Then for all* $A, B \in \mathbf{N_C}$*,* KB *entails* $A \sqsubseteq B$ *if and only if* $P(\text{KB})$ *entails* $\texttt{inst\_sc}(A, B, A)$*, whenever* $P(\text{KB})$ *is defined. Thus* K_{sc} *provides a materialisation calculus for subsumption checking for* $\mathcal{SROEL}(\sqcap, \times)$ *knowledge bases within which all axioms are normalised.*

It must be noted that K_{sc} is not very efficient since deductions that are globally true are inferred under each local assumption q independently. This means that the number of globally derived facts can multiply by the number of class names in the signature, e.g. by more than 300,000 for the popular SNOMED CT ontology. Our formalisation of materialisation calculi provides a direct measure of this increase: the maximal arity of IDB predicates in K_{sc} is four while it had been three in K_{inst}, leading to potentially higher space requirements for materialised derivations. Implementations may of course achieve lower space bounds by using suitable optimisations; yet standard implementation techniques for datalog, such as semi-naive materialisation, are sensitive to the number of parameters in IDB predicates. In developing the database-driven reasoner *Orel* [7], we also experienced major *runtime* penalties associated with higher arities due to the larger numbers of inferences that must be considered in each derivation step.

The arity of IDB predicates thus is an important measure for the efficiency of a materialisation calculus, and we will denote this parameter as the *arity of a calculus* and speak of binary/ternary/n-ary materialisation calculi. The search for more efficient materialisation calculi can now be formalised as the task of finding a ternary or binary calculus that is sound and complete for $\mathcal{SROEL}(\sqcap, \times)$ classification. Unfortunately, as shown in Section 5, such a calculus cannot exist. To illustrate that this is not obvious, we now present a classification calculus of lower arity for a fragment of $\mathcal{SROEL}(\sqcap, \times)$.

We now develop a ternary materialisation calculus that supports role chains but no $\top$, $\bot$, nominal classes, and concept products on the left-hand side of axioms. The input translation can remain as in Fig. 1 but without the cases that involve the excluded features. The EDB predicates `top`, `bot`, and `subProd` are no longer used.

A set of rules is developed by restricting the rules of K_{sc} of Theorem 2. We use the numbers as in Fig. 2 for referring to the rules obtained from K_{inst}. Rules (3), (4), (21), and (22) are no longer needed due to the restriction of EDB predicates. Without nominal classes, we find that all derivations $\texttt{inst\_sc}(x, y, q)$ are such that y is a DL class name, or y is a DL individual name and $x = y$. This is not hard to verify inductively by considering each rule, and the symbols used in relevant EDB facts. This shows that rules (27), (28), and (29) are obsolete as well. As shown in [6], the essential feature of the remaining rule set is that the additional parameter q that has been introduced for K_{sc} above is no longer required for obtaining a sound and complete materialisation calculus.

Theorem 3. *Consider the materialisation calculus* K_{scc} *with* I_{scc} *defined like* I_{inst} *in Fig. 1 but undefined for all axioms that use nominal classes,* $\top$*,* $\bot$*, or concept products*

on the left-hand side, and the program P_{scc} consisting of the rules (1), (2), (5)–(20), and (23)–(26) of Fig. 2 together with a new rule $\texttt{cls}(z) \rightarrow \texttt{inst}(z, z)$.

For a knowledge base KB *such that $I_{scc}(\alpha)$ is defined for all $\alpha \in$ KB, set* $P(\text{KB}) := P_{scc} \cup \bigcup_{\alpha \in \text{KB}} I_{scc}(\alpha) \cup \bigcup_{s \in \mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}} I_{scc}(s)$. *Then for all $A, B \in \mathbf{N_C}$,* KB *entails $A \sqsubseteq B$ if and only if* $P(\text{KB})$ *entails* $\texttt{inst}(A, B)$, *whenever* $P(\text{KB})$ *is defined. Thus K_{scc} provides a materialisation calculus for subsumption checking for $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases that contain only $\sqcap$ (for concepts and roles), $\exists$,* Self, $\circ$, *and concept products on the right-hand side.*

In terms of OWL 2, the DL of the previous theorem covers all OWL EL ontologies without datatype properties and the constructs `owl:Thing`, `owl:topObjectProperty`, `owl:Nothing`, `owl:bottomObjectProperty`, `objectHasValue` and `objectOneOf`.

It is not hard to further simplify K_{scc} for the case that no role chains occur in the knowledge base, leading to a binary classification calculus for normalised $\mathcal{SROEL}(\sqcap, \times)$ knowledge bases that contain only $\sqcap$ (for concepts and roles), $\exists$, Self, and concept products on the right-hand side. For reasons of space, the calculus has been removed from the final version of this paper; it can still be found in [6]. A similar approach was used to optimise a classification calculus for $\mathcal{ELH}$ presented in [4].

5 Minimal Arities of Materialisation Calculi

The previously discussed materialisation calculi for $\mathcal{SROEL}(\sqcap, \times)$ featured different arities: while some reasoning tasks could be solved by binary and ternary calculi, our classification calculus for $\mathcal{SROEL}(\sqcap, \times)$ is 4-ary. We have argued above that lower arities are important for efficient processing, so it is desirable to develop materialisation calculi of minimal arity. In this section, we establish lower bounds on the arity of materialisation calculi for various reasoning problems. This requires a concrete understanding of what a materialisation calculus is. Generalising the properties of the calculi discussed above, we obtain the following formalisation of this notion.

Definition 2. *A* materialisation calculus *K is a tuple $K = \langle I, P, O \rangle$ where I and O are partial functions, and P is a set of datalog rules, such that*

1. *given an axiom or signature symbol α, $I(\alpha)$ is either undefined or a set of datalog facts over EDB predicates,*
2. *given an axiom α, $O(\alpha)$ is either undefined or a datalog fact over an IDB predicate,*
3. *the set of EDB and IDB predicates used by I, P, and O is fixed and finite,*
4. *P contains no constant symbols,*
5. *all constant symbols used in $I(\alpha)$ or $O(\alpha)$ for some axiom (or signature symbol) α are either signature symbols that appear in (or are equal to) α, or constants of the form aux_i^α with $i \geq 0$, where all constant names aux_i^α are mutually distinct and unequal to any DL signature symbol,*
6. *I and O do not depend on concrete signature symbols, i.e. for a renaming ρ of signature symbols that maps individual/concept/role names to individual/concept/role names, we find $I(\rho(\alpha)) = \rho(I(\alpha))$ and $O(\rho(\alpha)) = \rho(O(\alpha))$ if $\rho(aux_i^\alpha) = aux_i^{\rho(\alpha)}$.*

We extend I to knowledge bases KB *by setting* $I(\mathrm{KB}) := \bigcup_{\beta\in\mathrm{KB}} I(\beta)$ *if* $I(\beta)$ *is defined for all* $\beta \in \mathrm{KB}$ *and undefined otherwise. We extend I to sets of signature symbols S by setting* $I(S) := \bigcup_{s\in S, I(s)\ defined} I(s)$. *$K$ induces an* entailment relation $\vdash_K$ *between knowledge bases* KB *and axioms α over a signature* $\langle \mathbf{N_I}, \mathbf{N_C}, \mathbf{N_R}\rangle$, *defined by setting* $\mathrm{KB} \vdash_K \alpha$ *whenever* $I(\mathrm{KB})$ *and* $O(\alpha)$ *are defined and* $I(\mathrm{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P \models O(\alpha)$.

We say that K is sound (complete) *if* $\mathrm{KB} \vdash_K \alpha$ *implies (is implied by)* $\mathrm{KB} \models \alpha$ *for all knowledge bases* KB *and axioms α for which* $I(\mathrm{KB})$ *and* $O(\alpha)$ *are defined.*

Note that this definition explicitly allows the datalog transformation I to introduce arbitrarily many auxiliary constants aux_i^α. This can be utilised, e.g., to perform a normalisation that introduces auxiliary concept names as part of the input translation, or to introduce new constants for handling existentials as in the above calculi. Yet, the input translation is limited in its expressivity, since it depends only on individual axioms and signature symbols. In particular, this precludes complex datalog translations as in [10,11]. Note that we do not make any assumptions on the computability or complexity of I and O, but both functions are typically very simple.

Now our general proof strategy is as follows. For a contradiction, we suppose that there is a materialisation calculus of lower arity that solves a given reasoning problem. We then consider a particular instance of that problem, given by a knowledge base KB from which a relevant consequence α must follow. Since the calculus is assumed to be complete, we obtain an according datalog derivation with a corresponding proof tree. This proof tree is then modified by renaming constants, leading to a variant of the proof tree that is still valid for the given materialisation calculus, but that is based on different (renamed) assumptions. The modified assumptions correspond to a modified knowledge base KB′, and by our construction we find that the materialisation calculus still computes the entailment of α on the input KB′. We then show that α is not entailed by KB′, so that the calculus is proven to be unsound. Since KB′ is based on the modified proof tree, some graph theoretic arguments are required to establish this last step.

A central notion of this proof strategy is the following modification of proof trees.

Definition 3. *Consider a materialisation calculus $K = \langle I, P, O\rangle$ and a knowledge base* KB *such that* $I(\mathrm{KB})$ *is defined, and a proof tree* $T = \langle N, E, \lambda\rangle$ *for* $I(\mathrm{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$. *We say that a DL signature symbol σ* occurs *in a ground atom F if F contains σ as a constant, or if F contains some auxiliary constant aux_i^α such that σ occurs in α. The* interface *of a node $n \in N$ is the set of signature symbols that occur in $\lambda(n)$.*

The (labels of) T can be diversified *by the following recursive construction:*

- *replace all signature symbols s that do not occur in the interface of the root node by a fresh symbol s' that has not yet been used in T or in this construction,*
- *recursively diversify the subtrees below each of the direct child nodes of the root.*

We tacitly assume that the datalog signature contains all required new constant names. Note that the renaming may affect auxiliary constants by renaming symbols in the axioms that are part of their name. The diversification is thus obtained by replacing some signature symbols with fresh symbols. This replacement may not be uniform throughout the tree, and we use s^n to denote the symbol by which s is replaced in node n.

Intuitively speaking, the above renaming removes any re-use of constant names throughout the proof tree that is not strictly necessary for applying the rules of P.

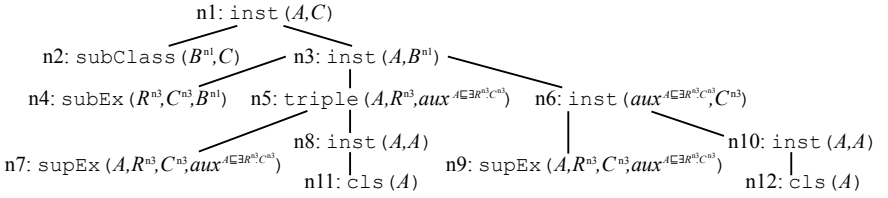


Fig. 3. Diversification of a K_{scc} proof for $\{A \sqsubseteq \exists R.C, \exists R.C \sqsubseteq B, B \sqsubseteq C\} \models A \sqsubseteq C$

What is "strictly necessary" is captured by the *interface* of each node: constants that are not in the interface of a rule application can be renamed uniformly in all descendants of the current node without affecting the correctness of the proof tree. This creates directly connects the arity of a calculus to the amount of renaming during diversification.

Figure 3 shows an example diversification based on the calculus K_{scc} of Theorem 3, where we use the notation from Definition 3 for denoting renamed symbols. Note how C is renamed to C^{n3} in some but not in all labels. Also note that no further renamings occur below the nodes $n5$ and $n6$ since all relevant symbols occur in their interface due to the auxiliary constant. As expected, the diversification is again a proof tree for a knowledge base that contains suitably renamed axioms:

Definition 4. *Consider a materialisation calculus K, knowledge base* KB*, and proof tree T as in Definition 3. Let λ' denote a diversified labelling for T.*

For each leaf node $m \in N$, there is some $\alpha \in$ KB with $\lambda(m) \in I(\alpha)$. By Definition 2, one can rename symbols in α to obtain an axiom α' such that $\lambda'(m) \in I(\alpha')$. Concretely, α' is obtained from α be replacing all symbols s in the interface of m by s^m, and by replacing all other symbols t by some fresh symbol t' not used anywhere yet. We select one such axiom α'_m for each leaf node.

The diversification KB′ *of* KB *is the knowledge base* $\text{KB}' := \{\alpha'_n \mid n \in N, n \text{ a leaf}\}$. *The tree structure of T can be used to represent* KB′ *as a set of nested sets Γ_n for $n \in N$, recursively defined by setting $\Gamma_n := \{\alpha'_m \mid \langle n, m\rangle \in E, m \text{ a leaf}\} \cup \{\Gamma_m \mid \langle n, m\rangle \in E, m \text{ not a leaf}\}$. We say that an axiom or set is* below *a set Γ_n if it is either an element of Γ_n, or if it is (recursively) below some element of Γ_n.*

For Fig. 3, the diversified knowledge base is $\{A \sqsubseteq \exists R^{n3}.C^{n3}, \exists R^{n3}.C^{n3} \sqsubseteq B^{n1}, B^{n1} \sqsubseteq C\}$ and we have $\Gamma_{n1} = \{B^{n1} \sqsubseteq C, \{\exists R^{n3}.C^{n3} \sqsubseteq B^{n1}, \{A \sqsubseteq \exists R^{n3}.C^{n3}\}\}\}$. Since the underlying calculus is correct, the conclusion still follows from the diversified knowledge base, and the diversified proof tree is still correct. Below we use diversification to construct proof trees with invalid conclusions for calculi with insufficient arities.

To this end, note that if l is the maximal number of premises in rules of K, then each set Γ_n has at most l elements (axioms α'_m for leaf children, sets Γ_m for non-leaf children). Moreover, if $\Gamma_m \in \Gamma_n$, then the DL signature symbols that occur in axioms below Γ_m either belong to the interface of n, or occur only in axioms of KB′ that are below Γ_m. The interface includes all DL symbols that occur in the ground IDB atom that is derived at a certain node of the proof tree, so the use of auxiliary constants can require the inclusion of *all* symbols of a given input axiom into the interface. Yet, the

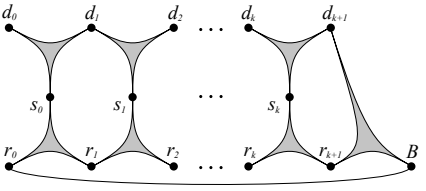


Fig. 4. Dependency graph for the proof of Theorem 4

arity clearly limits the number of axioms for which this may be the case: for a calculus of arity a, the interface of any node can comprise no more than the set of DL symbols that occur in a axioms of the input knowledge base.

These observations can also be interpreted graphically based on the *dependency graph* of KB′ – the graph that has the signature symbols in KB′ as its nodes, and, for each axiom of KB′ with exactly n signature symbols, an n-ary hyperedge connecting these n symbols. The sets of axioms Γ_n can be viewed as subgraphs of a dependency graph, where the interface of the node n describes the nodes that this subgraph is allowed to share with the remaining graph. These insights allow us to provide a proof sketch for our first minimality result; see [6] for details on each step of the argument.

Theorem 4. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, and role chains. Every materialisation calculus that is sound and complete for classification or instance retrieval in $\mathcal{L}$ has arity three or more.*

Proof. To obtain the result for classification, suppose that there is a binary materialisation calculus $K = \langle I, P, O\rangle$ for classification in $\mathcal{L}$. Let KB contain the following axioms:

$$D_i \sqsubseteq \exists S_i.D_{i+1}, \quad S_i \circ R_{i+1} \sqsubseteq R_i, \quad D_{k+1} \sqsubseteq \exists R_{k+1}.B, \quad \exists R_0.B \sqsubseteq B,$$

for all $i \in \{0, \ldots, k\}$, where $k > 2(l+1)$ for l the maximal number of body atoms in rules of P. Then KB entails $D_0 \sqsubseteq B$. Thus there is a proof tree T for deriving $O(D_0 \sqsubseteq B)$ for the program $I(\mathrm{KB}) \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$. Let $T' = \langle N, E, \lambda'\rangle$ be the diversified proof tree obtained from T by using renamed symbols s^n as in Definition 3, and let KB′ be the according diversified knowledge base. One can now construct a model $\mathcal{I}$ of KB′ in such a way that $\mathcal{I} \models D_0 \sqsubseteq B$ can hold only if KB′ contains axioms of the form:

$$d_0 \sqsubseteq s_0.d_1, \ldots, d_k \sqsubseteq s_k.d_{k+1}, \quad s_0 \circ r_1 \sqsubseteq r_0, \ldots, s_k \circ r_{k+1} \sqsubseteq r_k, \quad d_{k+1} \sqsubseteq B, \ \exists r_0.B \sqsubseteq B,$$

where $d_0 = D_0$, $d_i = D_i^o$ for some $o \in N$, $s_i = S_i^o$ for some $o \in N$, and $r_i = R_i^o$ for some $o \in N$. We claim that this is impossible. For a contradiction, suppose KB′ contains a set of axioms KB″ of this form. The axioms of KB″ are distributed over sets $(\Gamma_o)_{o\in N}$ as in Definition 4. Since T' has an out-degree of at most l (as specified above), our choice of k implies that T' contains a node $o \in N$ such that Γ_o has three axioms of the form $d_i \sqsubseteq \exists s_i.d_{i+1}$ below it, and such that three other axioms of this form are not below it.

The axioms below Γ_o induce a subgraph of the dependency graph of KB″ as shown in Fig. 4. As discussed above, this subgraph may share at most two nodes with the rest of the graph since K has arity two. Now it is not hard to argue that such a subgraph cannot exist. Hence Γ_o cannot exist, and KB″ cannot be contained in KB′. So $\mathcal{I}$ does

not satisfy $D_0 \sqsubseteq B$, and thus the latter is not a consequence of KB′. As T' is a proof tree for $I(\text{KB}') \cup I(\mathbf{N_I} \cup \mathbf{N_C} \cup \mathbf{N_R}) \cup P$, K derives $D_0 \sqsubseteq B$. So K cannot be sound, contradicting our assumption of its existence.

The result for instance retrieval is obtained by extending KB with an axiom $D_0(a)$, and using an analogous argument to show that $B(a)$ is not entailed by any diversification of this knowledge base on a materialisation calculus of arity 2. □

Analogous proofs can be given to obtain results for DLs that include nominals:

Theorem 5. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, and nominal classes. Every materialisation calculus that is sound and complete for classification in $\mathcal{L}$ has arity three or more.*

Theorem 6. *Let $\mathcal{L}$ be a DL with GCIs, existential quantification, role chains, and nominal classes. Every materialisation calculus that is sound and complete for classification in $\mathcal{L}$ has arity four or more.*

These results do not extend to instance retrieval, so in a sense classification is harder to implement efficiently. Indeed, Theorem 1 shows that a ternary instance retrieval calculus exists for a DL that includes existentials, nominals, and role chains. For DLs as in Theorem 5, we have not presented calculi of optimal arity. A ternary (binary) calculus for classification (instance retrieval) in this case can be obtained by eliminating the `triple_sc(triple)` predicate from K_{sc} (K_{inst}) as done for the binary calculus K_{sc} presented in [6]. Theorem 6 may be surprising, given that the calculus proposed in [2] for $\mathcal{EL}^{++}$ would be ternary in our notation. The explanation is that this algorithm is incomplete for classification; the proof of Theorem 6 can be used to find a suitable counter example [6].

6 Summary and Conclusions

The focus of this work has been the study of inferencing calculi for $\mathcal{SROEL}(\sqcap, \times)$ and its fragments, and especially this paper is – to the best of our knowledge – the first to present a sound and complete polynomial time calculus for inferencing in a DL that is so closely related to the OWL EL ontology language. For investigating properties of such calculi, we presented a simple framework for expressing materialisation calculi in terms of datalog. This revealed the arity of IDB predicates as an interesting measure for the worst-case space requirements of materialisation-based algorithms. While $\mathcal{SROEL}(\sqcap, \times)$ fragments without role chains and nominals admit classification calculi based on binary IDB predicates, the inclusion of either feature increases the required arity by one. Having both features, $\mathcal{SROEL}(\sqcap, \times)$ thus does not admit any sound and complete classification calculus of arity below four.

We are thus able to differentiate various $\mathcal{SROEL}(\sqcap, \times)$ fragments and inferencing tasks based on a measure that relates to the efficiency of actual implementations. Indeed, our findings agree with practical experiences that especially nominals and role chains are harder to implement efficiently than basic $\mathcal{EL}$ features.[3] Computational complexity

[3] Based on the author's experience implementing Orel [7], and personal communication with developers of DB [4] and CEL (`http://lat.inf.tu-dresden.de/systems/cel/`)

has not been able to provide an explanation for such discrepancies, since all reasoning problems we consider are P-complete. In addition, our study also shows that various other features are not harder to implement than some of the most basic ones, thus providing guidance for deciding which features to implement or to use in an application.

Although there are standard implementation strategies for datalog reasoning, our study is independent of actual algorithms. A promising next step thus is to develop control strategies for implementing our calculi in a "pay-as-you-go" algorithm that minimises the potential negative impact of the occurrence of certain features. Moreover, we conjecture that our results about datalog arity can be further strengthened to obtain more direct statements about space complexity of almost arbitrary monotone calculi.

Acknowledgements. The author thanks Yevgeny Kazakov for his valuable input, and the anonymous reviewers for helpful comments. This work was supported by DFG in project *ExpreSST* and by EPSRC in project *ConDOR* (EP/G02085X/1).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1994)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L., Saffiotti, A. (eds.) Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), pp. 364–369 (2005), Professional Book Center
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: Clark, K.G., Patel-Schneider, P.F. (eds.) Proc. OWLED 2008 DC Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 496 (2008), CEUR-WS.org
4. Delaitre, V., Kazakov, Y.: Classifying $\mathcal{ELH}$ ontologies in SQL databases. In: Patel-Schneider, P.F., Hoekstra, R. (eds.) Proc. OWLED 2009 Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 529 (2009), CEUR-WS.org
5. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press, Menlo Park (2006)
6. Krötzsch, M.: Efficient inferencing for the description logic underlying OWL EL. Tech. Rep. 3005, Institute AIFB, Karlsruhe Institute of Technology (2010), http://www.aifb.kit.edu/web/Techreport3005
7. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: Haarslev, V., Toman, D., Weddell, G. (eds.) Proc. 23rd Int. Workshop on Description Logics, DL 2010 (2010)
8. Krötzsch, M., Rudolph, S., Hitzler, P.: ELP: Tractable rules for OWL 2. In: Sheth, et al. (eds.) [12], pp. 649–664
9. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation (October 27, 2009), http://www.w3.org/TR/owl2-profiles/
10. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic ABoxes. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 227–241. Springer, Heidelberg (2006)
11. Rudolph, S., Krötzsch, M., Hitzler, P.: Description logic reasoning with decision diagrams: Compiling $\mathcal{SHIQ}$ to disjunctive datalog. In: Sheth, et al. (eds.) [12], pp. 435–450
12. Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.): ISWC 2008. LNCS, vol. 5318. Springer, Heidelberg (2008)

Translating First-Order Causal Theories into Answer Set Programming

Vladimir Lifschitz and Fangkai Yang

Department of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
{vl,fkyang}@cs.utexas.edu

Abstract. Nonmonotonic causal logic became a basis for the semantics of several expressive action languages. Norman McCain and Paolo Ferraris showed how to embed propositional causal theories into logic programming, and this work paved the way to the use of answer set solvers for answering queries about actions described in causal logic. In this paper we generalize these embeddings to first-order causal logic—a system that has been used to simplify the semantics of variables in action descriptions.

1 Introduction

Propositional nonmonotonic causal logic [McCain and Turner, 1997] and its generalizations became a basis for the semantics of several expressive action languages [Giunchiglia and Lifschitz, 1998; Giunchiglia *et al.*, 2004; Lifschitz and Ren, 2006; Lifschitz and Ren, 2007]. The last paper argues, in particular, that one of these generalizations—first-order causal logic in the sense of [Lifschitz, 1997]—is useful for defining the semantics of variables in action descriptions.

An important theorem due to Norman McCain [McCain, 1997, Proposition 6.7] shows how to embed a subset of propositional causal logic into the language of logic programming under the answer set semantics [Gelfond and Lifschitz, 1991]. A similar translation, applicable to arbitrary propositional causal theories, is defined in [Ferraris, 2007]. These results (reviewed in the next section) paved the way to the use of answer set programming (ASP) for answering queries about actions described in causal logic [Gebser *et al.*, 2010].

In this note we extend the translations given by McCain and Ferraris to first-order causal theories. Our generalizations rely on the approach to stable models (answer sets) proposed in [Ferraris *et al.*, 2007; Ferraris *et al.*, 2010].

2 Background: Translating Propositional Causal Theories into ASP

2.1 Propositional Causal Theories

A nonmonotonic causal theory in the sense of [McCain and Turner, 1997] is a set of *causal rules* of the form $F \Leftarrow G$, where F and G are propositional formulas

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 247–259, 2010.
© Springer-Verlag Berlin Heidelberg 2010

(the *head* and the *body* of the rule). The rule can be read "F is caused if G is true."

Distinguishing between being true and having a cause turns out to be essential for the study of commonsense reasoning. The assertion "if the light is on at time 2 and you toggle the switch then the light will be off at time 3" can be written as an implication:

$$On_2 \land Toggle_2 \rightarrow Off_3.$$

In causal logic, on the other hand, we can express that under the same assumption *there is a cause* for the light to be off at time 3:

$$Off_3 \Leftarrow On_2 \land Toggle_2.$$

(Performing the toggle action is the cause.) McCain and Turner show that distinctions like this help us solve the frame problem and overcome other difficulties arising in the theory of reasoning about actions.

The semantics of theories of this kind defines when a propositional interpretation (truth assignment) is a model of the given theory (is "causally explained" by the theory, in the terminology of McCain and Turner). We do not reproduce the definition here, because a more general semantics is described below in Section 3.1. But here is an example: the causal theory

$$\begin{array}{c} p \Leftarrow \neg q \\ \neg q \Leftarrow p \end{array} \tag{1}$$

has one model, according to the semantics from [McCain and Turner, 1997]. In this model, p is true and q is false. (Since the bodies of both rules are true in this model, both rules "fire"; consequently the heads of the rules are "caused"; consequently the truth values of both atoms are "causally explained.")

2.2 McCain's Translation

McCain's translation is applicable to a propositional causal theory T if the head of each rule of T is a literal, and the body is a conjunction of literals:

$$L \Leftarrow A_1 \land \cdots \land A_m \land \neg A_{m+1} \land \cdots \neg A_n. \tag{2}$$

The logic program corresponding to T consists of the logic programming rules

$$L \leftarrow not\ \neg A_1, \ldots, not\ \neg A_m, not\ A_{m+1}, \ldots, not\ A_n \tag{3}$$

for all rules (2) of T. According to Proposition 6.7 from [McCain, 1997], complete answer sets of this logic program are identical to the models of T. (A set of literals is *complete* if it contains exactly one member of each complementary pair of literals $A, \neg A$. In the statement above, we identify a complete set of literals with the corresponding truth assignment.)

For instance, McCain's translation turns causal theory (1) into

$$\begin{array}{c} p \leftarrow not\ q \\ \neg q \leftarrow not\ \neg p. \end{array} \tag{4}$$

The only answer set of this program is $\{p, \neg q\}$. It is complete, and it corresponds to the model of causal theory (1).

2.3 Eliminating Strong Negation

Rule (3) involves two kinds of negation: negation as failure (*not*) and strong, or classical, negation ($\neg$). As observed in [Gelfond and Lifschitz, 1991], strong negation can be eliminated from a logic program in favor of additional atoms. Denote the new atom representing a negative literal $\neg A$ by $\widehat{A}$. Then (3) will become

$$A_0 \leftarrow not\ \widehat{A_1}, \ldots, not\ \widehat{A_m}, not\ A_{m+1}, \ldots, not\ A_n \tag{5}$$

if L is a positive literal A_0, and

$$\widehat{A_0} \leftarrow not\ \widehat{A_1}, \ldots, not\ \widehat{A_m}, not\ A_{m+1}, \ldots, not\ A_n \tag{6}$$

if L is a negative literal $\neg A_0$. The *modified McCain translation* of T consists of

- the rules (5), (6) corresponding to the rules of T, and
- the completeness constraints

$$\begin{array}{l} \leftarrow A, \widehat{A} \\ \leftarrow not\ A, not\ \widehat{A} \end{array} \tag{7}$$

  for all atoms A.

For instance, the modified McCain translation of (1) is

$$\begin{array}{l} p \leftarrow not\ q \\ \widehat{q} \leftarrow not\ \widehat{p} \\ \leftarrow p, \widehat{p} \\ \leftarrow not\ p, not\ \widehat{p} \\ \leftarrow q, \widehat{q} \\ \leftarrow not\ q, not\ \widehat{q}. \end{array} \tag{8}$$

The only answer set of this program is $\{p, \widehat{q}\}$.

2.4 Rules as Formulas

The definition of an answer set for sets of propositional formulas proposed in [Ferraris, 2005] is a generalization of the concept of an answer set for propositional logic programs without strong negation, in the sense that rewriting each rule of a given program in the syntax of propositional logic produces a collection of formulas with the same answer sets as the given program. For instance, rules (5) and (6), rewritten as propositional formulas, become

$$\neg\widehat{A_1} \wedge \cdots \wedge \neg\widehat{A_m} \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \rightarrow A_0$$

and

$$\neg\widehat{A_1} \wedge \cdots \wedge \neg\widehat{A_m} \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \rightarrow \widehat{A_0}.$$

The completeness constraints for an atom A turn into

$$\begin{array}{l} \neg(A \wedge \widehat{A}) \\ \neg(\neg A \wedge \neg\widehat{A}). \end{array} \tag{9}$$

Note that the process of rewriting a rule as a formula is applicable only when the rule does not contain strong negation; the symbol $\neg$ in the resulting formula corresponds to the negation as failure symbol (*not*) in the rule.

One of the advantages of writing rules as formulas is that it allows us to relate properties of answer sets to subsystems of classical logic. We know, for instance, that if the equivalence of two sets Γ, Δ of formulas can be proved in intuitionistic logic (or even in the stronger logic of here-and-there) then Γ and Δ have the same answer sets [Ferraris, 2005, Proposition 2]. It follows that replacing the completeness constraints (9) with the intuitionistically equivalent formula

$$\neg(A \leftrightarrow \widehat{A}) \tag{10}$$

does not affect the class of answer sets.

If we rewrite program (8) in the syntax of propositional logic and modify the completeness constraints as shown above then (8) will turn into

$$\begin{array}{l} \neg q \rightarrow p \\ \neg\widehat{p} \rightarrow \widehat{q} \\ \neg(p \leftrightarrow \widehat{p}) \\ \neg(q \leftrightarrow \widehat{q}). \end{array} \tag{11}$$

This collection of formulas is essentially identical to logic program (8), and it has the same answer set.

2.5 Translating Arbitrary Definite Theories

The paper [Ferraris, 2007] shows, among other things, how to lift the requirement, in the definition of McCain's translation, that the bodies of all causal rules should be conjunctions of literals. Take any set T of causal rules of the forms

$$A \Leftarrow G \tag{12}$$

and

$$\neg A \Leftarrow G \tag{13}$$

where A is an atom and G is an arbitrary formula (such rules are called *definite*). For each rule (12), take the formula $\neg\neg G \rightarrow A$, and, for each rule (13), the formula $\neg\neg G \rightarrow \widehat{A}$. Then add completeness constraints (10) for all atoms A. Answer sets of this collection of propositional formulas correspond to the models of T.

In application to example (11), this modification of McCain's translation gives

$$\begin{array}{l} \neg\neg\neg q \rightarrow p \\ \neg\neg p \rightarrow \widehat{q} \\ \neg(p \leftrightarrow \widehat{p}) \\ \neg(q \leftrightarrow \widehat{q}). \end{array} \tag{14}$$

It is not surprising that (14) has the same answer set as (11): the two collections of formulas are intuitionistically equivalent to each other.

2.6 Ferraris's Translation

The main result of [Ferraris, 2007] deals with causal theories "in clausal form": the heads of rules are disjunctions of literals (and the bodies are arbitrary propositional formulas, as in Section 2.5). This is essentially the general case, because any propositional causal theory can be converted to clausal form by converting the head of each rule to conjunctive normal form $D_1 \wedge \cdots \wedge D_k$ and then breaking it into k rules with the heads $D_1, \dots, D_k$.

Ferraris's translation turns the rule

$$\bigvee_{A \in Pos} A \vee \bigvee_{A \in Neg} \neg A \Leftarrow G$$

(Pos and Neg are sets of atoms) into the implication

$$\neg\neg G \wedge \bigwedge_{A \in Pos} (\widehat{A} \vee \neg \widehat{A}) \wedge \bigwedge_{A \in Neg} (A \vee \neg A) \rightarrow \bigvee_{A \in Pos} A \vee \bigvee_{A \in Neg} \widehat{A}.$$

For instance, it transforms $p \vee \neg q \Leftarrow r$ into

$$\neg\neg r \wedge (\widehat{p} \vee \neg \widehat{p}) \wedge (q \vee \neg q) \rightarrow p \vee \widehat{q}. \tag{15}$$

The number of "excluded middle formulas" in the antecedent of the implication, such as $\widehat{p} \vee \neg \widehat{p}$ and $q \vee \neg q$ in (15), equals the number of disjunctive terms in the head of the given causal rule. In particular, the result of Ferraris's translation includes one such formula when the head of the given causal rule is a single literal, as in Section 2.5. For instance, in application to (1) this process would produce

$$\begin{array}{l} \neg\neg\neg q \wedge (\widehat{p} \vee \neg \widehat{p}) \rightarrow p \\ \neg\neg p \wedge (q \vee \neg q) \rightarrow \widehat{q} \\ \neg(p \leftrightarrow \widehat{p}) \\ \neg(q \leftrightarrow \widehat{q}). \end{array} \tag{16}$$

This collection of formulas differs from (14) by the presence of excluded middle formulas in the antecedents of the two implications. These conjuncive terms are redundant: dropping them from (16) is an intuitionistically equivalent transformation and consequently does not affect the collection of answer sets.

But when the result of translating a rule has more than one excluded middle formula in the antecedent, as in example (15), then the presence of these formulas may be crucial for the validity of the translation [Ferraris, 2007, Section 4].

3 Review: Causal Theories and Stable Models in a First-Order Setting

In this section we review the definition of a first-order causal theory from [Lifschitz, 1997] and the definition of a stable model of a first-order sentence from [Ferraris *et al.*, 2010]. Both definitions are based on syntactic transformations that produce second-order formulas.

3.1 First-Order Causal Theories

According to [Lifschitz, 1997], a first-order causal theory T is defined by

- a list $\mathbf{p}$ of distinct predicate constants (other than equality), called the *explainable symbols* of T,[1] and
- a finite set of *causal rules* of the form $F \Leftarrow G$, where F and G are first-order formulas.

The semantics of first-order causal theories can be described as follows. For each $p \in \mathbf{p}$, choose a new predicate variable υp of the same arity, and let $\upsilon\mathbf{p}$ stand for the list of all these variables. By $T^\dagger(\upsilon\mathbf{p})$ we denote the conjunction of the formulas

$$\forall\mathbf{x}(G \to F^{\mathbf{p}}_{\upsilon\mathbf{p}}) \tag{17}$$

for all rules $F \Leftarrow G$ of T, where $\mathbf{x}$ is the list of all free variables of F, G. (The expression $F^{\mathbf{p}}_{\upsilon\mathbf{p}}$ denotes the result of substituting the variables $\upsilon\mathbf{p}$ for the corresponding constants $\mathbf{p}$ in F.) We view T as shorthand for the sentence

$$\forall\upsilon\mathbf{p}(T^\dagger(\upsilon\mathbf{p}) \leftrightarrow (\upsilon\mathbf{p} = \mathbf{p})). \tag{18}$$

(By $\upsilon\mathbf{p} = \mathbf{p}$ we denote the conjunction of the formulas $\forall\mathbf{x}(\upsilon p(\mathbf{x}) \leftrightarrow p(\mathbf{x}))$ for all $p \in \mathbf{p}$, where $\mathbf{x}$ is a tuple of distinct object variables.)

Consider, for instance, the causal theory T with the explainable symbol p that consists of two rules:

$$p(a) \Leftarrow \top$$

(here $\top$ is the logical constant *true*) and

$$\neg p(x) \Leftarrow \neg p(x).$$

The first rule says that there is a cause for a to have property p. The second rule says that if an object does not have property p then there is a cause for that; including this rule in a causal theory has the same effect as saying that p is "false by default" [Lifschitz, 1997, Section 3]. In this case, $T^\dagger(\upsilon p)$ is

$$\upsilon p(a) \wedge \forall x(\neg p(x) \to \neg\upsilon p(x)),$$

so that T is understood as shorthand for the sentence

$$\forall\upsilon p(\upsilon p(a) \wedge \forall x(\neg p(x) \to \neg\upsilon p(x)) \leftrightarrow \forall x(\upsilon p(x) \leftrightarrow p(x))).$$

This sentence is equivalent to the first-order formula $\forall x(p(x) \leftrightarrow x = a)$.

[1] To be precise, the definition in [Lifschitz, 1997] is more general: object and function constants can be treated as explainable as well.

3.2 Operator SM

If p and q are predicate constants of the same arity then $p \leq q$ stands for the formula $\forall \mathbf{x}(p(\mathbf{x}) \rightarrow q(\mathbf{x}))$, where $\mathbf{x}$ is a tuple of distinct object variables. If $\mathbf{p}$ and $\mathbf{q}$ are tuples $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ of predicate constants then $\mathbf{p} \leq \mathbf{q}$ stands for the conjunction

$$(p_1 \leq q_1) \wedge \cdots \wedge (p_n \leq q_n),$$

and $\mathbf{p} < \mathbf{q}$ stands for $(\mathbf{p} \leq \mathbf{q}) \wedge \neg(\mathbf{q} \leq \mathbf{p})$. In second-order logic, we apply the same notation to tuples of predicate variables.

We will define the *stable model operator with the intensional predicates* $\mathbf{p}$, denoted by $\mathrm{SM}_{\mathbf{p}}$ [Ferraris *et al.*, 2010]. Some details of the definition depend on which propositional connectives and quantifiers are treated as primitives, and which of them are viewed as abbreviations. We assume that $\bot$ (falsity), $\wedge$, $\vee$, $\rightarrow$, $\forall$, $\exists$ are the primitives; $\neg F$ stands for $F \rightarrow \bot$, $\top$ stands for $\bot \rightarrow \bot$, and $F \leftrightarrow G$ is $(F \rightarrow G) \wedge (G \rightarrow F)$.

Let $\mathbf{p}$ be a list of distinct predicate constants (other than equality). For each $p \in \mathbf{p}$, choose a new predicate variable υp of the same arity, and let $\upsilon\mathbf{p}$ stand for the list of all these variables. For any first-order sentence F, by $\mathrm{SM}_{\mathbf{p}}[F]$ we denote the second-order sentence

$$F \wedge \neg\exists \upsilon\mathbf{p}((\upsilon\mathbf{p} < \mathbf{p}) \wedge F^*(\upsilon\mathbf{p})),$$

where $F^*(\upsilon\mathbf{p})$ is defined recursively:

- $p(\mathbf{t})^* = \upsilon p(\mathbf{t})$ for any $p \in \mathbf{p}$ and any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic F that does not contain members of $\mathbf{p}$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

A model of F is *stable* (relative to the set $\mathbf{p}$ of intensional predicates) if it satisfies $\mathrm{SM}_{\mathbf{p}}[F]$.

For instance, let F be the formula

$$\forall x(p(x) \rightarrow (q(x) \vee \neg q(x))$$

(it represents the LPARSE choice rule `{q(X)} :- p(X).`[2]). If we take q to be the only intensional predicate then $F^*(\upsilon q)$ is

$$\forall x((p(x) \rightarrow (\upsilon q(x) \vee (\neg\upsilon q(x) \wedge \neg q(x)))) \wedge (p(x) \rightarrow (q(x) \vee \neg q(x)))),$$

which is equivalent to $\forall x(p(x) \rightarrow (\upsilon q(x) \vee \neg q(x)))$. Consequently $\mathrm{SM}_q[F]$ is equivalent to

$$\forall x(p(x) \rightarrow (q(x) \vee \neg q(x))) \wedge \neg\exists \upsilon q((\upsilon q < q) \wedge \forall x(p(x) \rightarrow (\upsilon q(x) \vee \neg q(x)))).$$

[2] For a description of the language see `http://www.tcs.hut.fi/Software/smodels/lparse.ps`

The first conjunctive term here is logically valid and can be dropped. The second is equivalent to the first-order formula $\forall x(q(x) \to p(x))$, which reflects the intuitive meaning of choice: q is an arbitrary subset of p.

4 Translating First-Order Causal Theories

4.1 A First-Order Counterpart of McCain's Translation

In this section we extend the McCain translation as described in Section 2.5 to first-order causal theories. By T we denote here a causal theory in the sense of Section 3.1 such that the head of every rule of T is a literal containing an explainable predicate. Thus every rule of T has the form

$$p(\mathbf{t}) \Leftarrow G \tag{19}$$

or the form

$$\neg p(\mathbf{t}) \Leftarrow G, \tag{20}$$

where p is an explainable predicate and $\mathbf{t}$ is a tuple of terms. For instance, the example at the end of Section 3.1

$$\begin{array}{l} p(a) \Leftarrow \top \\ \neg p(x) \Leftarrow \neg p(x) \end{array} \tag{21}$$

is a causal theory of this type.

For every member p of the list $\mathbf{p}$ of explainable predicates, let $\widehat{p}$ be a new predicate constant of the same arity, and let $\widehat{\mathbf{p}}$ be the list of all these predicate constants. By CC we denote the conjunction of the formulas

$$\forall \mathbf{x} \neg (p(\mathbf{x}) \leftrightarrow \widehat{p}(\mathbf{x})), \tag{22}$$

where $\mathbf{x}$ is a tuple of distinct object variables, for all p from $\mathbf{p}$. These formulas are first-order counterparts of completeness constraints (10).

The *McCain translation* MC[T] of T is the conjunction of

- formulas $\widetilde{\forall}(\neg\neg G \to p(\mathbf{t}))$ for all rules (19) of T, and
- formulas $\widetilde{\forall}(\neg\neg G \to \widehat{p}(\mathbf{t}))$ for all rules (20) of T, and
- completeness constraints CC.

(The symbol $\widetilde{\forall}$ denotes universal closure.) For instance, if T is (21) then MC[T] is the conjunction of the formulas

$$\begin{array}{l} \neg\neg\top \to p(a) \\ \forall x(\neg\neg\neg p(x) \to \widehat{p}(x)) \\ \forall x \neg (p(x) \leftrightarrow \widehat{p}(x)), \end{array}$$

or, after (intuitionistically acceptable) simplifications,

$$p(a) \wedge \forall x(\neg p(x) \to \widehat{p}(x)) \wedge \forall x \neg (p(x) \leftrightarrow \widehat{p}(x)).$$

In logic programming syntax, this formula can be rewritten as

$$\begin{array}{l} p(a) \\ \widehat{p}(x) \leftarrow not\ p(x) \\ \qquad \leftarrow p(x), \widehat{p}(x) \\ \qquad \leftarrow not\ p(x), not\ \widehat{p}(x). \end{array} \tag{23}$$

Theorem 1. *The sentence* $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{MC}[T]]$ *is equivalent to* $T \wedge CC$.[3]

Note that formula (22) is classically equivalent to

$$\forall \mathbf{x}(\widehat{p}(\mathbf{x}) \leftrightarrow \neg p(\mathbf{x})), \tag{24}$$

so that CC can be viewed as the conjunction of explicit definitions of the predicates $\widehat{\mathbf{p}}$ in terms of the predicates $\mathbf{p}$. Since the predicates $\widehat{\mathbf{p}}$ do not belong to the language of T, Theorem 1 shows that $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{MC}[T]]$ is a definitional, and consequently conservative, extension of T. In other words, the $\mathbf{p}\widehat{\mathbf{p}}$-stable models of $\mathrm{MC}[T]$ are identical to the models of T extended by the interpretations of the predicates $\widehat{\mathbf{p}}$ given by explicit definitions (24). Note also that in this characterization of the stable models of $\mathrm{MC}[T]$ the set of intensional predicates includes both the explainable predicates $\mathbf{p}$ of T and the corresponding predicates $\widehat{\mathbf{p}}$.

4.2 A First-Order Counterpart of Ferraris's Translation

In this section, T is a causal theory in the sense of Section 3.1 such that

- the head of each rule of T is a disjunction of literals, and
- all predicate constants occurring in the heads of rules are explainable.

In other words, we assume that every rule of T has the form

$$\bigvee_{A \in Pos} A \vee \bigvee_{A \in Neg} \neg A \Leftarrow G \tag{25}$$

for some sets Pos, Neg of atomic formulas that contain a predicate constant from the set $\mathbf{p}$ of explainable symbols.

As in Section 4.1, for each $p \in \mathbf{p}$ we choose a new predicate constant $\widehat{p}$ of the same arity. If A is an atomic formula $p(\mathbf{t})$, where $p \in \mathbf{p}$ and $\mathbf{t}$ is a tuple of terms, then $\widehat{A}$ stands for $\widehat{p}(\mathbf{t})$.

The *Ferraris translation* $\mathrm{Fer}[T]$ of T is the conjunction of

- formulas

$$\widetilde{\forall}\left(\neg\neg G \wedge \bigwedge_{A \in Pos} (\widehat{A} \vee \neg \widehat{A}) \wedge \bigwedge_{A \in Neg} (A \vee \neg A) \rightarrow \bigvee_{A \in Pos} A \vee \bigvee_{A \in Neg} \widehat{A}\right) \tag{26}$$

  for all rules (25) of T, and
- completeness constraints CC.

[3] Recall that we identify a causal theory T with the corresponding sentence (18).

For instance, if T is (21) then Fer[T] is the conjunction of the formulas

$$\neg\neg\top \wedge (\widehat{p}(a) \vee \neg\widehat{p}(a)) \rightarrow p(a)$$
$$\forall x(\neg\neg\neg p(x) \wedge (p(x) \vee \neg p(x)) \rightarrow \widehat{p}(x))$$
$$\forall x\neg(p(x) \leftrightarrow \widehat{p}(x)).$$

Theorem 2. *The sentence* $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Fer}[T]]$ *is equivalent to* $T \wedge CC$.

Thus the $\mathbf{p}\widehat{\mathbf{p}}$-stable models of Fer[$T$] are identical to the models of T extended by the interpretations of the predicates $\widehat{\mathbf{p}}$ given by explicit definitions (24).

5 Proof Outlines

Our proofs of Theorems 1 and 2 are quite different from the published proofs of similar results for the propositional case, because of the difference between the semantics used in this paper (Section 3) and the semantics of propositional causal theories and logic programs, which are based on reducts.

Theorem 1 follows from Theorem 2 in view of the following fact:

Lemma 1. *For any causal theory* T *consisting of rules of forms (19) and (20),* MC[T] *is intuitionistically equivalent to* Fer[T].

In the proof of Theorem 2, Π stands for the conjunction of sentences (26) for all rules (25) in T. Then Fer[T] is $\Pi \wedge CC$. Formula $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\mathrm{Fer}[T]]$ is equivalent to $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi] \wedge CC$, because CC has no strictly positive occurrences of intensional predicates [Ferraris *et al.*, 2010, Section 5.1]. Therefore the statement of Theorem 2 is equivalent to the claim that CC entails

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi] \leftrightarrow T. \tag{27}$$

By $\upsilon\mathbf{p}$, $\upsilon\widehat{\mathbf{p}}$ we denote the lists of the predicate variables υp, $\upsilon\widehat{p}$ used in the second-order formula $\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi]$ (see Section 3.2). If A is an atomic formula $p(\mathbf{t})$, where $p \in \mathbf{p}$ and $\mathbf{t}$ is a tuple of terms, then we will write υA for $\upsilon p(\mathbf{t})$, and $\upsilon\widehat{A}$ for $\upsilon\widehat{p}(\mathbf{t})$. By $\widetilde{\forall}_{obj} F$ we denote the formula $\forall \mathbf{x} F$, where $\mathbf{x}$ is list of all free object variables of F ("object-level universal closure").

The expression $H(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}})$ stands for the conjunction of the implications

$$\widetilde{\forall}_{obj}\left(G \rightarrow \bigvee_{A\in Pos}((\upsilon\widehat{A} \vee A) \rightarrow \upsilon A) \vee \bigvee_{A\in Neg}((\upsilon A \vee \neg A) \rightarrow \upsilon\widehat{A})\right)$$

for all rules (25) in T. The role of this formula is determined by the following lemma:

Lemma 2. *Formula* CC *entails*

$$\mathrm{SM}_{\mathbf{p}\widehat{\mathbf{p}}}[\Pi] \leftrightarrow \Pi \wedge \forall(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})(((\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}}) < (\mathbf{p}, \widehat{\mathbf{p}})) \rightarrow \neg H(\upsilon\mathbf{p}, \upsilon\widehat{\mathbf{p}})).$$

For any formula F, by $F_{\Sigma 1}$ we denote the formula

$$F^{(\upsilon\mathbf{p})(\upsilon\widehat{\mathbf{p}})}_{(\upsilon\mathbf{p}\wedge\mathbf{p})(\neg\upsilon\mathbf{p}\wedge\neg\mathbf{p})}$$

where $\upsilon\mathbf{p}\wedge\mathbf{p}$ is understood as the list of predicate expressions[4] $\lambda\mathbf{x}(\upsilon p(\mathbf{x})\wedge p(\mathbf{x}))$ for all $p\in\mathbf{p}$, and $\neg\upsilon\mathbf{p}\wedge\neg\mathbf{p}$ is understood in a similar way.

Lemma 3. *Formula* $((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p}))_{\Sigma 1}$ *is equivalent to* $\upsilon\mathbf{p}\neq\mathbf{p}$. *Formula* $H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})_{\Sigma 1}$ *is equivalent to* $T^{\dagger}(\upsilon\mathbf{p})$.

The proof of the first part of this lemma is based on the fact that formula $(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p})$ is equivalent to

$$\bigvee_{p\in\mathbf{p}}(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\leq(\mathbf{p},\neg\mathbf{p}))\wedge\exists\mathbf{x}(\neg\upsilon p(\mathbf{x})\wedge\neg\upsilon\widehat{p}(\mathbf{x}))).$$

The fact that CC entails the "only if" part of equivalence (27) follows from Lemmas 2 and 3.

For any formula F, by $F_{\Sigma 2}$ we denote the formula

$$F^{\upsilon\mathbf{p}}_{(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\leq(\mathbf{p},\neg\mathbf{p}))\wedge\neg\upsilon\mathbf{p}\wedge\neg\upsilon\widehat{\mathbf{p}})\leftrightarrow\neg\mathbf{p}}$$

where the subscript

$$(((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\leq(\mathbf{p},\neg\mathbf{p}))\wedge\neg\upsilon\mathbf{p}\wedge\neg\upsilon\widehat{\mathbf{p}})\leftrightarrow\neg\mathbf{p}$$

is understood as the list of predicate expressions

$$\lambda\mathbf{x}((((\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\leq(\mathbf{p},\neg\mathbf{p}))\wedge\neg\upsilon p(\mathbf{x})\wedge\neg\upsilon\widehat{p}(\mathbf{x}))\leftrightarrow\neg p(\mathbf{x}))$$

for all $p\in\mathbf{p}$.

Lemma 4. *Formula* $(\upsilon\mathbf{p}\neq\mathbf{p})_{\Sigma 2}$ *is equivalent to* $(\upsilon\mathbf{p},\widehat{\mathbf{p}})<(\mathbf{p},\neg\mathbf{p})$. *The implication* $(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}})\leq(\mathbf{p},\neg\mathbf{p})\rightarrow(T^{\dagger}(\upsilon\mathbf{p})_{\Sigma 2}\leftrightarrow H(\upsilon\mathbf{p},\upsilon\widehat{\mathbf{p}}))$ *is logically valid.*

The fact that CC entails the "if" part of equivalence (27) follows from Lemmas 2 and 4.

6 Conclusion

The definition of a stable model based on the operator SM, reviewed in Section 3.2, is more general than the traditional defintion [Gelfond and Lifschitz, 1988] in several ways. It is more general syntactically, because it is applicable to formulas containing quantifiers. It is more general semantically, in the sense that it is applicable to non-Herbrand models. It also allows us to distinguish between intensional and extensional predicates. Ferraris *et al.* [2010] argued that these features can be useful in applications to knowledge representation. They showed how to extend many

[4] See [Lifschitz, 1994, Section 3.1].

familiar properties of stable models to the first-order case. This line of work was continued in [Lee *et al.*, 2008] and [Ferraris *et al.*, 2009], and the theorems presented in this paper belong to the same direction of research.

We expect that Theorem 2 will help us extend the theorem on synonymity proved in [Lee *et al.*, 2010] to first-order causal theories, and that it will help us in this way to design a new implementation of modular action language MAD [Erdoğan, 2008; Ren, 2009].

In application to causal theories with variables, the translations defined in this paper often generate logic programs that are not safe and thus cannot be processed by existing answer set solvers.[5] For instance, the second rule of (23) is unsafe, because the only occurrence of x in its body is in the scope of negation as failure. It may be possible to find modifications of the McCain and Ferraris translations that produce safe logic programs in practically important cases.

Another topic for future research is extending the translations to causal theories with explainable object and function constants; such constants correspond to non-Boolean fluents in action languages.

Acknowledgements

We are grateful to the anonymous referees for useful comments. This research was partially supported by the National Science Foundation under grant IIS-0712113.

References

[Erdoğan, 2008] Erdoğan, S.T.: A Library of General-Purpose Action Descriptions[6]. PhD thesis, University of Texas at Austin (2008)

[Ferraris *et al.*, 2007] Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 372–379 (2007)

[Ferraris *et al.*, 2009] Ferraris, P., Lee, J., Lifschitz, V., Palla, R.: Symmetric splitting in the general theory of stable models. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 797–803 (2009)

[Ferraris *et al.*, 2010] Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription[7]. Artificial Intelligence (to appear, 2010)

[Ferraris, 2005] Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 119–131. Springer, Heidelberg (2005)

[Ferraris, 2007] Ferraris, P.: A logic program characterization of causal theories. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 366–371 (2007)

[Gebser *et al.*, 2010] Gebser, M., Grote, T., Schaub, T.: Coala: a compiler from action languages to ASP. In: Janhunen, T., Niemelä, I. (eds.) JELIA 2010. LNCS (LNAI), vol. 6341, pp. 357–359. Springer, Heidelberg (2010)

[5] See Chapter 3 of the DLV manual, `http://www.dbai.tuwien.ac.at/proj/dlv/man/`

[6] `http://www.cs.utexas.edu/users/tag/mad/erdogan-dissertation.pdf`

[7] `http://peace.eas.asu.edu/joolee/papers/smcirc.pdf`

[Gelfond and Lifschitz, 1988] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of International Logic Programming Conference and Symposium, pp. 1070–1080. MIT Press, Cambridge (1988)

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

[Giunchiglia and Lifschitz, 1998] Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proceedings of National Conference on Artificial Intelligence (AAAI), pp. 623–630. AAAI Press, Menlo Park (1998)

[Giunchiglia *et al.*, 2004] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence 153(1-2), 49–104 (2004)

[Lee *et al.*, 2008] Lee, J., Lifschitz, V., Palla, R.: Safe formulas in the general theory of stable models (preliminary report). In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 672–676. Springer, Heidelberg (2008)

[Lee *et al.*, 2010] Lee, J., Lierler, Y., Lifschitz, V., Yang, F.: Representing synonymity in causal logic and in logic programming[8]. In: Proceedings of International Workshop on Nonmonotonic Reasoning, NMR (2010)

[Lifschitz and Ren, 2006] Lifschitz, V., Ren, W.: A modular action description language. In: Proceedings of National Conference on Artificial Intelligence (AAAI), pp. 853–859 (2006)

[Lifschitz and Ren, 2007] Lifschitz, V., Ren, W.: The semantics of variables in action descriptions. In: Proceedings of National Conference on Artificial Intelligence (AAAI) (2007)

[Lifschitz, 1994] Lifschitz, V.: Circumscription. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.) Handbook of Logic in AI and Logic Programming, vol. 3, pp. 298–352. Oxford University Press, Oxford (1994)

[Lifschitz, 1997] Lifschitz, V.: On the logic of causal explanation. Artificial Intelligence 96, 451–465 (1997)

[McCain and Turner, 1997] McCain, N., Turner, H.: Causal theories of action and change. In: Proceedings of National Conference on Artificial Intelligence (AAAI), pp. 460–465 (1997)

[McCain, 1997] McCain, N.: Causality in Commonsense Reasoning about Actions[9]. PhD thesis, University of Texas at Austin (1997)

[Ren, 2009] Ren, W.: A Modular Language for Describing Actions[10], PhD thesis, University of Texas at Austin (2009)

[8] `http://userweb.cs.utexas.edu/users/vl/papers/syn.pdf`

[9] `ftp://ftp.cs.utexas.edu/pub/techreports/tr97-25.ps.gz`

[10] `http://www.cs.utexas.edu/users/rww6/dissertation.pdf`

Preprocessing Boolean Formulae for BDDs in a Probabilistic Context

Theofrastos Mantadelis[1], Ricardo Rocha[2], Angelika Kimmig[1], and Gerda Janssens[1]

[1] Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
{Theofrastos.Mantadelis,Angelika.Kimmig,Gerda.Janssens}@cs.kuleuven.be
[2] CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
ricroc@dcc.fc.up.pt

Abstract. Inference in many probabilistic logic systems is based on representing the proofs of a query as a DNF Boolean formula. Assessing the probability of such a formula is known as a #P-hard task. In practice, a large DNF is given to a BDD software package to construct the corresponding BDD. The DNF has to be transformed into the input format of the package. This is the preprocessing step. In this paper we investigate and compare different preprocessing methods, including our new trie based approach. Our experiments within the ProbLog system show that the behaviour of the methods changes according to the amount of sharing in the original DNF. The decomposition method is preferred when there is not much sharing in the DNF, whereas DNFs with sharing benefit from our trie based method. While our methods are motivated and applied in the ProbLog context, our results are interesting for other applications that manipulate DNF Boolean formulae.

Keywords: Boolean Formula Manipulation, Binary Decision Diagrams, ProbLog, Probabilistic Logic Learning.

1 Introduction

The past few years have seen a surge of interest in the field of Probabilistic Logic Learning (PLL) [1], also known as Statistical Relational Learning [2]. A multitude of formalisms combining logical or relational languages with probabilistic reasoning has been developed. One line of work, based on the distribution semantics [3], extends Logic Programming (LP) with probabilistic facts, i.e., facts whose truth values are determined probabilistically. Main representatives of this approach are PRISM [4], ICL [5] and ProbLog [6]. Even in such simple probabilistic logics, inference is computationally hard. As learning requires evaluating large amounts of queries, efficient inference engines are crucial for PLL.

The core of inference in these LP-based languages is a reduction to propositional formulae in Disjunctive Normal Form (DNF). Such a DNF describes all

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 260–272, 2010.
© Springer-Verlag Berlin Heidelberg 2010

proofs of a query in terms of the probabilistic facts used, thus reducing probabilistic inference to calculating the probability of a DNF formula. The PRISM system requires programs to ensure that the probability of the DNF corresponds to a sum of products. ProbLog has been motivated by a biological network mining task where this is impossible, and therefore obtains the probability from an external Binary Decision Diagram (BDD) tool. To this aim, the DNF is first constructed using logical inference, and then preprocessed into a sequence of BDD definitions which builds up the final BDD by applying Boolean operations on subformulae. While previous work on the efficient implementation of ProbLog has been focused on obtaining the DNF, little attention has been devoted to its further processing. However, as the performance of BDD construction depends on the size and structure of the intermediate BDDs and on the operations among them, the performance of this second phase is crucial for the overall performance of inference in ProbLog.

In this paper, we therefore study different approaches to preprocessing with special attention to the exploitation of repeated formulae to avoid redundant work in BDD construction. To this aim, we introduce a new data structure, named *depth breadth trie*, which facilitates detecting repeated subformulae in the DNF. This results in an improvement of the performance of ProbLog's preprocessing step. At the same time, this new data structure allows one to easily identify more shared subformulae, which can be used to further simplify BDD construction. A second contribution of this work is the implementation in ProbLog of an alternative preprocessing method, called *decomposition* [7], which we used to perform a comparative study of preprocessing methods in ProbLog. Our experimental results show that in structured problems, our trie based approaches are clearly outperforming the decomposition method, but in less structured problems decomposition seems to be better.

The remainder of the paper is organized as follows. First, Section 2 briefly introduces some background concepts about ProbLog, tries and BDDs. Next, Section 3 reviews different preprocessing methods. Then, we present our new approach in detail, including three new optimizations that can be performed with the depth breadth trie in Section 4. We present experimental results in Section 5 and end by outlining some conclusions in Section 6.

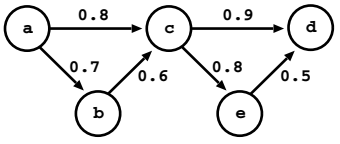
2 ProbLog

A ProbLog program T [6] consists of a set of labeled ground facts $p_i :: c_i$ together with a set of definite clauses. Each such fact c_i is true with probability p_i, i.e., these facts correspond to random variables, which are assumed to be mutually independent. Together, they define a distribution over subsets of $L_T = \{c_1, \ldots, c_n\}$. The definite clauses allow one to add arbitrary *background knowledge* (BK) to those sets of *logical* facts. Given the one-to-one mapping between ground definite clause programs and Herbrand interpretations, a ProbLog program also defines a distribution over its Herbrand interpretations.

Inference in ProbLog calculates the *success probability* $P_s(q|T)$ of a query q in a ProbLog program T, i.e., the probability that the query q is *provable* in a program

that combines BK with a randomly sampled subset of L_T. Figure 1 shows a ProbLog program encoding a probabilistic graph. The success probability of `path(a,d)` corresponds to the probability that a randomly sampled subgraph contains at least one of the four possible paths from node `a` to node `d`.

As checking whether a query is provable in each subprogram is clearly infeasible in most cases, ProbLog inference uses a reduction to a Boolean formula in DNF. This formula describes the set of programs where the query is provable. Variables in the formula correspond to probabilistic facts in the program, conjunctions correspond to specific proofs, and the entire disjunction to the set of all proofs. ProbLog then calculates the probability of that formula being true.




```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8 :: edge(a,c).   0.9 :: edge(c,d).
0.7 :: edge(a,b).   0.8 :: edge(c,e).
0.6 :: edge(b,c).   0.5 :: edge(e,d).
```

Fig. 1. A probabilistic graph and its encoding in ProbLog

While the probability of a single conjunction, which represents the programs containing at least the facts used by the corresponding proof, is the product of the probabilities of these facts, it is impossible to simply sum the probabilities of conjunctions, as enumerating proofs does not partition the set of programs. Instead, we face the so called *disjoint-sum-problem*, which is known to be #P-hard [8]. By tackling this problem with (reduced ordered) BDDs [9], a graphical representation of Boolean formulae that enables probability calculation by means of dynamic programming, the ProbLog implementation scales to DNFs with tens of thousands of conjunctions.

ProbLog programs are executed in three steps. Given a ProbLog program T and a query q, the first step, *SLD-resolution*, collects all proofs for query q in $BK \cup L_T$. Proofs are stored as lists of identifiers corresponding to probabilistic facts in a *trie data structure*. This trie represents the DNF for query q. An essential property of the trie data structure is that common prefixes are stored only once, which in the context of ProbLog allows us to exploit natural prefix sharing of proofs, as two proofs with common prefix will branch off from each other at the first distinguishing probabilistic fact.

The second step, *preprocessing*, converts the DNF represented by the trie into a so-called *script*. A script is a sequence of BDD definitions, which define BDDs corresponding to Boolean random variables or Boolean formulae obtained by applying Boolean operators to previously defined BDDs. The last BDD defined in the script corresponds to the entire DNF.

Finally, the third step, *BDD construction*, follows the script to construct a sequence of intermediate BDDs leading to the final BDD for probability calculation. To this aim, ProbLog uses the front-end SimpleCUDD[1] for the BDD package CUDD[2].

[1] `http://people.cs.kuleuven.be/~theofrastos.mantadelis/tools/simplecudd.html`

[2] `http://vlsi.colorado.edu/~fabio/CUDD/`

The complexity of combining BDDs by Boolean operators is proportional to the product of their sizes, which depend on the variable order used by the BDD package and can be exponential in the number of Boolean variables. As computing the order that minimizes the size of a BDD is a coNP-complete problem [9], BDD packages include heuristics to reduce the size by reordering variables. While reordering is often necessary to handle large BDDs, it can be quite expensive. To control the complexity of BDD construction, it is therefore crucial to restrain the size of the intermediate BDDs and the amount of operations performed. At the very least, preprocessing should aim to avoid repeated construction of BDDs for identical subformulae. In this work, we therefore use tries to exploit prefix sharing on the level of proofs as well as – by means of the new data structure depth breadth trie – on the level of BDD definitions.

3 From Tries to BDDs

In this section, we discuss the different approaches for preprocessing. Remember that preprocessing converts a DNF (represented as trie) to a script. We will use the example in Figure 1 as our running example. Figure 2 shows the set of proofs and the trie for the query `path(a,d)`. On top, the four proofs of the query are represented as conjunctions, where we use xy to denote the Boolean variable corresponding to probabilistic fact `edge(x,y)`. The disjunction of those conjunctions is depicted as a trie, where each branch of the trie corresponds to one conjunction. For simplicity of illustration, in the figures that follow, we will use the same xy notation. In scripts, we use n_i to refer to the i^{th} defined BDD.

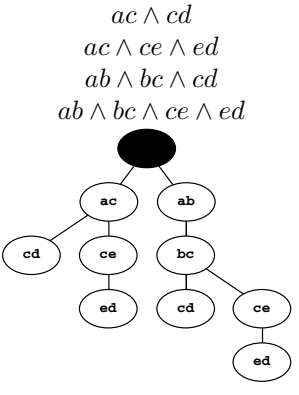


Fig. 2. Collected proofs and respective trie for `path(a,d)`

The *naive method* directly mirrors the structure of the DNF by first constructing all conjunctions of the DNF formula and then combining those in one big disjunction. Figure 3 shows, on the left, the resulting script for the proofs of our example. The worst case complexity of this preprocessing step is $O(N \cdot M)$[3].

The *decomposition method* [7] recursively divides a Boolean formula in DNF into smaller ones until only one variable remains. To do so, it first chooses a Boolean variable from the formula, the so-called decomposition variable dv, and then breaks the formula into three subformulae. The first subformula f_1' contains the conjunctions that include dv, the second subformula f_2' those that include the negation of dv and the third subformula f_3 those that include neither of the two. Then, by applying the distribution axiom, the original Boolean formula can be re-written as $f = f_1' \vee f_2' \vee f_3 = (dv \wedge f_1) \vee (\neg dv \wedge f_2) \vee f_3$.

[3] Complexity results for N proofs and M probabilistic facts. For more details see: `https://lirias.kuleuven.be/bitstream/123456789/270070/2/complexity.pdf`

Naive Method	Decomposition Method	Recursive Node Merging
$n_1 = ac \wedge cd$	$n_1 = ce \wedge ed$	$n_1 = ce \wedge ed$
$n_2 = ac \wedge ce \wedge ed$	$n_2 = cd \vee n_1$	$n_2 = cd \vee n_1$
$n_3 = ab \wedge bc \wedge cd$	$n_3 = ce \wedge ed$	$n_3 = ac \wedge n_2$
$n_4 = ab \wedge bc \wedge ce \wedge ed$	$n_4 = cd \vee n_3$	$n_4 = bc \wedge n_2$
$n_5 = n_1 \vee n_2 \vee n_3 \vee n_4$	$n_5 = bc \wedge n_4$	$n_5 = ab \wedge n_4$
	$n_6 = ab \wedge n_5$	$n_6 = n_3 \vee n_5$
	$n_7 = ac \wedge n_2$	
	$n_8 = n_7 \vee n_6$	

Fig. 3. Scripts obtained by different preprocessing methods for the example DNF

Algorithm 1. Recursive node merging. Takes a trie T representing a DNF and an index i and writes a script. REPLACE(T, C, n_i) replaces each occurrence of C in T by n_i.

function RECURSIVE_NODE_MERGING(T, i)
  if $\neg leaf(T)$ **then**
    $S_\wedge := \{(C, P) |$ leaf C is the only child of P in $T\}$
    for all $(C, P) \in S_\wedge$ **do**
      write $n_i = P \wedge C$
      $T :=$ REPLACE$(T, (C, P), n_i)$
      $i := i + 1$
    $S_\vee := \{[C_1, \ldots, C_n] ||$ leaves C_j are all the children of some node P in $T, n > 1\}$
    for all $[C_1, \ldots, C_n] \in S_\vee$ **do**
      write $n_i = C_1 \vee \ldots \vee C_n$
      $T :=$ REPLACE$(T, [C_1, \ldots, C_n], n_i)$
      $i := i + 1$
    RECURSIVE_NODE_MERGING(T, i)

As all three new subformulae f_1, f_2 and f_3 do not contain dv, they can be decomposed independently. The most basic choice for the decomposition variable is the first variable of the current formula, however, various heuristic functions can be used as well, cf. [7]. All definitions resulting from the same decomposition step are written as a block at the end of that step, omitting those equivalent to $false$ to avoid unnecessary BDD operations. Figure 3 (middle column) again shows the result for our example query. The worst case complexity is $O(N \cdot M^2)$.

The approach followed in ProbLog, as described in [6], exploits the sharing of both prefixes – as directly given by the tries – and suffixes, which have to be extracted algorithmically. We will call this approach *recursive node merging*.

Recursive node merging traverses the trie representing the DNF bottom-up. In each iteration it applies two different operations that reduce the trie by merging nodes. The first operation (*depth reduction*) creates the conjunction of a leaf node with its parent, provided that the leaf is the only child of the parent. The second operation (*breadth reduction*) creates the disjunction of all child nodes of a node, provided that these child nodes are all leaves. Algorithm 1 shows the details for recursive node merging and Figure 4 illustrates its step-by-step

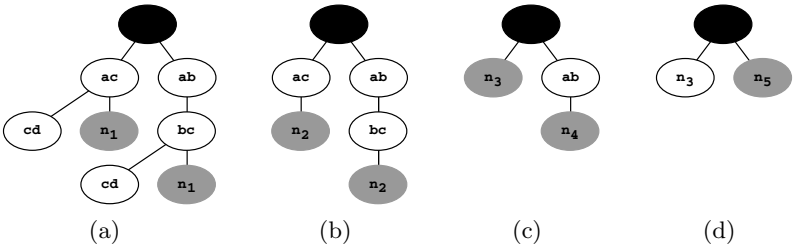


Fig. 4. Tries obtained during recursive node merging applied to the trie for `path(a,d)`

application to the example trie in Figure 2. The resulting script can be found on the right in Figure 3.

For both reduction types, a subtree that occurs multiple times in the trie is reduced only once, and the resulting conjunction/disjunction is used for all occurrences of that subtree, thus performing some suffix sharing. Note however that the REPLACE() procedure can be quite costly when fully traversing the trie to search for repeated occurrences of subtrees.

4 Depth Breadth Trie

In this section, we introduce our new approach to implementing recursive node merging. The initial implementation explicitly performed the costly REPLACE() procedure of Algorithm 1. The new approach avoids this by storing all BDD definitions during recursive node merging. Once merging is completed, the script is obtained from this store. For each definition encountered during merging, we first check if it is already present in the store, and if so, reuse the corresponding reference n_i. As such a check/insert operation can be done in a single pass for tries, we introduce an additional and specific trie configuration for this purpose, that we named *depth breadth trie*. Apart from this improvement, the depth breadth trie has the additional advantage of allowing one to easily identify common prefixes on the level of BDD definitions, which was not possible before. As we will see, this leads to the definition of three new (optional) optimizations that can be performed during recursive node merging to further reduce the number of Boolean operations to be performed in BDD construction.

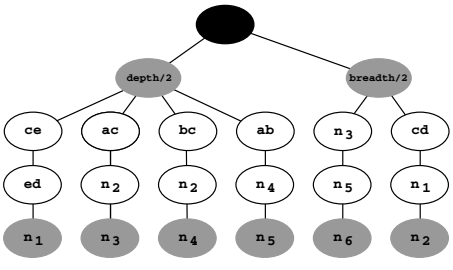
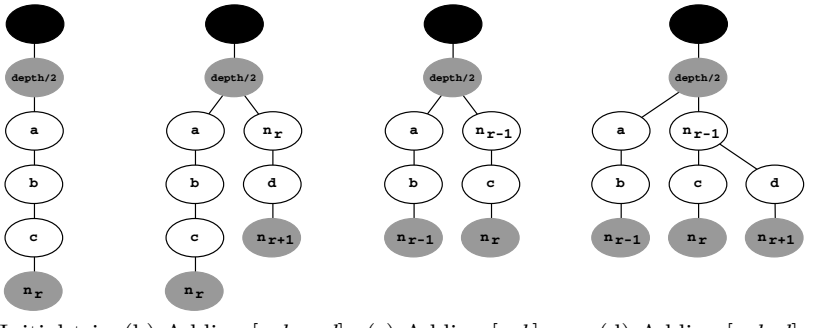


Fig. 5. Depth breadth trie containing a complete set of definitions (right column of Figure 3) for the DNF Boolean formula representing `path(a,d)`. Each leaf node contains a unique reference identifying the corresponding path's definition. Each branch in the depth/2 (breadth/2) part defines the conjunction (disjunction) of the entries in its white nodes (Boolean variables or definition references).

A depth breadth trie is divided in two parts corresponding to the two reduction types of recursive node merging: the *depth part* collects the conjunctions, the *breadth part* the disjunctions. This separation is achieved by two specific functors of arity two, `depth/2` and `breadth/2`. Their first argument is a Prolog list containing the literals that participate in the formula, the second argument is the unique reference n_i assigned to the corresponding BDD definition.

For example, the definitions $n1 = ce \wedge ed$ and $n2 = cd \vee n1$ are represented respectively by the terms `depth([ce,ed],n1)` and `breadth([cd,n1],n2)`. Note that reference `n1` introduced by the first term is used in the second term to refer to the corresponding subformula. At the same time, those references provide the order in which BDDs are defined in the script. Figure 5 shows the complete depth breadth trie built by recursive node merging for our example.

In the following, we introduce the three new optimizations that can be exploited with the depth breadth trie. The motivation is again to decrease the amount of operations performed in BDD construction. The optimizations are illustrated in Figure 6. Figure 6(a) presents the initial trie used in all cases, Figures 6(b), 6(c) and 6(d) show Optimizations I, II and III, respectively. The worst case complexity is $O(N \cdot M)$ in all cases.



(a) Initial trie (b) Adding $[a, b, c, d]$ (c) Adding $[a, b]$ (d) Adding $[a, b, d]$

Fig. 6. Examples of definitions that trigger (b) Optimization I, (c) Optimization II and (d) Optimization III when added to the depth breadth trie in (a)

Optimization I (Contains Prefix): The first optimization occurs when a new formula $[p_1, \ldots, p_n]$ to be added to the depth breadth trie contains as prefix an existing formula $[p_1, \ldots, p_i]$, $i \geq 2$, with reference n_r. In this case, the existing formula will be reused and, instead of inserting $[p_1, \ldots, p_n]$, we will insert $[n_r, p_{i+1}, \ldots, p_n]$ and assign a new reference to it.

Optimization II (Is Prefix): The second optimization considers the inverse case of the first optimization. It occurs when a new formula $[p_1, \ldots, p_i]$, $i \geq 2$, to be added to the depth breadth trie is a prefix of an existing formula $[p_1, \ldots, p_n]$ with reference n_r. In this case, we split the existing subformula representing $[p_1, \ldots, p_n]$ in two: the first one representing the new formula

Algorithm 2. Depth breadth trie optimizations. Takes a T representing either the depth or breadth part of the depth breadth trie and a list L with the formula to be added and returns the reference n_i assigned to L in T. $COUNTER$ is a global counter and REPLACE(L, C, n_i) replaces C in L by n_i.

```
function UPDATE_DEPTH_BREADTH_TRIE(T, L)
  if (L, n_i) ∈ T then
    return n_i
  for all (list, n_i) ∈ T do
    if list is prefix of L then
      /* Optimization I */
      L := REPLACE(L, list, n_i)
      return UPDATE_DEPTH_BREADTH_TRIE(T, L)
    if L is prefix of list then
      /* Optimization II */
      T := REMOVE((list, n_i), T)
      T := ADD((L, n_{i-(length(list)-length(L))}), T)
      list := REPLACE(list, L, n_{i-(length(list)-length(L))})
      T := ADD((list, n_i), T)
      return n_{i-(length(list)-length(L))}
    if L and list have a common prefix prefix with length(prefix) > 1 then
      /* Optimization III */
      n_j := UPDATE_DEPTH_BREADTH_TRIE(T, prefix)
      L := REPLACE(L, prefix, n_j)
      return UPDATE_DEPTH_BREADTH_TRIE(T, L)
  COUNTER := COUNTER + length(L)
  T := ADD ((L, n_COUNTER), T)
  return n_COUNTER
```

$[p_1, \ldots, p_i]$ with a new reference n_{r-1}, the other representing the existing formula, but modified to re-use the new reference n_{r-1}, i.e., $[p_1, \ldots, p_n]$ is replaced by $[n_{r-1}, p_{i+1}, \ldots, p_n]$.

Optimization III (Common Prefix): The last optimization exploits definitions branching off from each other. It occurs when a new formula $[p_1, \ldots, p_n]$ shares a common prefix $[p_1, \ldots, p_i]$, $n > i \geq 2$, with an existing formula $[p_1, \ldots, p_i, p'_{i+1}, \ldots, p'_m]$, $m > i$, with reference n_r. In this case, first, the common prefix is inserted as a new formula with reference n_{r-1}, triggering the second optimization, and second, the original new formula is added as $[n_{r-1}, p_{i+1}, \ldots, p_n]$ using n_{r-1} as in the first optimization.

Each repeated occurrence of a prefix of length P identified by one of the optimizations decreases the total number of operations required by $P - 1$. For example, if Optimization III identifies a common prefix f_P of length P of two formulae f_M and f_N of length M and N respectively, the number of operations decreases from $(N-1)+(M-1)$ to $(P-1)+(N-P)+(M-P) = N+M-P-1$, and if a third formula f_K shares the same prefix, the number of operations it requires again reduces to $(K-1)-(P-1) = K-P$.

Algorithm 2 formalizes the implementation of these three optimizations, which roughly speaking replaces the write and replace operations in Algorithm 1. One should notice that with the depth breadth trie, the references n_i are no longer incremented by one but by the length of the formula being added. This is necessary as Optimizations II and III insert additional subformulae that have to be created before the current formula being added, and thus need to be assigned a smaller reference. As our formulae always contain at least two elements, using the length of the formula to increment n_i is sufficient to leave enough free places for later use with Optimizations II and III. The order given by those references will therefore ensure that subformulae will be generated before being referred to. Moreover, as we are using tries, these optimizations can be performed while adding the new formulae. Note that the optimizations require a modification of the trie insertion procedure[4]: if the new definition first differs from an existing one after two or more steps, the insertion of the new formula is frozen while the appropriate optimization is performed and resumed afterwards.

To assess the effect of optimizations, our implementation in fact offers four choices of optimization level: no optimizations, Optimization I only, Optimizations I and II, or all three optimizations. Furthermore, the minimal length of common prefixes (2 by default) can be adapted. Note that depending on the order in which the formulae are inserted, different optimizations might trigger and the resulting trie might be slightly different.

5 Experimental Results

We next report on experiments comparing the four preprocessing methods: naive, decomposition (**dec**), recursive node merging as described in [6] (**rnm**) and recursive node merging with the depth breadth trie (**dbt**). The environment for our experiments was a C2Q 2.83 GHz 8 GB machine running Linux using a single core. The entire ProbLog engine, including preprocessing, is implemented in Yap Prolog 6.0, except for **dbt**, which is implemented in C as its optimizations require modifications to Yap's trie insertion, which is itself implemented in C. BDD construction uses the CUDD BDD package with automatic triggering of variable reordering by group sifting [10]. As **rnm** has been developed to exploit structure sharing in the trie, whereas **dec** is a general purpose method, we consider two benchmarks that contrast in this aspect.

The first benchmark is a three-state Markov model, where we query for the probability of an arbitrary sequence of N steps (starting in a random state at time point 0) ending in a given state. Each of the N time steps in such a sequence involves two new random variables (jointly encoding the three different start states of the step), and the number of proofs is thus 3^N.

[4] A definition is inserted term by term incrementally and each term is compared for identical prefix.

The second benchmark comes from the domain of connectivity queries in biological graphs that originally motivated ProbLog. In this case, we consider two different ProbLog inference methods which lead to different types of formulae. Exact inference as discussed in Section 2 produces formulae with high sharing in both prefixes and suffixes of proofs (which correspond to paths starting and ending at specific nodes). On the other hand, upper bound formulae as encountered in bounded approximation [6] typically contain small numbers of proofs and large numbers of so-called *stopped derivations*, i.e., partial proofs cut off at a probability threshold. While the latter still share prefixes, their suffixes are a lot more diverse. This type of formulae has been observed to be particularly hard for ProbLog. In our experiments, we use a graph with 144 edges (and thus 144 random variables) extracted from the Biomine network also used in [6], and query for acyclic paths between given pairs of nodes. We use a set of 45 different queries, some chosen randomly, and some maximizing the degrees of both nodes. In exact inference, the number of conjunctions in the DNF ranges from 13136 to 351600, with an average of 90127, for upper bound formulae, it ranges from 53 to 26085, with an average of 9516. The number of trie nodes in the exact case varies from 44479 to 1710621 (average 387073), representing between 161100 and 5776734 virtual nodes (average 1376030)[5]. In the upper bound case, tries have 126 to 60246 nodes (average 22251), corresponding to 334 to 232618 virtual nodes (average 80525).

Table 1. Average runtimes for BDD construction on three-state Markov model, for sequence length N. Cases with (/) exceeded the time limit for BDD construction, cases with (−) fail for memory reasons.

N	naive	dec	rnm	dbt
7	251	45	28	25
8	786	87	28	25
9	3,698	188	31	25
10	20,854	539	35	29
11	256,330	1,638	43	31
12	/	5,024	96	31
13	/	-	205	48
14	/	-	-	75

We set up experiments to study the following questions:

Q1: How do the different preprocessing methods compare on more, or less structured problems?

Q2: What is the impact of each optimization and which parameters affect them?

As the main goal of this work is to optimize the performance of BDD construction, we will focus on the runtime of this last step of ProbLog inference as central evaluation criterion. The time to calculate probabilities on the final BDD is included in the construction time; it typically is a very small fraction thereof.

Table 1 presents BDD construction times for the Markov model. In this benchmark, no extra optimizations are triggered for **dbt**. Times are given in milliseconds and are averages over three runs. BDD construction uses a timeout of 600 seconds, when a method reaches this timeout, it is not applied to larger problems. These cases are marked with (/). Cases marked with (-) fail due to

[5] The number of virtual nodes roughly corresponds to the number of occurrences of Boolean variables in the DNF written in uncompressed form. Each trie branch is represented by a node for each variable and by two special start and end nodes.

memory during script preprocessing; this also occurs when using **dbt** at length 15. In this experiment, both trie based methods clearly outperform the naive and **dec** methods, and BDD construction also seems to benefit slightly from the modifications used in the **dbt**-based version of **rnm**. As a first answer to **Q1**, we thus conclude that for structured problems, trie-based methods are indeed the first choice to optimize BDD construction.

For the graph domain, we use a timeout of 300 seconds on BDD construction, and a cutting threshold $\delta = 0.05$ for obtaining upper bound formulae. As the naive method always performs worst, it is excluded from the following discussion. Typically, all optimizations for the **dbt** method are triggered, with type I being most frequent (note that type III increases type I usage).

In exact inference, cf. Table 2, BDD construction times for all methods are very close and rarely exceed 4 seconds. However, preprocessing time for **dec** is one order of magnitude higher than for **rnm** (remember that **dbt** should not directly be compared, as it is implemented in a different language). Again, this is due to the high amount of suffix sharing in those tries, which is exploited by our method, but causes repeated work during construction for the decomposition method. These results clearly enforce our first conclusions about **Q1**. Regarding **Q2**, these results show that the **dbt** optimizations are incrementally effective in reducing construction time without introducing costs in preprocessing time.

For upper bound BDDs, the results are more diverse. Here, we focus on the comparison between **dec** and **dbt** with different optimiza-

Table 2. BDDs for exact inference: average and standard deviation of times over 45 queries (**dbt***i* uses all optimization levels $l \leq i$)

Method	Preprocessing		BDD Constr	
	avg	sdev	avg	sdev
dec	40,076	38,417	2,235	1,313
rnm	3,694	3,632	1,844	1,150
dbt	124	117	1,998	1,318
dbt1	125	118	1,891	1,697
dbt2	125	118	1,481	630
dbt3	128	120	1,446	769

Table 3. BDDs for upper bounds at threshold 0.05: average and standard deviation of times over 44 queries grouped into categories according to runtimes

Group	Method	BDD Constr		Time outs
		avg	sdev	
Easy 19/44	**dec**	9,351	2,323	0
	rnm	24,710	6,415	0
	dbt	10,148	2,192	0
	dbt1	10,714	2,389	0
	dbt2	14,417	3,263	0
	dbt3	15,311	4,055	0
Medium 14/44	**dec**	21,785	5,197	0
	rnm	46,428	9,084	0
	dbt	29,719	4,029	0
	dbt1	39,914	9,084	0
	dbt2	28,522	3,165	0
	dbt3	46,263	19,231	0
Hard 11/44	**dec**	28,979	9,172	0
	rnm	114,870	18,225	0
	dbt	62,612	16,350	3
	dbt1	121,442	29,052	2
	dbt2	94,454	28,753	3
	dbt3	122,150	37,751	3

tion levels. For presentation of results in Table 3, we partition queries in categories by using two thresholds on BDD construction time ($t < 15,000ms$ for **Easy**, $15,000ms \leq t < 50,000ms$ for **Medium** and $t \geq 50,000ms$ for **Hard**), and majority vote among the methods (as one single test query finishes in few milliseconds, it is omitted from the results). The last column gives the number of queries reaching the timeout in BDD construction.

Upper bound DNFs contain less conjunctions than those obtained in exact inference, and preprocessing times are one order of magnitude lower for all methods. BDD construction times, however, are generally higher when considering upper bounds. On average, BDDs obtained by **dec** have smaller construction times than those obtained from **rnm** and **dbt**, even though variation is high.

Table 4 compares methods by counting the number of queries for which they achieve fastest upper bound BDD construction compared to competing methods. As **dec** performs best in the overall comparison for this type of problem, we further compare the two implementations of recursive node merging. While the BDDs obtained from the implementation using depth breadth tries often outperform those from the previous implementation, there is no clear winner between the various optimization levels for this type of problem. Together, those results provide the second part of the answer to **Q1**: for problems with less suffix sharing, scripts obtained from **dec** often outperform those obtained from **dbt**.

Table 4. Upper bound BDDs: number of queries where a given method leads to fastest BDD construction, comparing all methods (**All**), the two implementations of recursive node merging without optimizations (**rnm/dbt**) or including different optimization levels (**rnm/dbt***), and different depth breadth trie optimization levels only (**dbt***)

Method	All	rnm/dbt	rnm/dbt*	dbt*
dec	26	-	-	-
rnm	2	14	6	-
dbt	6	30	13	16
dbt1	2	-	9	10
dbt2	5	-	9	9
dbt3	3	-	7	9

Concerning the optimization levels, Tables 3 and 4 indicate that for the graph case, their effect varies greatly. For all levels, we observe cases of improvement as well as deterioration. We suspect that optimizations are often performed too greedily. Initial experimentation on artificially created formulae indicates that several factors influence performance of optimizations, among which are: (i) the length of the shared prefix; (ii) the number of times it occurs; (iii) the structure of the subformulae occurring in the prefix; and (iv) the structure of the suffixes sharing the prefix. While the latter three are harder to control during preprocessing, we performed a first experiment where we only trigger the optimizations for shared prefixes of minimal length $n > 2$. Results on upper bound formulae confirm that this parameter indeed influences BDD construction, again to the better or the worse. We conclude that, while we identified certain parameters influencing the success of optimizations in synthetic data, in the case of less regular data, the answer to **Q2** remains an open issue for further investigation.

6 Conclusions and Future Work

We introduced depth breadth tries as a new data structure to improve preprocessing in ProbLog, and compared the resulting method and its variations to the method used so far as well as to the decomposition method presented by [7]. Our experiments with the three-state Markov model and with exact inference confirm that our trie based method outperforms the other methods on problems with high amount of suffix sharing between proofs. At the same time they reveal that the decomposition method is more suited if this is not the case, and thus is a valuable new contribution to ProbLog.

While the three new optimizations, aimed at reducing the number of BDD operations, can greatly improve performance in some cases, in others, they have opposite effects. Initial experiments suggest that those optimizations should be applied less greedily. Future work therefore includes a more in depth study of the factors influencing the effects of optimizations. We also plan to further investigate the respective strengths of our trie based approach and the decomposition method, and to exploit those in a hybrid preprocessing method. Finally, we need to further explore existing work on BDD construction in other fields, which might provide valuable insights for our specific application context.

Acknowledgments. T. Mantadelis is supported by the GOA/08/008 Probabilistic Logic Learning, A. Kimmig is supported by the Research Foundation Flanders (FWO Vlaanderen) and R. Rocha has been partially supported by the FCT research projects STAMPA (PTDC/EIA/67738/2006) and HORUS (PTDC/EIA-EIA/100897/2008).

References

1. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911. Springer, Heidelberg (2008)
2. Getoor, L., Taskar, B. (eds.): Statistical Relational Learning. The MIT press, Cambridge (2007)
3. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of ICLP, pp. 715–729 (1995)
4. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. JAIR 15, 391–454 (2001)
5. Poole, D.: The independent choice logic and beyond. In: [1], pp. 222–243
6. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of ProbLog programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 175–189. Springer, Heidelberg (2008)
7. Rauzy, A., Châtelet, E., Dutuit, Y., Bérenguer, C.: A practical comparison of methods to assess sum-of-products. Reliab. Eng. Syst. Safe 79(1), 33 – 42 (2003)
8. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
9. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
10. Panda, S., Somenzi, F.: Who are the variables in your neighborhood. In: Proceedings of ICCAD 1995, pp. 74–77 (1995)

Minimal Knowledge and Belief via Minimal Topology

David Pearce[1] and Levan Uridia[2]

[1] Universidad Politécnica de Madrid, Spain
david.pearce@upm.es
[2] Universidad Rey Juan Carlos, Madrid, Spain
uridia@ia.urjc.es

Abstract. We introduce and study a modal logic $wK4f$ that is related to the idea of *minimal belief* in much the same way as its strengthening, $S4F$, has been shown to be related to the idea of minimal knowledge. $wK4f$ can be obtained by adding a weakened version of axiom F to the modal logic $wK4$. We show that, like $S4F$, $wK4f$ is sound and complete with respect to the class of all minimal topological spaces ie topological spaces with only three open sets. We describe the rooted frames of $wK4f$ by quadruples of natural numbers. Finally we characterise non-monotonic $wK4f$ in terms of minimal models.

1 Introduction

In this paper we explore an approach to minimal belief that borrows the basic ideas of minimal knowledge studied by Schwarz and Truszczynski [8,9]. At the same time we show how the logics underlying minimal knowledge and belief are related to minimal topologies using the well-known methods for obtaining logics from topologies described by [19] and [2]. We start with a brief review of these basic ideas.[1]

Minimal Knowledge and Minimal Belief. The paradigm of minimal knowledge derives from the well-known work of Halpern and Moses, especially [3], later extended and modified in works such as [15,13,14] and others. Many approaches are based on Kripke-$S5$-models with a universal accessibility relation and the minimisation of knowledge is represented by maximising the set of possible worlds with respect to inclusion. In general, this has the effect of minimising objective knowledge, ie knowledge of basic facts and propositions. A somewhat different approach was developed by Schwarz and Truszczynski [8] and can be seen as a special case of the very general method of Shoham [15] for obtaining different concepts of minimality by changing the sets of models and preference relations between them. The initial models considered by [8] (see also [22,9]) consist of not one ($S5$) cluster but rather two clusters arranged in such a way that all worlds in one cluster are accessible from all worlds in the other (but not vice versa). In Figure 1 clusters are labelled W_1, W_2, all points are reflexive and

[1] The authors are grateful to anonymous reviewers whose comments helped to improve the readability of the paper. The second author is grateful to Leo Esakia for discussions on monotonic $wK4f$ and its connections with topology and also to David Gabelaia due to whom the axiom f is much more readable. This research has been partially supported by the MCICINN projects TIN2006-15455, TIN2009-14562-CO5, and CSD2007-00022.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 273–285, 2010.
© Springer-Verlag Berlin Heidelberg 2010

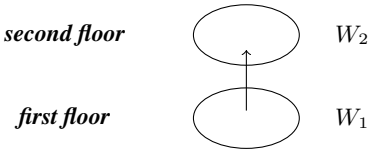


Fig. 1.

every point in W_2 is accessible from every point in W_1. We call W_1, W_2 respectively the *first* and the *second floor* of the model. The former may be empty but the latter not.

In [8,9], given a background theory or knowledge set I, minimal knowledge (with respect to I) is captured by an $S5$-model of the theory, say M, but now the idea of minimality is that there should be no two-floor model M' as in Figure 1 of the same theory I, where M coincides with the restriction of M' to the second floor W_2, and W_1 is smaller in the sense that it fails to verify some objective (non-modal) sentence true in M. Schwarz and Truszczynski argue that this approach to minimal knowledge has some important advantages over the method of [3] and they study its properties in depth, in particular showing that while the two-floor models correspond to the modal logic $S4F$ first studied by Segerberg [5], minimal knowledge is precisely captured by non-monotonic $S4F$. In [9] they show that non-monotonic $S4F$ captures, under some intuitive encodings, several important approaches to knowledge representation. They include disjunctive logic programming under answer set semantics [10], (disjunctive) default logic [11], [12], the logic of grounded knowledge [13], the logic of minimal belief and negation as failure [14] and the logic of minimal knowledge and belief [9]. Recently, Truszczynski [16] and Cabalar [17] have revived the study of $S4F$ in the context of a general approach to default reasoning.

Logics via Topology. Alfred Tarski [6], together with Chen McKinsey [19,20], laid the foundations for the algebraic and topological study of intuitionistic and modal logics. The basic idea, recalled and developed in a recent paper by Leo Esakia [2], is that from an arbitrary topological space X we can generate three different algebraic structures each giving rise to different logical systems.[2] By considering the algebra of open sets, $Op(X)$, one is led to the well-known Heyting algebra that forms a semantical basis for intuitionistic logic. By considering the closure algebra, $(\mathcal{P}(X), \mathbf{c})$ one is led to the modal system $S4$.[3]

The third path from topology to logic is via what are known as *derivative algebras*, $(\mathcal{P}(X), der)$. These are Boolean algebras with a unary operation der representing topological derivation: if A is a subset of X then $der(A)$ is the set of all accumulation or limit points of A. The derivative algebra $(P(X), der)$ gives rise to the modal logic

[2] For the basic notions of topology see eg [4] or any appropriate textbook.

[3] Recall that a Heyting algebra $(\mathbf{H}, \vee, \wedge, \rightarrow, \bot)$ is a distributive lattice with smallest element $\bot$ containing a binary operation $\rightarrow$ such that $x \leq a \rightarrow b$ iff $a \wedge x \leq b$. $(\mathbf{B}, \vee, \wedge, -, \mathbf{c})$ is a closure algebra if $(\mathbf{B}, \vee, \wedge, -)$ is a Boolean algebra and $\mathbf{c}$ is a closure operator satisfying: $a \leq \mathbf{c}a, \mathbf{cc}a = \mathbf{c}a, \mathbf{c}(a \vee b) = \mathbf{c}a \vee \mathbf{c}b, \mathbf{c}\bot = \bot$.

$wK4$, a slightly weaker version of the logic $K4$, that was first studied from a topological point of view in [1] (see [2] for a detailed overview).

All three paths to logic are of interest for the modelling of agents' reasoning, their knowledge and beliefs in AI. Intuitionistic logic and its extensions capture different forms of constructive reasoning, while extensions of $S4$, including $S5$, have formed the basis for epistemic logics of knowledge. On the other hand, extensions of $wK4$ may be considered good candidates for doxastic logics of belief inasmuch as the axiom $\Box p \rightarrow p$ does not hold. In fact, the standard doxastic logic $KD45$ is one such extension of $wK4$.

From the viewpoint of non-monotonic reasoning, there is a special interest in examining the logics that arise as above from topological spaces X that are *minimal*, that is where X has only three open sets. In the first case, we obtain the three-element Heyting algebra that captures a logic known as *here-and-there*, HT, the maximal intermediate logic that is properly contained in classical logic. The well-known non-monotonic extension of HT called *equilibrium logic* [18] provides a logical foundation for reasoning with the stable model semantics of logic programs and thus for the popular approach to knowledge representation and declarative problem solving known as *answer set programming*, ASP. Starting from a minimal topological space and using instead the idea of closure algebras one arrives at $S4F$, a reflexive normal modal logic first studied by Segerberg [5]. We have already observed how non-monotonic $S4F$ relates to minimal knowledge and is important in knowledge representation and reasoning.

In this paper we study the third path from topology to logic based on minimal topological spaces. This yields a logic that we call $wK4f$. Our main motivation is that this logic (and some close variants) can serve to model minimal belief, in the same way as $S4F$ captures minimal knowledge. We would like to emphasise again that these ideas of minimal knowledge and belief are based on properties of models (and what they verify) and not on the shape of modal axioms. While obtaining a complete axiomatisation is therefore an important and even essential part of our study, it is not the axioms themselves that motivate the choice of logic. They provide a compact formulation of a calculus rather than a direct formalisation of some or other intuitive property of belief. The connections with minimal belief will be explored in the second half of the paper, in Section 5, while the first parts of the paper are devoted to the study of $wK4f$ itself.

As we have seen, $S4F$ is captured by Kripke frames consisting of two clusters connected by an accessibility relation. In the case of $wK4f$ the picture is similar except that we drop the condition of reflexivity on frames: in Fig. 2 some points in W_1, W_2 may now be irreflexive (where i and r label this difference). Since $S4F$ and $wK4f$ are closely related, many results can be transferred from one to the other.

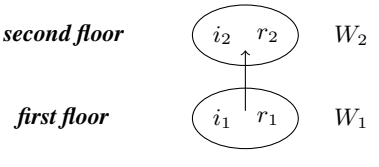


Fig. 2.

The paper is organised in the following way. In section 2 we present the syntax and Kripke semantics of $wK4f$. We prove completeness and the finite model property. In section 3 we characterise finite one-step, weakly-transitive frames and their bounded morphisms in terms of quadruples of natural numbers. In section 4 we prove a main theorem of the paper, which states that $wK4f$ is the (sound and complete) logic of all minimal topological spaces. Section 5 describes non-monotonic $wK4f$ and relates it to the idea of minimal belief. In the last section we state some conclusions and mention topics for future work.

2 The Modal Logic $wK4f$

Following the Tarski/McKinsey suggestion to treat modality as the derivative of the topological space [19], Esakia in [1,21] introduced $wK4$ as the modal logic of all topological spaces, with the desired (derivative operator) interpretation of the modal $\Diamond$. $wK4f$ is a normal modal logic obtained by adding the axiom weak-F to the modal logic $wK4$. $wK4f$ is a weaker logic than $S4F$ discussed in Segerberg [5] since it doesn't satisfy the axiom T. However since the frames of $wK4f$ and $S4F$ are closely related, some results about $wK4f$ may carry over to $S4F$.

Syntax. The normal modal logic $wK4f$ is defined in a basic modal language with an infinite set $Prop$ of propositional letters and connectives $\vee, \wedge, \neg, \Box$. The axioms are all classical tautologies plus the axioms listed below. Rules of inference are: modus ponens, substitution and necessitation.

$$\begin{aligned} K &: \ \Box(p \to q) \to (\Box p \to \Box q) \\ w4 &: \ \Box p \wedge p \to \Box\Box p \\ f &: \ p \wedge \Diamond(q \wedge \Box\neg p) \to \Box(q \vee \Diamond q) \end{aligned}$$

Semantics. Kripke semantics for the modal logic $wK4f$ is provided by frames which have in a weak sense height at most two and which do not allow forking. This is made precise in the following two definitions.

Definition 1. *We will say that a relation $R \subseteq W \times W$ is weakly-transitive if* $(\forall x, y, z)$ $(xRy \wedge yRz \wedge x \neq z \Rightarrow xRz)$.

Clearly every transitive relation is weakly-transitive as well. Moreover, clusters (where a cluster is a subset of a frame where every two distinct points are related with each other) with weakly transitive relations differ from those with transitive relations in that they allow for irreflexive points. Such clusters will be called *weak-clusters*.

Definition 2. *We will say that a relation $R \subseteq W \times W$ is a one-step relation if the following two conditions are satisfied:*

1)$(\forall x, y, z)((xRy \wedge yRz) \Rightarrow (yRx \vee zRy))$,

2)$(\forall x, y, z)((xRy \wedge \neg(yRx) \wedge xRz \wedge y \neq z) \Rightarrow zRy)$.

As the reader can see the first condition restricts the 'strict' height of the frame to two. Where informally by "strict" we mean that the steps are not counted *within* a cluster. The second condition is more complicated. Essentially it restricts the 'strict' width of the frame to one, though more points could be allowed at the bottom.

We briefly recall some standard definitions in modal logic. The pair (W, R), with W an arbitrary set and $R \subseteq W \times W$ is called a Kripke frame. If we additionally have a third component, a function $V : Prop \times W \rightarrow \{0, 1\}$, then we say that we have a Kripke model $\mathcal{M} = (W, R, V)$.

For a given Kripke model $\mathcal{M} = (W, R, V)$ the satisfaction of a formula at a point $w \in W$ is defined inductively as follows: $\mathcal{M}, w \Vdash p$ iff $V(p, w) = 1$, the Boolean cases are standard, $\mathcal{M}, w \Vdash \Box\phi$ iff $(\forall v \in W)(wRv \Rightarrow v \Vdash \phi)$. A formula ϕ is valid in a model $\mathcal{M}$ if for every point $w \in W$ we have $\mathcal{M}, w \Vdash \phi$ in this case we write $\mathcal{M} \Vdash \phi$. A formula is valid in a frame if it is valid in every model based on the frame. A formula is valid in a class of frames if it is valid in every frame in the class. Notice that from the definition of $\Diamond$, $(\Diamond\phi \equiv \neg\Box\neg\phi)$ it easily follows that $\mathcal{M}, w \Vdash \Diamond\phi$ iff $\exists v \in W$ such that $wRv \wedge v \Vdash \phi$.

Let $\mathcal{K}$ denote the class of all one-step and weakly-transitive Kripke frames . The following theorem links the logic $wK4f$ with the class $\mathcal{K}$. The proof uses standard modal logic completeness techniques, so we will not enter into all the details.

Theorem 3. *The modal logic $wK4f$ is sound and strongly complete wrt the class $\mathcal{K}$.*

We give the soundness proof only for the axiom f. For the proofs for other axioms the reader may consult [21].

Proof. Take an arbitrary, weakly-transitive, one-step model (W, R, V). Assume at some point $w \in W$ it holds that $w \Vdash p \wedge \Diamond(q \wedge \Box\neg p)$. This implies that $w \Vdash p$ and there exists w' such that wRw', $w' \Vdash q$ and it is not the case that $w'Rw$ (as far as $w' \Vdash \Box\neg p$). Now for an arbitrary v with wRv and $v \neq w'$, by the second condition of definition 2, we have that vRw', which implies that $v \Vdash \Diamond q$ and hence $w \Vdash \Box(q \vee \Diamond q)$.

For strong completeness assume $I \nvdash \phi$. We will construct the one-step and weakly-transitive model $M^c = \{W^c, R^c, V^c\}$ such that $M^c \Vdash I$ and $M^c \nVdash \phi$. For M^c we take a standard canonical model ie. $W^c = \{\Gamma | \Gamma \vdash I \text{ and } \Gamma \text{ is a maximal consistent set}\}$. The relation is defined in a standard way $\Gamma R^c \Gamma'$ iff $(\forall\alpha)(\Box\alpha \in \Gamma \Rightarrow \alpha \in \Gamma')$ and $V^c(p, \Gamma) = 1$ iff $p \in \Gamma$.

Lemma 4 (Truth Lemma). *For any formula ϕ we have $M^c, \Gamma \Vdash \phi$ iff $\phi \in \Gamma$.*

The proof follows a standard pattern found in modal logic textbooks. As $I \nvdash \phi$ we have that $I \cup \{\neg\phi\}$ is consistent, so there exists a maximally consistent set $\Gamma_{\neg\phi}$ containing $I \cup \{\neg\phi\}$ and by the truth lemma this means that $M^c, \Gamma_{\neg\phi} \Vdash \neg\phi$ which completes the proof. The main thing to be checked is that $M^c \in \mathcal{K}$. For weak transitivity of the relation R^c the reader may consult [21]. Let us show that R^c is a one-step relation.

First let us show that R^c satisfies the first condition of definition 2. For the contradiction assume there exist three distinct points $\Gamma, \Gamma', \Gamma'' \in W^c$ such that $\Gamma R^c \Gamma' \wedge \Gamma' R^c \Gamma''$ and $\neg(\Gamma' R^c \Gamma) \wedge \neg(\Gamma'' R^c \Gamma')$. This means that there is a formula ψ such that $\Box\psi \in \Gamma'$ and $\neg\psi \in \Gamma$ and there is a formula ϕ such that $\Box\phi \in \Gamma''$ and $\neg\phi \in \Gamma'$ and as $\Gamma' \neq \Gamma''$

there exists a formula γ with $\gamma \in \Gamma'$ and $\neg\gamma \in \Gamma''$. From these assumptions we have that $(\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi \in \Gamma'$. Now as $\Gamma R^c \Gamma'$ we have that $\Diamond((\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi) \in \Gamma$ and as $\neg\psi \in \Gamma$ we have that $\Diamond((\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi) \wedge \neg\psi \in \Gamma$. Applying axiom f (with $p = \neg\psi, q = \neg\phi \wedge \gamma$) we get that $\Box((\neg\phi \wedge \gamma) \vee \Diamond(\neg\phi \wedge \gamma)) \in \Gamma$. Hence as $\Gamma R^c \Gamma''$ (because of weak transitivity) we have that $(\neg\phi \wedge \gamma) \vee \Diamond(\neg\phi \wedge \gamma) \in \Gamma''$. On the other hand $\Diamond(\neg\phi \wedge \gamma) \notin \Gamma''$ since $\Box\phi \in \Gamma''$ and $\neg\phi \wedge \gamma \notin \Gamma''$ because $\neg\gamma \in \Gamma''$. Hence we get a contradiction.

Now let us show that R^c satisfies the second condition of definition 2. Again the proof is by contradiction. Assume there exist three distinct points $\Gamma, \Gamma', \Gamma'' \in W^c$ such that $\Gamma R^c \Gamma' \wedge \Gamma R^c \Gamma'' \wedge \neg(\Gamma' R^c \Gamma) \text{ and } \neg(\Gamma'' R^c \Gamma')$. This means that there is a formula ψ such that $\Box\psi \in \Gamma'$ and $\neg\psi \in \Gamma$ and there is a formula ϕ such that $\Box\phi \in \Gamma''$ and $\neg\phi \in \Gamma'$ and as $\Gamma' \neq \Gamma''$ there exists a formula γ with $\gamma \in \Gamma'$ and $\neg\gamma \in \Gamma''$. From these assumptions we have that $(\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi \in \Gamma'$. Now as $\Gamma R^c \Gamma'$ we have that $\Diamond((\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi) \in \Gamma$ and as $\neg\psi \in \Gamma$ we have that $\Diamond((\neg\phi \wedge \gamma) \wedge \Box\neg\neg\psi) \wedge \neg\psi \in \Gamma$. Applying axiom f we get that $\Box((\neg\phi \wedge \gamma) \vee \Diamond(\neg\phi \wedge \gamma)) \in \Gamma$. Hence as $\Gamma R^c \Gamma''$ we have that $(\neg\phi \wedge \gamma) \vee \Diamond(\neg\phi \wedge \gamma) \in \Gamma''$. On the other hand $\Diamond(\neg\phi \wedge \gamma) \notin \Gamma''$ since $\Box\phi \in \Gamma''$. Nor can we have $\neg\phi \wedge \gamma \in \Gamma''$ because $\neg\gamma \in \Gamma''$. Hence we get a contradiction.

Theorem 5. *The modal logic $wK4f$ is sound and complete wrt the class of all **finite** one-step and weakly-transitive Kripke frames.*

3 Finite, Rooted, Weakly-Transitive and One-Step Kripke Frames

We saw from (Theorem 5), which we do not prove in this paper, that the class of finite, weakly-transitive and one-step Kripke frames fully captures the modal logic $wK4f$. From general theorems in modal logic it is well known that this class can be reduced to a smaller class of frames which are rooted so that the completeness theorem still holds. In this section we characterise finite, rooted, weakly-transitive and one-step Kripke frames in terms of quadruples of natural numbers.

Definition 6. *The upper cone of a set $A \subseteq W$ in a weakly-transitive Kripke frame (W, R) is defined as a set $R(A) = \bigcup\{y : x \in A \& xRy\} \cup A$.*

Observe that the general definition of upper cone in an arbitrary Kripke frame is given in terms of the reflexive, transitive closure of a relation, while Definition 6 is a simplified version for the particular case of weakly transitive frames.

Definition 7. *A Kripke frame (W, R) is called* rooted *if there exists a point $w \in W$ such that the upper cone $R(\{w\}) = W$; w is called the root of the frame.*

Let N^4 be the set of all quadruples of natural numbers and let $\mathcal{N}^4 = N^4 - \{(n, m, 0, 0) | n, m \in N\}$. The following theorem states that the set $\mathcal{K}_r$ of all finite, rooted, one-step, weakly-transitive frames considered up to isomorphism can be seen as the set $\mathcal{N}^4$.

Theorem 8. *There is a one-to-one correspondence between the set $\mathcal{K}_r$ and the set $\mathcal{N}^4$.*

Proof. We know that any one-step frame has "strict" width one and "strict" height less than or equal to two (We didn't give the formal definition of "strict" height and width, but it should be clear from the intuitive explanation after the definitions 2 and 1 what we mean by this). If additionally we have that the frame is rooted, the case where strict width is greater than one at the bottom is also restricted. It is not difficult to verify that any such frame (W, R) is of the form (W_1, W_2), where $W_1 \cup W_2 = W$, $W_1 \cap W_2 = \varnothing$ and $(\forall u \in W_1, \forall v \in W_2)(uRv)$. Besides because of the weak-transitivity, we have that $(\forall u, u' \in W_1)(u \neq u' \Rightarrow uRu')$ and the same for every two points $v, v' \in W_2$. Pictorially any rooted, weak-transitive and one-step Kripke frame can be represented as in Figure 2. Again we call W_1 the ***first floor*** and W_2 the ***second floor*** of the frame (W, R). Notice that W_1 or W_2 may be equal to $\varnothing$ ie the frame has only one floor. In this case we treat the only floor of the frame as the second floor.

Now let us describe how to construct the function from $\mathcal{K}_r$ to $\mathcal{N}^4$. With every frame $(W, R) \in \mathcal{N}^4$ we associate the quadruple (i_1, r_1, i_2, r_2), where i_1 is the number of irreflexive points in W_1, r_1 is the number of reflexive points in W_1, i_2 is the number of irreflexive points in W_2 and r_2 is the number of reflexive points in W_2. We will call the quadruple (i_1, r_1, i_2, r_2) the ***characteriser*** of the frame (W, R). In case the frame (W, R) has only one floor, by our earlier remark it is treated as the frame $(\varnothing, W)$. Hence its characteriser has the form $(0, 0, i, r)$. Now it is clear that the correspondence described above defines a function from the set $\mathcal{K}_r$ to the set $\mathcal{N}^4$. We denote this function by Ch.

Claim 1: Ch is injective. Take any two distinct finite, rooted, weakly-transitive, one-step Kripke frames (W, R) and (W', R'). That they are distinct in $\mathcal{K}_r$ means that they are non-isomorphic ie either $|W| \neq |W'|$ or $R \not\simeq R'$. In the first case it is immediate that $Ch(W, R) \neq Ch(W', R')$ since $|W| = i_1 + i_2 + r_1 + r_2$. In the second case we have three subcases:

1) $|W_1| \neq |W_1'|$. In this subcase $i_1 + r_1 \neq i_1' + r_1'$ and hence $Ch(W, R) \neq Ch(W', R')$.

2) The number of reflexive (irreflexive) points in $|W_1|$ differs from the number of reflexive (irreflexive) points in $|W_1'|$. In this subcase $i_1 \neq i_1'$ and again $Ch(W, R) \neq Ch(W', R')$.

3) The number of reflexive (irreflexive) points in $|W_2|$ differs from the number of reflexive (irreflexive) points in $|W_2'|$. This case is analogous to the previous one.

It is straightforward to see that if none of these cases above occur ie $|W| = |W'|$, $|W_1| = |W_1'|$, $|\{w | w \in W_1 \wedge wRw\}| = |\{w' | w' \in W_1' \wedge w'R'w'\}|$ and $|\{w | w \in W_2 \wedge wRw\}| = |\{w' | w' \in W_2 \wedge w'R'w'\}|$ then (W, R) is isomorphic to $(W'R')$ and hence $(W, R) = (W', R')$ in $\mathcal{K}_r$.

Claim 2: Ch is surjective. Take any quadruple $(i_1, r_1, i_2, r_2) \in \mathcal{N}^4$. Let us show that the pre-image $Ch^{-1}((i_1, r_1, i_2, r_2))$ is not empty. Take the frame $(W, R) = (W_1, W_2)$, where $|W_1| = i_1 + r_1$, $|W_2| = i_2 + r_2$, W_1 contains i_1 irreflexive and r_1 reflexive points and $|W_2|$ contains i_2 irreflexive and r_2 reflexive points. Then by the definition of Ch, we have that $Ch(W, R) = (i_1, r_1, i_2, r_2)$.

4 Connection with Minimal Topological Spaces

In this section we show that $wK4f$ is the modal logic of minimal topological spaces. A topological space is minimal if it has only three open sets. It is well known that there is a bijection between Alexandrof spaces and weakly-transitive, irreflexive Kripke frames and this bijection preserves modal formulas. In this section we show that the special case of this correspondence for minimal topological spaces gives one-step, irreflexive and weakly-transitive relations as a counterpart. As a corollary it follows that the logic $wK4f$ is sound and complete wrt the class of minimal topological spaces.

Theorem 9. *There is a one-to-one correspondence between the class of all* ***irreflexive****, weakly-transitive, finite, rooted, one-step Kripke frames and the class of all finite minimal topological spaces.*

Proof. Assume (W, R) is a finite, rooted, irreflexive, weakly-transitive and one-step relational structure. (Note that as the frame is irreflexive its characteriser has the form $(i_1, 0, i_2, 0)$, where $i_1 + i_2 = |W|$.) Let W_1 be the first floor and W_2 the second floor of the frame, then the topology we construct is $\{W, \varnothing, W_2\}$. It is immediate that the space (W, Ω_R), where $\Omega_R = \{W, \varnothing, W_2\}$, is a minimal topological space.

Let us show that the correspondence we described is injective. Take two arbitrary distinct irreflexive, finite, rooted, weakly-transitive frames (W, R) and (W', R'). As they are distinct, either $W \neq W'$ or $R \neq R'$. In the first case it is immediate that $(W, \Omega_R) \neq (W', \Omega_{R'})$. In the second case as both R and R' are irreflexive the second floors are not the same, so $W_2 \neq W_2'$ and hence $\Omega_R \neq \Omega_{R'}$.

For surjectivity take an arbitrary minimal topological space (W, Ω), where $\Omega = \{W, \varnothing, W_0\}$ for some subset $W_0 \subseteq W$. Take the frame (W, R), where $R = (W_0 \times W_0 - \{(w, w) | w \in W_0\}) \cup (-W_0 \times -W_0 - \{(w, w) | w \in -W_0\}) \cup \{(w, w') | w \in -W_0, w' \in W_0\}$. In words every two distinct points are related in W_0 by R and the same in the complement $-W_0 = W - W_0$, besides every point from the $-W_0$ is related to every point from W_0. What we get is the rooted one-step relation which is weakly-transitive, with the second floor equal to W_0. As we didn't allow wRw for any point $w \in W$, the relation R is also irreflexive.

We now give the definition of a derived set (or set of accumulation points) of a set in a topological space. This definition is needed to give the derived set semantics of modal formulas in an arbitrary topological space.

Definition 10. *Given a topological space (W, Ω) and a set $A \subseteq W$ we will say that $w \in W$ is an* accumulation point *of A if for every neighborhood U_w of w the following holds: $U_w \cap A - \{w\} \neq \varnothing$. The set of all accumulation points of A will be denoted by $der(A)$ and will be called the* derived *set of A.*

Below we give the definition of satisfaction of modal formulas.

Definition 11. *A topological model (W, Ω, V) is a triple, where (W, Ω) is a topological space and $V : Prop \to P(W)$ is a valuation function. Satisfaction of a modal formula in a topological model (W, Ω, V) at a point $w \in W$ is defined by:*

$$w \Vdash p \quad \textit{iff} \quad w \in V(p) \; ; \; w \Vdash \Diamond p \quad \textit{iff} \quad w \in der(V(p)),$$

Boolean cases are standard. Validity of a formula in a topological space and class of topological spaces is defined in a standard way

Fact 12 *Let (W, R) be a finite, weakly-transitive and irreflexive frame and let (W, Ω_R) be its Alexandrof space. For every modal formula α the following holds:*

$$(W, R) \Vdash \alpha \text{ iff } (W, \Omega_R) \Vdash \alpha.$$

Note that here $\Vdash$ on the left hand side denotes the validity in Kripke frames while on the right hand side it denotes the validity in topological frames in derived set semantics.

Theorem 13. *The modal logic $wK4f$ is sound and complete with respect to the class of all minimal topological spaces.*

Proof. Soundness can be checked directly so we do not prove it here. For completeness assume $\nvdash \phi$. By theorem 5 there exists a finite, one-step, weakly-transitive frame (W, R) which falsifies ϕ. Assume that $Ch(W, R) = (i_1, r_1, i_2, r_2)$. It is not difficult to check that (W, R) is a p-morphic image of (W', R'), where $(W', R') = Ch^{-1}(i_1 + 2 \times r_1, 0, i_2 + 2 \times r_2, 0)$. Roughly speaking the main idea here is that two distinct irreflexive points from first floor (second floor) of (W', R') are mapped to one reflexive point of first floor (second floor) of (W, R). So each reflexive point in (W, R) has two irreflexive preimages and each irreflexive point in (W, R) has one irreflexive point as a preimage. Now as you can see on each floor in (W', R') there are enough irreflexive points to cover both reflexive and irreflexive point of the corresponding floor in (W, R). So we have a surjection. To check that the described function satisfies back and forth conditions of the p-morphism is left to the reader. The surjection implies that $(W', R') \nVdash \phi$. Now as far as (W', R') is irreflexive, the result immediately follows from theorem 9, and the fact 12.

5 Minimal Belief and Non-Monotonic $wK4f$

It is often held that $KD45$ represents an adequate logic for belief. One motivation for this is that it allows positive and negative introspection and additionally $\Box p \rightarrow p$ is not derivable in the logic. A Kripke model $\mathcal{M}$ for $KD45$ consists of cluster W plus one irreflexive point w so that w is related to every point in W but no point in W is related to w. In other words the first floor of $\mathcal{M}$ is one irreflexive point and the second floor is a cluster. The belief set of an agent is obtained as a theory of the second floor. Indeed $\varphi \in Th(W)$ iff $\mathcal{M} \Vdash \Box\varphi$.[4] In particular minimisation is relative to some base set I of beliefs. The aim is to capture a set which contains I, is closed under positive and negative introspection, is closed under logical deduction and does not contain anything superfluous. One way to obtain such a set is to consider a $KD45$ Kripke model $\mathcal{M}$ such that $v \Vdash I$ for every $v \in W$ and extend the cluster W by adding points which still make I true. As a result the set of objective facts true in every world will be reduced while the starting beliefs I will stay unchanged. This approach is applied in [3] to knowledge sets, but as discussed in [9] it has some unintuitive consequences. For this reason we

[4] Recall that the idea of minimisation applies to the belief set of an agent.

follow the pattern of [9] which relies on the idea that an agent's belief is dependent not only on the objective facts but also on the things that are believed by agent. More concretely we minimise the belief set by adding worlds on the first floor of the model $\mathcal{M}$ leaving the second floor untouched. This form of minimal model semantics provides an alternative way of minimising belief and I-expansions for $wK4f$ are exactly minimal belief sets. This is the chief motivation for considering non-monotonic $wK4f$ to be a good candidate for the logic of minimal belief.

Formally we want to relate non-monotonic $wK4f$ to the idea of *minimal model* introduced and characterised in [22]. However we cannot directly apply the general result (Theorem 3.1) of [22] since that theorem refers to what are called *cluster-closed* logics. Instead we can adapt Schwarz's techniques to our case, starting with the definition of preferred model for a class $\mathcal{K}$. The preference relation is between one-floor $S5$-models and two-floor models and only two-floor models can be preferred over $S5$-models. For example we can not compare two one-floor models with each other.

Definition 14. *We say that a two-floor model $\mathcal{N} = (N, S, U)$ is preferred over $S5$-model $\mathcal{M} = (W, R, V)$ if:*
a) There is a propositional formula ψ such that $\mathcal{M} \Vdash \psi$ and $\mathcal{N} \nVdash \psi$,
b) (W, R) is the second floor of (N, S) and V equals to the restriction of U to the second floor. Briefly, $\mathcal{M}$ is the model which is obtained by deleting the first floor in $\mathcal{N}$.

We next define the notion of minimal model that is central for the semantics of non-monotonic modal logics.

Definition 15. *An $S5$-model $\mathcal{M} = (W, R, V)$ is called a $\mathcal{K}$-minimal model for the set of formulas I if $\mathcal{M} \Vdash I$ and for every preferred model $\mathcal{N} \in \mathcal{K}$ we have $\mathcal{N} \nVdash I$.*

Non-monotonic $wK4f$ does not fit the scope of Theorem 3.1 [22] because the class $\mathcal{K}$ which characterises monotonic $wK4f$ is not cluster closed. In particular some two-floor models in $\mathcal{K}$ may not have a cluster as a maximum. On the other hand every model in $\mathcal{K}$ has a maximal *weak-cluster* (that is a cluster where irreflexive points are allowed or more precisely it is a rooted, symmetric, weakly-transitive frame). For this reason we need to consider weak-cluster closed classes.

Definition 16. *Let $\mathcal{N} = (N, S, U)$ be a Kripke model. A nonempty set $W \subseteq N$ is called a final weak-cluster if:*
a) W is an upper cone (def. 6),
b) W is weak-cluster,
c) For every $v \in N - W$ and for every $w \in W$, vRw.

It is immediate from Definition 16 and from Theorem 8 that every rooted, weakly-transitive, one-step frame has a final weak-cluster and it is the second floor (or the only floor) of the frame.

Definition 17. *Let $\mathcal{N} = (N, S, U)$ be a Kripke model and let N_2 be its final weak-cluster. Let $\mathcal{M} = (W, R, V)$ be a cluster. By cluster substitution of $\mathcal{M}$ in $\mathcal{N}$ we mean the model $< (N - N_2) \cup W, S', V' >$, where for each $w, v \in (N - N_2) \cup W, wS'v$ if and only if wSv or $v \in W$ and V' agrees with U on $(N - N_2)$ and agrees with V on W. In other words we substitute the cluster W instead of the weak-cluster N_2 into $\mathcal{N}$.*

Definition 18. *By the concatenation of two models* (W, R, V) *and* (N, S, U) *with* $W \cap N = \varnothing$ *we mean the model* $(N \cup W, S \cup N \times W \cup R, U \cup V)$.

Definition 19. *Let* C *be a class of models. We say that* C *is weak-cluster closed if* C *contains all weak-clusters and for each* $\mathcal{N} \in C$*, at least one of the following two conditions holds: the concatenation of* $\mathcal{N}$ *and each cluster belongs to* C*, or* $\mathcal{N}$ *has a final weak-cluster and for each* $S5$*-model* $\mathcal{M}$*, the cluster substitution of* $\mathcal{M}$ *in* $\mathcal{N}$ *belongs to* C.

It is immediate that $\mathcal{K}$ is weak-cluster closed. As usual, non-monotonic modal logics are defined via the notion of *expansion*.

Definition 20. *Let* L *be a modal logic. A set of formulas* T *is said to be an* L*-expansion of a set of formulas* I *if* $T = Cn_L(I \cup \{\neg\Box\varphi : \varphi \notin T\})$.

where Cn_L denotes consequence in L. Now we are ready to prove the main theorem of this section.

Theorem 21. *Let* $\mathcal{M} = (W, R, V)$ *be an* $S5$*-model, and* $T = \{\phi | \mathcal{M} \Vdash \phi\}$*. Then* T *is an* $wK4f$*-expansion of* I *if and only if* $\mathcal{M}$ *is a* $\mathcal{K}$*-minimal model of* I.

Proof. Assume T is a $wK4f$-expansion for I. This means that $T = Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}]$, where L stands for $wK4f$. For the contradiction assume $\mathcal{M}$ is not minimal. This means that there is a $wK4f$-model $\mathcal{N} = (N, S, U)$ such that $\mathcal{N}$ is preferred over $\mathcal{M}$ and $\mathcal{N} \Vdash I$. That $\mathcal{N}$ is preferred over $\mathcal{M}$ means that there is a propositional formula α such that $\mathcal{M} \Vdash \alpha$ while $\mathcal{N} \nVdash \alpha$. Now take an arbitrary formula $\psi \notin T$. Since T is an expansion we have that $\Diamond\neg\psi \in T$ hence $\mathcal{M} \Vdash \Diamond\neg\psi$. Hence there is at least one point $w \in W$ with $w \Vdash \neg\psi$ and hence for every point y in the first floor of N we have $y \Vdash \Diamond\neg\psi$ which yields that $\mathcal{N} \Vdash \Diamond\neg\psi$. So we get that $\mathcal{N} \Vdash I \cup \{\neg\Box\phi | \phi \notin T\}$ and hence $\mathcal{N} \Vdash T$ which is a contradiction since $\alpha \in T$.

For the other direction assume $\mathcal{M}$ is $\mathcal{K}$-minimal for I. That $Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}] \subseteq T$ follows directly from the fact that $\mathcal{M} \Vdash I$ and if $\mathcal{M} \nVdash \psi$ then there exists at least one point $w \in W$ with $w \Vdash \neg\psi$ and since R is a universal relation, $\mathcal{M} \Vdash \Diamond\neg\psi$.

For the other inclusion we show that for every rooted weakly-transitive and one-step model $\mathcal{N} = (N, S, U)$ the following holds:

$$(*) \quad \mathcal{N} \Vdash Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}] \Rightarrow \mathcal{N} \Vdash T.$$

This by Theorem 3 will imply that $Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}] \vdash T$ in $wK4f$ and, as the left side is closed under consequence, we get that $T \subseteq Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}]$. Now let us prove the star.

Assume $\mathcal{N} \Vdash Cn_L[I \cup \{\neg\Box\phi | \phi \notin T\}]$. Note that $\mathcal{N}$ cannot have one irreflexive point as a maximum. This would imply $Ch(N, S) = (i_1, r_1, 1, 0)$, see Theorem 8. Then the irreflexive point does not satisfy $\neg\Box\bot$, hence $\bot \in T$, which is a contradiction as far a T is the theory of $\mathcal{M}$.

Let us denote the floors of $\mathcal{N}$ by N_1 and N_2 respectively. In case $\mathcal{N}$ is a one-floor frame, $N_2 = \varnothing$. Since $\mathcal{K}$ is weak-cluster closed, there is $\mathcal{N}^* \in \mathcal{K}$ which is either the concatenation of $\mathcal{N}$ and $\mathcal{M}$ or is a cluster substitution of $\mathcal{M}$ in $\mathcal{N}$. We prove by

induction on the complexity of a formula that for every point $w \in N_1$, we have $\mathcal{N}^*, w \Vdash \phi$ iff $\mathcal{N}, w \Vdash \phi$. The only non-trivial case is for formulas of the form $\Box\phi$. Assume $\mathcal{N}, w \Vdash \Box\phi$, then $\phi \in T$. This means that $\mathcal{M} \Vdash \phi$. Now for every point $w' \in N_1$ such that wSw' we have $\mathcal{N}, w' \Vdash \phi$ and hence by the inductive assumption we get that $\mathcal{N}^*, w' \Vdash \phi$. So $\mathcal{N}^*, w \Vdash \Box\phi$.

Conversely assume for some point $w \in N_1$ we have $\mathcal{N}^*, w \Vdash \Box\phi$. By the same argument as in the previous case, for every point $v \in N_1$ such that $wSv, \mathcal{N}, v \Vdash \phi$. Now if $\mathcal{N}^*$ is a concatenation of $\mathcal{N}$ and $\mathcal{M}$ then $N = N_1$, and hence we have $\mathcal{N}, w \Vdash \Box\phi$. In case $\mathcal{N}^*$ is cluster substitution we additionally need to show that for every point $v \in N_2$, $\mathcal{N}, v \Vdash \phi$. From $\mathcal{N}^*, w \Vdash \Box\phi$ we have that $\mathcal{M} \Vdash \phi$ and hence $\mathcal{M} \Vdash \Box\phi$. This implies that $\neg\Box\phi \notin T$ and hence $\mathcal{N} \Vdash \Diamond\Box\phi$. It is not hard to check that this implies that for every point $v \in N_2$ we have $\mathcal{N}, v \Vdash \phi$. The main point here is that $\mathcal{N}$ cannot have one-irreflexive point as a maximum. Now as $\mathcal{N} \Vdash I$, we have that $\mathcal{N}^* \Vdash I$, hence $\mathcal{N}^*$ is not preferred over $\mathcal{M}$ which implies that $\mathcal{N}^* \Vdash T$. Hence $\mathcal{N} \Vdash T$.

This yields the promised link between $wK4f$ and minimal models in the style of [8,9]; so non-monotonic $wK4f$ may be a promising first step in the search for logics of minimal belief.

6 Conclusions

Following Tarski and McKinsey there are three natural paths from topology to algebraic semantics for logics. The third path involves derivative algebras and has been explored in particular by Leo Esakia [1,21,2] and the Tbilisi group in logic. The first two paths give rise to logics extending intuitionistic logic and to modal $S4$, respectively. In these cases the logics obtained from minimal topological spaces have proved to be highly relevant in AI, for non-monotonic reasoning, logic programming and epistemic logic based on the idea of minimal knowledge. From this point of view, the third path to logics from minimal topological spaces has not previously been investigated. It gives rise to the logic $wK4f$ introduced and characterised in this paper that seems a good starting point for studying the idea of minimal belief, analogous to the minimal knowledge approach of [8,9] based on $S4F$.

We conclude by mentioning briefly how these two logics, hence their corresponding epistemic concepts, can be formally related. There are well-known embedding relations holding between intuitionistic logic H and $S4$ and between $S4$ and $wK4$. It can be shown that these relations extend to those logics based on minimal topological spaces. In fact we get the following picture

$$\begin{array}{ccccc} HT & \xrightarrow{G} & S4F & \xrightarrow{Sp} & wK4f \\ \uparrow & & \uparrow & & \uparrow \\ H & \xrightarrow{G} & S4 & \xrightarrow{Sp} & wK4 \end{array}$$

Here HT is again the logic of here-and-there, G is the well-known Gödel translation and Sp is known as the splitting translation from modal formulas to modal formulas such that in particular for an atom p, $Sp(\Diamond p) = p \vee \Diamond p$, $Sp(\Box p) = p \wedge \Box p$ (see eg [2]). Then in particular $S4F \vdash \varphi$ iff $wK4f \vdash Sp(\varphi)$. In other words, we obtain the natural

interpretation of knowledge as truth together with belief. We plan to elaborate on this in future work and to study how to extend this picture to include the non-monotonic versions of each of the logics at the top of the diagram. Another future topic is to study of the exact relations between non-monotonic $wK4f$ and autoepistemic logic as well as non-monotonic $S4F$.

References

1. Esakia, L.: The modal logic of topological spaces. Georgian Academy of Sciences, 22 p. (1976) (Preprint)
2. Esakia, L.: Intuitionistic logic and modality via topology. Ann. Pure & App. Logic 127, 155–170 (2004)
3. Halpern, J.Y., Moses, Y.: Towards a theory of knowledge and ignorance: preliminary report. In: Apt, K. (ed.) Logics and Models of Concurrent Systems, pp. 459–476. Springer, Heidelberg (1985)
4. Kuratowski, R.: Topology, 2nd edn., vol. 1. Academic Press, London (1976)
5. Segerberg, K.: An Essay in Classical Modal Logic. Filosofiska Studier. Uppsala: Filosofiska Foreningen och Filosofiska Institutionen vid Uppsala Universitet, vol. 13
6. Tarski, A.: Der Aussagenkalkul und die Topologie. Fund. Math. 31, 103–134 (1939)
7. Truszczynski, M.: Embedding Default Logic into Modal Nonmonotonic Logics. In: LPNMR 1991, pp. 151–165 (1991)
8. Schwarz, G., Truszczynski, M.: Modal Logic S4F and the minimal knowledge paradigm. In: Proc. TARK-IV, pp. 184–198. Morgan Kaufmann, Monterey (1992)
9. Truszczynski, M., Schwarz, G.: Minimal Knowledge Problem: A New Approach. Artif. Intell. 67(1), 113–141 (1994)
10. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
11. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
12. Gelfond, M., Lifschitz, V., Przy-musinska, H., Truszcynski, M.: Disjunctive defaults. In: Second International Conference on Principles of Knowledge Representation and Reasoning, KR 1991, Cambridge, MA (1991)
13. Lin, F., Shoham, Y.: Epistemic semantics for fixed-points nonmonotonic logics. In: Parikh, R. (ed.) Proc. of the Third Conf. on Theoretical Aspects of Reasoning about Knowledge, pp. 111–120 (1990)
14. Lifschitz, V.: Minimal Belief and Negation as Failure. Art. Intell. 70, 53–72 (1994)
15. Shoham, Y.: Nonmonotonic logics: meaning and utility. In: Proc. IJCAI 1987. Morgan Kaufmann, San Mateo (1987)
16. Truszczynski, M.: The Modal Logic $S4F$, the Default Logic, and the Logic Here-and-There. In: Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007). AAAI Press, Menlo Park (2007)
17. Cabalar, P., Lorenzo, D.: New Insights on the Intuitionistic Interpretation of Default Logic. In: López de Mántaras, R., Saitta, L. (eds.) ECAI 2004, pp. 798–802. IOS Press, Amsterdam (2004)
18. Pearce, D.: Equilibrium logic. AMAI 47(1-2), 3–41 (2006)
19. McKinsey, J., Tarski, A.: The algebra of topology. Annals of Mathematics 45, 141–191 (1944)
20. McKinsey, J., Tarski, A.: On Closed Elements in Closure Algebras. Annals of Mathematics 47, 122–162 (1946)
21. Esakia, L.: Weak transitivity - restitution. In: Logical Studies 2001, vol. 8, pp. 244–255 (2001)
22. Schwarz, G.F.: Minimal model semantics for nonmonotonic modal logics. In: Proceedings of LICS 1992. IEEE Computer Society Press, Washington (1992)

A Logical Account of Lying

Chiaki Sakama[1], Martin Caminada[2], and Andreas Herzig[3]

[1] Wakayama University, Japan
sakama@sys.wakayama-u.ac.jp
[2] University of Luxembourg, Luxembourg
martin.caminada@uni.lu
[3] Université Paul Sabatier, France
herzig@irit.fr

Abstract. This paper aims at providing a formal account of *lying* – a dishonest attitude of human beings. We first formulate lying under propositional modal logic and present basic properties for it. We then investigate why one engages in lying and how one reasons about lying. We distinguish between offensive and defensive lies, or deductive and abductive lies, based on intention behind the act. We also study two weak forms of dishonesty, *bullshit* and *deception*, and provide their logical features in contrast to lying. We finally argue dishonesty postulates that agents should try to satisfy for both moral and self-interested reasons.

1 Introduction

Lying can be considered to be one of the basic behaviors of human beings. In spite of its familiarity to most of us, the question of "What is lying?" has been studied by a number of philosophers (for instance, [5,9,14] and references therein). Surprisingly, however, the topic has been almost completely ignored in artificial intelligence. There are several reasons why the study of lying is important in AI. First, lying is a linguistic behavior inherent to human beings, that requires intelligence and thinking. Studies on lying can thereby contribute to better understand human intelligence. Second, elucidating the mechanism of lying opens possibilities to develop computers that lie [16]. For instance, we can imagine a nurse robot who knows that a patient has a serious cancer but informs the patient that he/she is not in a serious state. Some potential applications of lying in AI and knowledge engineering are also addressed in [3,20]. Third, lying is an act of social interaction. Hence, studying the act in the context of multiagent systems is necessary for designing intelligent agents. A recent study reports that an intelligent agent could behave dishonestly to win a debate in formal argumentation systems [4]. Lying has a distinctive feature as a speech act. According to Searle [19], a speech act is *sincere* if a speaker utters a believed-true sentence. This basic attitude is not applied to lying, that is, *a liar utters a believed-false sentence*. Saint Augustine, who was a Berber philosopher and theologian, says that "the heart of a liar is said to be double, that is, twofold in its thinking: one part consisting of that knowledge which he knows or thinks to be true, yet does not so express it; the other part consisting of that knowledge which he knows or thinks to be false, yet expresses as true" [2, p.55].

Providing a formal account of lying requires one to overcome various difficulties. First of all, there is no universally accepted definition of lying and even the definition

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 286–299, 2010.
© Springer-Verlag Berlin Heidelberg 2010

in the Oxford English Dictionary is problematic[1] [14]. Furthermore, formal logics are usually employed for formulating the truth of sentences and the correctness of inferences, whereas lies contradict the truth [22]. Thus, a formal account of lying is still an open and challenging topic in AI.

The purpose of this paper is to provide a logical account of lying. We formulate various forms of lies using propositional modal logic and investigate formal properties. We also characterize other types of dishonesty and compare them with lying. We propose basic postulates for dishonesty that agents should try to satisfy. The rest of this paper is organized as follows. Section 2 introduces a modal language for belief and intention and provides a logical framework of lying. Section 3 investigates different types of lying and argues their properties. Section 4 formulates *bullshit* and *deception* as weaker forms of dishonesty. Section 5 discusses related issues and Section 6 concludes the paper.

2 Liars' Logic

2.1 A Simple Logic for Belief and Intention

In this paper, we consider a propositional modal logic of intentional communication [7]. A propositional modal language L_0 is built from a finite set of propositional constants $\{p, q, r, \ldots\}$ on the logical connectives $\neg, \vee, \wedge, \supset, \equiv$, and on two families of modal operators, $(B_a)_{a\in A}$ and $(I_a)_{a\in A}$, where A is a finite set of agents. Well-formed formulas (or *sentences*) in L_0 are defined as usual as those belonging to a multi-modal propositional logic. Sentences in L_0 will be denoted by the small Greek letters, and parentheses are employed as usual to clarify the structure of sentences. $\top$ and $\bot$ represent valid and contradictory sentences, respectively. The set of all sentences in L_0 is denoted by Φ and $\Phi^\star = \Phi \setminus \{\top, \bot\}$. A finite set of sentences is identified with the conjunction of all sentences included in the set. The intuitive reading of $B_a\phi$ and $I_a\phi$ are that an agent a believes that ϕ and intends that ϕ, respectively. A Kripkean semantics is defined for L_0, although we omit the details here.[2] A logic BI_0 is defined over L_0, that is an extension of KD45$_n$ [11] and has the following axioms and inference rules:

$(\mathbf{P})$ All propositional tautologies.

$(\mathbf{K_B})$ $B_a\phi \wedge B_a(\phi \supset \psi) \supset B_a\psi$ and $(\mathbf{K_I})$ $I_a\phi \wedge I_a(\phi \supset \psi) \supset I_a\psi$.

$(\mathbf{D_B})$ $B_a\phi \supset \neg B_a\neg\phi$ and $(\mathbf{D_I})$ $I_a\phi \supset \neg I_a\neg\phi$.

$(\mathbf{4_B})$ $B_a\phi \supset B_aB_a\phi$ and $(\mathbf{4_{IB}})$ $I_a\phi \supset B_aI_a\phi$.

$(\mathbf{5_B})$ $\neg B_a\phi \supset B_a\neg B_a\phi$ and $(\mathbf{5_{IB}})$ $\neg I_a\phi \supset B_a\neg I_a\phi$.

$$(\mathbf{MP})\ \frac{\phi \quad \phi \supset \psi}{\psi}, \qquad (\mathbf{N_B})\ \frac{\phi}{B_a\phi}, \qquad (\mathbf{N_I})\ \frac{\phi}{I_a\phi}.$$

To represent a speech act of an agent, we introduce the unary predicate $utter_{xy}$ defined over sentences in L_0 with $x, y \in A$. An expression $utter_{ab}(\sigma)$ means that an agent a expresses a sentence σ to another agent b. A language L_0^U is defined as L_0

[1] The OED definition of lying is: to make a false statement with the intention to deceive.

[2] Informally speaking, $B_a\phi$ (resp. $I_a\phi$) holds iff ϕ is true in all states of affairs compatible with a's current beliefs (resp. intentions).

together with the predicate $utter_{xy}$. If an agent utters something, he/she intends the speech act and is aware of his/her utterance. This is expressed by the next axiom:

$$(\mathbf{U_{IB}}) \quad utter_{ab}(\sigma) \supset I_a(utter_{ab}(\sigma)) \wedge B_a(utter_{ab}(\sigma)).$$

The system BI_0^U, defined over L_0^U, is the weakest extension of BI_0 containing the axiom $(\mathbf{U_{IB}})$. If a sentence ϕ is a theorem of BI_0^U, we write $\vdash \phi$. An agent a has a *knowledge base* K_a as a finite set of believed-true sentences from L_0^U. Each agent believes that other agents follow the same logic BI_0^U in their beliefs and intentions. Thus, $B_aB_b\phi \supset B_a\neg B_b\neg\phi$ and $B_a(I_b\phi \wedge I_b(\phi \supset \psi)) \supset B_aI_b\psi$, for instance. Given two sentences σ and λ in Φ, we write $\sigma \succeq \lambda$ if $\vdash \sigma \supset \lambda$. In this case, we say that σ is *stronger than or equal to* λ (or λ is *weaker than or equal to* σ). We write $\sigma \succ \lambda$ if $\sigma \succeq \lambda$ and $\lambda \not\succeq \sigma$, and say that σ is *stronger than* λ (or λ is *weaker than* σ).

2.2 Lying

Lying can be seen as a speech act of an agent (a speaker) towards another agent (a hearer). For our purpose, we will use a relatively simple definition of lying which seems to be well-accepted in the literature.

> *To lie (to another person) is: to make a believed-false statement (to another person) with the intention that that statement be believed to be true (by the other person).* – [12] and (L6) of [14]

We can then provide a formal definition of lying in L_0^U as follows.

Definition 2.1. (lie) Let a and b be two agents and $\sigma \in \Phi$. Then, define

$$LIE_{ab}(\sigma) \stackrel{def}{=} utter_{ab}(\sigma) \wedge B_a\neg\sigma \wedge I_aB_b\sigma. \tag{1}$$

In this case, we say that a *lies* to b on the sentence σ. σ is also called a *lie*.

By the definition, a lies to b if a utters a believed-false sentence σ to b with the intention that σ is believed by b. Some researchers argue that lying does not necessarily require the use of words [14], but here we consider lying as a statement of a sentence. Note also that the speaker believes $\neg\sigma$, but that the truth of $\neg\sigma$ is not actually required. That is,"a person is to be judged as lying or not lying according to the intention of his own mind, not according to the truth or falsity of the matter itself" [2, p.55]. Lying is not simply saying something that one believes to be false, but involves an intention to deceive. Thus, if one says something manifestly false as a joke or a metaphor, it is not a lie.[3] Lying on valid or contradictory sentences is meaningless.

Proposition 2.1. $\vdash LIE_{ab}(\top) \supset \bot$ *and* $\vdash LIE_{ab}(\bot) \supset \bot$.

Proof. $LIE_{ab}(\top)$ implies $B_a\bot$ that implies $\neg B_a\top$ $(\mathbf{D_B})$, while $\top$ implies $B_a\top$ $(\mathbf{N_B})$. Contradiction. Next, $LIE_{ab}(\bot)$ implies $I_aB_b\bot$, while $B_b\top$ implies $\neg B_b\bot$ $(\mathbf{D_B})$ that implies $I_a\neg B_b\bot$ $(\mathbf{N_I})$ then $\neg I_aB_b\bot$ $(\mathbf{D_I})$. Contradiction. □

If an agent lies, he/she is aware of his/her dishonest act.

[3] Some philosophers argue that an intention to deceive is not a necessary condition of lying, however [5].

Proposition 2.2. $\vdash LIE_{ab}(\sigma) \supset B_a(LIE_{ab}(\sigma))$ *for any* $\sigma \in \Phi$.

Proof. The result holds by Def. 2.1(1) and the axioms ($\mathbf{U_{IB}}$), ($\mathbf{4_B}$), and ($\mathbf{4_{IB}}$). □

Lying to oneself leads to contradiction.[4]

Proposition 2.3. $\vdash LIE_{aa}(\sigma) \supset \bot$ *for any* $\sigma \in \Phi$.

Proof. $LIE_{aa}(\sigma)$ implies $B_a\neg\sigma \wedge I_aB_a\sigma$. $B_a\neg\sigma$ implies $\neg B_a\sigma$ ($\mathbf{D_B}$), which implies $I_a\neg B_a\sigma$ ($\mathbf{N_I}$). On the other hand, $I_aB_a\sigma$ implies $\neg I_a\neg B_a\sigma$ ($\mathbf{D_I}$). Contradiction. □

Note that when lying, a speaker does in general not care about the belief state of the hearer. If a speaker a believes that a hearer b believes σ, $LIE_{ab}(\sigma)$ would have the effect of strengthening the incorrect belief of the hearer. On the other hand, if a believes that b disbelieves σ, $LIE_{ab}(\sigma)$ might cause belief revision of the hearer.

3 Various Forms of Lying

3.1 Offensive Lie vs. Defensive Lie

One has motives for lying and several reasons are considered behind the act. Here we consider two typical cases. First, one lies to have a positive (or wanted) outcome that would not be gained by telling the truth. Second, one lies to avoid a negative (or unwanted) outcome that would happen when telling the truth. An example of the first case is that a salesperson lies about the quality of a product, which leads a customer to make a (wrong) decision of buying the product. An example of the second case is that a child lies about his/her good performance in the exam to avoid punishment by his/her parents. We say that the first case of lying is *offensive*, while the second case is *defensive*. A *positive outcome* or a *negative outcome* is an effect expected by a speaker with respect to the result of reasoning by a hearer. Thus, in offensive/defensive lying a speaker reasons about what a hearer believes in the context of discourse.

Definition 3.1. (offensive/defensive lie) Let a and b be two agents and $\sigma, \phi, \psi \in \Phi$. Then, define

$$O\text{-}LIE_{ab}(\sigma,\phi) \stackrel{def}{=} I_aB_b\,\phi \wedge \neg B_aB_b(\neg\sigma \supset \phi) \wedge B_aB_b(\sigma \supset \phi) \wedge LIE_{ab}(\sigma). \quad (2)$$

In this case, a *offensively lies* to b on σ to have the positive outcome ϕ. σ is also called an *offensive lie* for ϕ. Next, define

$$D\text{-}LIE_{ab}(\sigma,\psi) \stackrel{def}{=} I_a\neg B_b\,\psi \wedge \neg B_a\neg B_b(\neg\sigma \wedge \psi) \wedge B_a\neg B_b(\sigma \wedge \psi) \wedge LIE_{ab}(\sigma). \quad (3)$$

In this case, a *defensively lies* to b on σ to avoid the negative outcome ψ. σ is also called a *defensive lie* for ψ.

Intuitive meanings of the definition are as follows. In (2), a offensively lies on σ if a has an intension to make b believe ϕ. And a disbelieves that the believed-true sentence $\neg\sigma$ leads b to believe a positive outcome ϕ, while a believes that the believed-false sentence

[4] "In short, self-deception involves an inner conflict, perhaps the existence of contradiction" [8].

σ does. With these conditions, a lies to b on σ. In (3) a defensively lies on σ if a has an intension to make b disbelieve ψ. And a considers it possible that b believes that the believed-true sentence $\neg\sigma$ and a negative outcome ψ hold at the same time, while a does not consider it possible that b believes that the believed-false sentence σ and ψ hold simultaneously. With these conditions, a lies to b on σ. As a special case, a may lie to b on the sentence ϕ (resp. $\neg\psi$) to make b believe ϕ (resp. disbelieve ψ).

Proposition 3.1. *Let a and b be two agents and $\phi, \psi \in \Phi$.*

(i) $O\text{-}LIE_{ab}(\phi, \phi) \equiv \neg B_a B_b\, \phi \wedge LIE_{ab}(\phi)$.
(ii) $D\text{-}LIE_{ab}(\neg\psi, \psi) \equiv \neg B_a \neg B_b\, \psi \wedge LIE_{ab}(\neg\psi)$.

Proof. The result (i) directly follows by the definition. (ii) also follows by the fact that $I_a \neg B_a \psi$ is implied by $I_a B_a \neg\psi$ of $LIE_{ab}(\neg\psi)$ by $(\mathbf{D_B})$, $(\mathbf{N_I})$ and $(\mathbf{K_I})$. □

In $O\text{-}LIE_{ab}(\phi, \phi)$, the condition $\neg B_a B_b\, \phi$ means that a has motives for offensive lying when a disbelieves that b believes the positive outcome ϕ. In $D\text{-}LIE_{ab}(\neg\psi, \psi)$, the condition $\neg B_a \neg B_b\, \psi$ means that a has motives for defensive lying when a considers it possible that b believes the negative outcome ψ. Thus, the definitions of offensive and defensive lies are stronger than the definition of lies of Definition 2.1. This is due to the fact that offensive (resp. defensive) lying has additional objectives to have positive outcomes (resp. avoid negative outcomes), while lying in general is only aimed at making the hearer believe the uttered statement itself.

Example 3.1. Suppose that a salesperson a is dealing with a customer b, and that a is requested to provide b with information about the quality of the product. The salesperson believes $\neg\, high\_quality$ and also believes that the customer has the knowledge base $K_b = \{\, high\_quality \supset buy \,\}$. When the salesperson has the positive outcome $\phi = buy$, telling the true belief does not lead b to buy the product. In this case, a offensively lies to b on $\sigma = high\_quality$. Next, suppose that a child a and his/her mother b talk about examination, and a is requested to provide b with information about the score and the rank. The child believes $\neg(high\_score \wedge high\_rank)$, and also believes that mother has the knowledge base $K_b = \{\, \neg(\, high\_score \wedge high\_rank) \supset punish \,\}$. When the child has the negative outcome $\psi = punish$, telling the true belief leads b to punish a. In this case, a defensively lies to b on $\sigma = high\_score \wedge high\_rank$.

When an agent lies to another agent, the success of the act depends on the belief state or knowledgeability of the hearer. For instance, it is easier to mislead children than adults, and it is more difficult to mislead experts than novices. We next consider how different degrees of lies are used depending on a hearer's belief state that is believed by a speaker. An agent b is *knowledgeable not less than* another agent c if $B_c\, \phi \supset B_b\, \phi$ holds for any formula $\phi \in \Phi$. Suppose that there are three agents a, b and c, and a believes that b is knowledgeable not less than c. Then, in offensive lying Def.3.1(2), (i) $\neg B_a B_b(\neg\sigma \supset \phi)$ implies $\neg B_a B_c(\neg\sigma \supset \phi)$, and (ii) $B_a B_b(\sigma \supset \phi)$ does not imply $B_a B_c(\sigma \supset \phi)$. By (i) if a disbelieves that a positive outcome ϕ is not gained by telling the believed-true sentence $\neg\sigma$ to b, then a also has the same disbelief for c. This means that a's motive of offensively lying to c is not less than the motive of offensively lying to b. By (ii) even if a believes that a lie σ leads b to a positive outcome ϕ, a does not believe that

the same lie leads c to ϕ. This means that, to have a positive outcome ϕ from c, a has to craft a lie that is not weaker than σ in general. In case of defensive lying Def.3.1(3), (iii) $\neg B_a \neg B_b(\neg\sigma \wedge \psi)$ does not imply $\neg B_a \neg B_c(\neg\sigma \wedge \psi)$, and (iv) $B_a \neg B_b(\sigma \wedge \psi)$ implies $B_a \neg B_c(\sigma \wedge \psi)$. By (iii) even if a considers it possible that b believes that the believed-true sentence $\neg\sigma$ and a negative outcome ψ hold simultaneously, a does not have the same belief for c. This means that a's motive of defensively lying to c is not more than the motive of defensively lying to b. By (iv) if a does not consider it possible that b believes the believed-false sentence σ and ψ hold simultaneously, then a also has the same belief for c. This means that, to avoid a negative outcome ψ from c, a can craft a lie that is not stronger than σ in general. We next formulate the situation.

Let σ be an offensive lie for a positive outcome ϕ. If $\sigma' \succeq \sigma$ implies $\sigma \succeq \sigma'$ for any offensive lie σ' for ϕ, then σ is called a *strongest* offensive lie (denoted by σ_s). By contrast, if $\sigma \succeq \sigma'$ implies $\sigma' \succeq \sigma$ for any offensive lie σ' for ϕ, then σ is called a *weakest* offensive lie (denoted by σ_w). The notion of the strongest/weakest defensive lies is similarly defined.

Proposition 3.2. *Suppose that there are three agents a, b and c, and a believes that b is knowledgeable not less than c. Let ϕ and ψ be sentences in Φ. Then,*

(i) $\vdash (O\text{-}LIE_{ab}(\sigma_w, \phi) \wedge O\text{-}LIE_{ac}(\lambda, \phi)) \supset \bot$ *for any $\lambda \in \Phi$ such that $\sigma_w \succ \lambda$.*
(ii) $\vdash (D\text{-}LIE_{ab}(\sigma_s, \psi) \wedge D\text{-}LIE_{ac}(\lambda, \psi)) \supset \bot$ *for any $\lambda \in \Phi$ such that $\lambda \succ \sigma_s$.*

Proof. (i) Suppose that $O\text{-}LIE_{ab}(\sigma_w, \phi) \wedge O\text{-}LIE_{ac}(\lambda, \phi)$ and $\sigma_w \succ \lambda$ hold. As a believes that b is knowledgeable not less than c, $B_a B_c(\lambda \supset \phi)$ implies $B_a B_b(\lambda \supset \phi)$ $(*)$. Next, assume that $B_a B_b(\neg\lambda \supset \phi)$ $(**)$. By $(*)$ and $(**)$, it holds that $B_a B_b\, \phi$, which implies $B_a B_b(\neg\sigma_w \supset \phi)$ $(\dagger)$. As $O\text{-}LIE_{ab}(\sigma_w, \phi)$, it holds that $\neg B_a B_b(\neg\sigma_w \supset \phi)$ which contradicts $(\dagger)$. So $\neg B_a B_b(\neg\lambda \supset \phi)$ $(\ddagger)$. The facts $(*)$ and $(\ddagger)$ imply that a can offensively lie to b on the sentence λ for the outcome ϕ. As σ_w is the weakest lie, $\sigma_w \not\succ \lambda$. Contradiction. (ii) is proved in a similar way. □

Example 3.2. (cont. Example 3.1) Suppose that the salesperson a deals with another customer c. a notices that c is more cautious than b in making decisions, and believes that c will buy the product if it is valuable as well as good in quality. But a believes that the product is neither of these $\neg high\_quality \wedge \neg valuable$, and also believes that $K_c = \{\, (high\_quality \wedge valuable) \supset buy \,\}$ where $B_c(K_c) \supset B_b(K_c)$ holds. To have the positive outcome $\phi = buy$, a has to lie offensively on the sentence $\lambda = high\_quality \wedge valuable$, which is stronger than σ, to convince c to buy the product. Next, suppose that a child a has a dialogue with his/her father c. a knows that father is more generous than mother, and believes that he is only concerned about the score. But a believes $\neg\, high\_score$, and also believes that $K_c = \{\, \neg high\_score \supset punish \,\}$ where $B_c(K_c) \supset B_b(K_c)$ holds. To avoid the negative outcome $\psi = punish$, a lies defensively on the sentence $\lambda = high\_score$, which is weaker than σ, to persuade father not to punish him/her.

3.2 Deductive Lie vs. Abductive Lie

By an offensive lie (resp. a defensive lie), a speaker intends to mislead a hearer to deduce a wrong conclusion (resp. not to deduce a right conclusion). We call these types

of lies *deductive lies*. By contrast, a person often lies in order to block another person for generating assumptions. For instance, suppose a man, say, Sam, who is coming home late because he is cheating on his wife. Based on the observation "Sam arrives late", his wife could perform *abduction* and one of the possible explanations would be "Sam cheats on his wife". Sam, of course, does not want this abduction to take place, so he lies about a possible other reason, "I had to do overtime at work". Sam's hope is that once his wife has this incorrect information, her abductive reasoning process will stop. She will no longer continue possible abduction, and will never even be aware of the possibility of Sam's cheating on her (if she trusts her husband).

Abduction is the process of forming an explanatory hypothesis from an observation [18]. Formally, let o be a sentence representing an *observation* and H a set of sentences representing a *hypothesis*. Given a knowledge base K and an observation o, a hypothesis H *explains* o in K if $K \wedge H \vdash o$ where $K \wedge H$ is consistent. An agent lies to interrupt abduction (by another agent) that produces an unwanted explanation for him/her. Let $\Sigma_a\ (\subseteq K_a)$ be a set of sentences (called a *secret set*) which an agent a wants to conceal from another agent b. Abductive lie is then defined as follows.

Definition 3.2. (abductive lie) Let a and b be two agents and $o \in \Phi \setminus \Sigma_a$ a sentence observed by them. Also, let $\sigma \in \Phi \setminus \Sigma_a$ such that $\sigma \not\equiv o$. Then, define

$$A\text{-}LIE_{ab}(\sigma, o) \stackrel{def}{=} B_a\, o \wedge B_a \neg B_b(\Delta \supset o) \wedge B_a(B_b(\Gamma \supset o) \wedge \neg B_b \neg \Gamma) \\ \wedge\ B_a(B_b(\sigma \supset o) \wedge \neg B_b \neg \sigma) \wedge \bigwedge_{\gamma \in \Sigma_a} I_a \neg B_b\, \gamma \wedge LIE_{ab}(\sigma) \quad (4)$$

where Δ is any subset of $K_a \setminus \Sigma_a$ and $\Gamma\ (\subseteq \Phi)$ is a set of sentences such that $\Gamma \cap \Sigma_a \neq \emptyset$. In this case, we say that the agent a *abductively lies* to another agent b on the sentence σ. σ is also said an *abductive lie* for the observation o.

In (4), $B_a\, o$ represents that a believes o. $B_a \neg B_b(\Delta \supset o)$ implies $B_a \neg B_b\, o$, so that a believes that b requires some explanation once he/she observes o. However, a believes that b does not explain o by believed-true sentences of a without some secret sentences (i.e., $B_a \neg B_b(\Delta \supset o)$). a believes that b explains o by either using some secret sentences of a (i.e., $B_a(B_b(\Gamma \supset o) \wedge \neg B_b \neg \Gamma)$ or some believed-false sentence σ of a (i.e., $B_a(B_b(\sigma \supset o) \wedge \neg B_b \neg \sigma)$), but a does not want b's believing any sentence γ in Σ_a (i.e., $\bigwedge_{\gamma \in \Sigma_a} I_a \neg B_b\, \gamma$). In this case, a abductively lies to b on σ for explaining o. Note that $\vdash A\text{-}LIE_{ab}(\sigma, \top) \supset \bot$ and $\vdash A\text{-}LIE_{ab}(\sigma, \bot) \supset \bot$ for any σ.

Example 3.3. Suppose that Sam has the knowledge base $K_a = \{\, cheat,\ \neg overtime,\ cheat \supset late,\ \ overtime \supset late \,\}$, and believes that his wife has the knowledge base $K_b = \{\, cheat \supset late,\ \ overtime \supset late \,\}$. Let $\Sigma_a = \{cheat\}$, that is, Sam wants to keep his cheating behavior secret. Given the observation $o = late$, Sam believes that his wife can abduce $\Gamma = \{cheat\}$ as a possible explanation for o. Then, Sam abductively lies on $\sigma = overtime$ which explains his late arrival and would stop her abducing the explanation $cheat$.

The effect of an abductive lie also depends on the belief state of a hearer. If a believes that an agent b is knowledgeable not less than another agent c, then the condition $B_a \neg B_c(\Delta \supset o)$ in $A\text{-}LIE_{ac}(\sigma, o)$ holds, while $B_a(B_c(\Gamma \supset o) \wedge \neg B_c \neg \Gamma)$ and

$B_a(B_c(\sigma \supset o) \wedge \neg B_c \neg \sigma)$ do not necessarily hold. This means that a's motive of abductively lying to c is not more than the motive of abductively lying to b. For instance, if Sam believes that his daughter has the knowledge base $K_c = \{\, overtime \supset late \,\}$, then $B_a \neg B_c(cheat \supset late)$ and Sam does not need to lie her. When a abductively lies to c, however, a has to craft a lie that is not weaker than σ in general. If $K'_c = \{\, cheat \supset late \,\}$, then Sam has to make the stronger lie $\lambda = overtime \wedge (overtime \supset late)$, for instance. Given an observation o, the notion of a weakest abductive lie σ_w is defined in a way similar to a weakest offensive lie. Then we have the next result.

Proposition 3.3. *Suppose that there are three agents a, b and c, and a believes that b is knowledgeable not less than c. Let o be a sentence in $\Phi \setminus \Sigma_a$. Then,*
$\vdash (A\text{-}LIE_{ab}(\sigma_w, o) \wedge A\text{-}LIE_{ac}(\lambda, o)) \supset \bot$ *for any $\lambda \in \Phi$ such that $\sigma_w \succ \lambda$.*

Proof. Similar to the proof of Proposition 3.2(1). □

3.3 What Are the Most Effective Lies?

In deductive lying and abductive lying, a number of candidate lies exist to achieve a speaker's goal. Then a question is how good liars select "best lies". As observed in Propositions 3.2 and 3.3, a speaker can select different degrees of lies according to the knowledgeability of a hearer. A stronger lie would be needed to have a positive outcome from a less knowledgeable hearer, while a weaker lie would be enough to avoid a negative outcome from the same hearer. A liar normally wants to keep his/her lie as small as possible. This is because, "The lie, to his immediate advantage, often results in an overall net loss of freedom in what he can do or say... The need to maintain the deception binds him" [12, p.119]. A stronger lie makes the liar less free, which he wants to avoid anyway. Besides, lies make the belief state of a hearer deviate from the objective reality (or, at least from the reality as believed by a speaker) and a stronger lie would increase such deviation. This is undesirable for a speaker because it increases the chance of the lie being detected. The best lie is a lie that does not have too much "collateral damage" on a hearer. We state a guideline for agents to satisfy in lying as the next postulate. Let λ, σ, o, ϕ, $\psi \in \Phi^\star$ and $\sigma \succeq \lambda$. Then, we have the next postulate.

Postulate I: Never tell an unnecessarily strong lie.
(i) $B_a(O\text{-}LIE_{ab}(\sigma, \phi) \supset B_b\phi) \wedge B_a(O\text{-}LIE_{ab}(\lambda, \phi) \supset B_b\phi) \supset \neg O\text{-}LIE_{ab}(\sigma, \phi)$.
(ii) $B_a(D\text{-}LIE_{ab}(\sigma, \psi) \supset \neg B_b\psi) \wedge B_a(D\text{-}LIE_{ab}(\lambda, \psi) \supset \neg B_b\psi) \supset \neg D\text{-}LIE_{ab}(\sigma, \psi)$.
(iii) $B_a(A\text{-}LIE_{ab}(\sigma, o) \supset B_b\, o) \wedge B_a(A\text{-}LIE_{ab}(\lambda, o) \supset B_b\, o) \supset \neg A\text{-}LIE_{ab}(\sigma, o)$.

4 Weak Form of Dishonesty

4.1 Bullshit

Frankfurt [10] studies a category of dishonesty, called *bullshit*, that is different from lies. Bullshit is a statement that "is grounded neither in a belief that it is true nor, as a lie must be, in a belief that it is not true" (ibid., p.33). As an example, consider a financial consultant paid by the hour to provide advice to his clients. The consultant gives advice to buy stocks, for instance, but he may or may not believe that buying stocks is the best

strategy (due to the lack of expertise). Bullshit is a quite common phenomenon in daily life. Frankfurt states a reason for its occurrence as follows: "Bullshit is unavoidable whenever circumstances require someone to talk without knowing what he is talking about. Thus the production of bullshit is stimulated whenever a person's obligations or opportunities to speak about some topic exceed his knowledge of the facts that are relevant to that topic" (ibid., p.63). Bullshit can formally be defined as follows.

Definition 4.1. (bullshit) Let a and b be two agents and $\sigma \in \Phi$. Then,

$$BS_{ab}(\sigma) \stackrel{def}{=} utter_{ab}(\sigma) \wedge \neg B_a\sigma \wedge \neg B_a\neg\sigma. \tag{5}$$

In this case, we say that an agent a *bullshits* to another agent b on the sentence σ. σ is also called *bullshit* (shortly, BS).

In lying Def.2.1(1), the speaker a disbelieves σ but believes $\neg\sigma$. When bullshitting Def.4.1(5), on the other hand, a disbelieves $\neg\sigma$ either. In other words, a has no belief with respect to the truth value of σ. So one cannot bullshit about one's own beliefs.

Proposition 4.1. $\vdash BS_{ab}(B_a\sigma) \supset \bot$ *and* $\vdash BS_{ab}(\neg B_a\sigma) \supset \bot$ *for any* $\sigma \in \Phi$.

Proof. Both $BS_{ab}(B_a\sigma)$ and $BS_{ab}(\neg B_a\sigma)$ imply $\neg B_aB_a\sigma \wedge \neg B_a\neg B_a\sigma$. Here $\neg B_aB_a\sigma$ implies $\neg B_a\sigma$ ($\mathbf{4_B}$), which implies $B_a\neg B_a\sigma$ ($\mathbf{N_B}$). This contradicts $\neg B_a\neg B_a\sigma$. □

Bullshitting on valid or contradictory sentences is meaningless.

Proposition 4.2. $\vdash BS_{ab}(\top) \supset \bot$ *and* $\vdash BS_{ab}(\bot) \supset \bot$.

Proof. Both $BS_{ab}(\top)$ and $BS_{ab}(\bot)$ imply $\neg B_a\top$, but $\top$ implies $B_a\top$ ($\mathbf{N_B}$). □

Like lying, a bullshitter notices his/her act.

Proposition 4.3. $\vdash BS_{ab}(\sigma) \supset B_a(BS_{ab}(\sigma))$ *for any* $\sigma \in \Phi$.

There are some differences between lies and BS. First, bullshitting to oneself $BS_{aa}(\sigma)$ is possible in general. Second, $BS_{ab}(\sigma)$ does not contradict the belief of the speaker a. These facts imply that one cannot lie and bullshit on the same sentence.

Proposition 4.4. $\vdash LIE_{ab}(\sigma) \wedge BS_{ab}(\sigma) \supset \bot$ *for any* $\sigma \in \Phi$.

Proof. $LIE_{ab}(\sigma)$ implies $B_a\neg\sigma$, while $BS_{ab}(\sigma)$ implies $\neg B_a\neg\sigma$. □

Another important difference is that BS does not require the intention of a speaker a to make a hearer b believe σ. In the above example, the financial consultant has no interest in making the client believe that buying stocks is the best strategy or not. The only concern of the consultant is that the client believes that the statement is based on financial expertise. Since a has no belief with respect to σ, there is a freedom for a speaker to utter σ or $\neg\sigma$. The most effective BS is the one that is coherent with the speaker's belief. The choice whether to utter σ or $\neg\sigma$ is also decided by how likely it will be for a hearer to believe one of them (given some additional explanation). This is in contrast to lying where speakers have no freedom to make this choice because one of these options (either σ or $\neg\sigma$) will have consequences they might want to enjoy (or

which they might want to avoid). A liar usually has an interest in creating a particular belief at a hearer. This is not always the case for BS, however.

On the other hand, there is BS that accompanies some intention. For instance, suppose a salesperson who is paid on commission basis, but does not really know the products that he is selling. The salesperson would make the claim that a product has a high quality, without having any knowledge on this. This is also an example of BS. However, making a client believe that the product has a high quality is preferred to making the client believe that the product has a low quality. The situation here differs from that of the financial consultant mentioned above (who is paid by the hour by the client, and hence has no intrinsic interest to advise to buy stocks or not). Such *intentional bullshit* is defined as

$$I\text{-}BS_{ab}(\sigma) \stackrel{def}{=} BS_{ab}(\sigma) \wedge I_a B_b \sigma. \tag{6}$$

By contrast, $BS_{ab}(\sigma)$ without $I_a B_b \sigma$ is called *unintentional*. In this paper, we will ignore this difference in cases where it is unimportant. Intentional BS (6) is similar to lies, so that offensive/defensive or deductive/abductive intentional BS could be considered. Different from unintentional BS, intentional BS to oneself is inconsistent.

Proposition 4.5. $\vdash I\text{-}BS_{aa}(\sigma) \supset \bot$ *for any* $\sigma \in \Phi$.

Proof. Similar to the proof of Proposition 2.3. □

Next we consider what is best BS. Any BS is dishonest, but the consequences of faking are generally less severe for a weak bullshitter than for a strong bullshitter. Suppose a salesperson who bullshits for selling specified products, say $high\_quality$, that is weaker than the bullshit $high\_quality \wedge valuable$. If a customer decides to buy the product, the strong bullshitter would be responsible for the value as well as the quality. Getting more responsibility is undesirable for a bullshitter anyway. We formulate the situation for *offensive* intentional BS. For $\sigma, \phi \in \Phi$, let us define

$$O\text{-}BS_{ab}(\sigma, \phi) \stackrel{def}{=} I_a B_b \phi \wedge \neg B_a B_b(\neg\sigma \supset \phi) \wedge B_a B_b(\sigma \supset \phi) \wedge I\text{-}BS_{ab}(\sigma).$$

Then we have the next postulate for BS.

Postulate II: Never tell unnecessarily strong BS. Let $\lambda, \sigma, \phi \in \Phi^\star$ and $\sigma \succeq \lambda$. Then, $B_a(O\text{-}BS_{ab}(\sigma, \phi) \supset B_b\phi) \wedge B_a(O\text{-}BS_{ab}(\lambda, \phi) \supset B_b\phi) \supset \neg O\text{-}BS_{ab}(\sigma, \phi)$.

Similar postulates are considered for defensive or abductive intentional BS. Lies and BS are two different forms of dishonesty, but lies are considered more sinful than BS.[5] This is because a liar intentionally implants wrong beliefs at the hearer, while a bullshitter spits out statements, intentionally or not, without knowing if they are true. As a result, "people do tend to be more tolerant of bullshit than of lies, perhaps because we are less inclined to take the former as a personal affront" [10, p.50]. This leads us to the next postulate.

Postulate III: Never lie if you can bullshit your way out of it. Let $\lambda, \sigma, \phi \in \Phi^\star$. Then, $B_a(O\text{-}BS_{ab}(\sigma, \phi) \supset B_b\phi) \wedge B_a(O\text{-}LIE_{ab}(\lambda, \phi) \supset B_b\phi) \supset \neg O\text{-}LIE_{ab}(\lambda, \phi)$.

[5] Some philosophers consider that bullshit is a class of lies [5, cf. L5].

4.2 Deception

Another form of dishonesty which we consider here is *deception*. There is no universally agreed definition of deception [6,13], so we consider the one argued in [1]. Different from lying, there is no untruthfulness condition in deception. That is, a speaker makes a believed-true statement with the intention that a hearer misuses it to reach a wrong conclusion. For instance, John, who wants to marry his girlfriend Mary, tells her that he got a job at a company. Mary then considers that John has a stable income now and would agree to marry him. The company is almost bankrupt, however, and John believes that he would not get a stable income. But John does not tell Mary that his company is going bankrupt. In this speech act, John is telling the truth, while he expects that Mary will reach a conclusion "stable income" which he believes to be false. Thus, different from lies or BS, a deceiver asserts what he/she believes true, while, at the same time, he/she conceals something of the truth hoping that a hearer will make an incorrect inference based on incomplete beliefs.[6] Caminada [4] captures the point as "With deception, one makes use of the *nonmonotonic* inference capabilities of the other person in order to implant wrong beliefs, without having to resort to lying ourselves". In the above example, John believes that Mary has the belief "$B_m((get\_job \wedge \neg B_m \neg stable) \supset stable)$". John then intends to make Mary believe get_job, while withholding $\neg stable$, which would result in Mary's believing $stable$. This is the effect of *default reasoning*. Now deception is formulated as follows.

Definition 4.2. (deception) Let a and b be two agents and $\delta, \sigma \in \Phi$ such that $\delta \not\equiv \sigma$. Then, define

$$DEC_{ab}(\sigma, \delta) \stackrel{def}{=} utter_{ab}(\sigma) \wedge B_a\sigma \wedge I_a B_b\sigma \wedge B_a B_b((\sigma \wedge \neg B_b \neg\delta) \supset \delta) \wedge B_a \neg B_b \neg\delta \wedge B_a \neg\delta \wedge I_a B_b \delta. \quad (7)$$

In this case, we say that an agent a *deceives* another agent b on the sentence σ. σ is also called *deception*.

In (7), the speaker a utters a believed-true sentence σ with the intention of making a hearer b believe it (i.e., $utter_{ab}(\sigma) \wedge B_a\sigma \wedge I_a B_b\sigma$). a believes that b uses σ to reach a default conclusion δ (i.e., $B_a B_b((\sigma \wedge \neg B_b \neg\delta) \supset \delta)$). a also believes that b disbelieves the falsity of δ (i.e., $B_a \neg B_b \neg\delta$), while a believes it (i.e., $B_a \neg\delta$). And believing δ by the hearer b is what the speaker a intends to achieve (i.e., $I_a B_b \delta$). Note that nonmonotonicity arises in $B_b((\sigma \wedge \neg B_b \neg\delta) \supset \delta)$. Compared with definitions of lies and bullshit, one can observe that the act of deception is a bit complicated. In fact, "The deceiver takes a more circuitous route to his success, where lying is an easier and more certain way to mislead" [1, p.440]. A reason for the complication is due to the fact that deception works by nonmonotonic reasoning.

Like lying and $I\text{-}BS$, the following properties hold.

Proposition 4.6. $\vdash DEC_{ab}(\bot, \delta) \supset \bot$ *for any* $\delta \in \Phi$.

Proposition 4.7. $\vdash DEC_{ab}(\sigma, \delta) \supset B_a(DEC_{ab}(\sigma, \delta))$ *for any* $\sigma, \delta \in \Phi$.

[6] Some philosophers call this a "lie of omission" [14].

Proposition 4.8. $\vdash DEC_{aa}(\sigma, \delta) \supset \bot$ *for any* $\sigma, \delta \in \Phi$.

In contrast to lying and BS, $DEC_{ab}(\top, \delta)$ is consistent. In fact, it becomes

$$DEC_{ab}(\top, \delta) = utter_{ab}(\top) \wedge B_a B_b(\neg B_b \neg \delta \supset \delta) \wedge B_a \neg B_b \neg \delta \wedge B_a \neg \delta \wedge I_a B_b \delta.$$

In this case, a deceiver utters no meaningful information and just expects a hearer to reach a default conclusion δ. Different from lying and BS, a deceiver utters believed-true sentences. This implies that one cannot lie and deceive, nor bullshit and deceive, on the same sentence.

Proposition 4.9. $\vdash LIE_{ab}(\sigma) \wedge DEC_{ab}(\sigma, \delta) \supset \bot$ *and* $\vdash BS_{ab}(\sigma) \wedge DEC_{ab}(\sigma, \delta) \supset \bot$ *for any* $\sigma, \delta \in \Phi$.

As deception accompanies intention, offensive/defensive or deductive/abductive deception can also be defined. In lying and bullshitting, it is reasonable (and courteous to a hearer) not to lie and bullshit more than absolutely necessary (Postulates I and II). In case of deception, on the other hand, this is not necessarily the case. If an agent a deceives another agent b on the sentence $\sigma \wedge \lambda$, then the deception $\sigma \wedge \lambda$ is stronger than the deception σ. However, providing more information increases the knowledge of a hearer. For a speaker, providing more information implies concealing less information, which alleviates immoral feeling of the speaker. Thus, there is no reason to prefer the weakest form of deception, so we do not have a postulate mandating it. On the other hand, deception is considered preferable to lies and BS as a speaker utters a believed-true sentence. This leads to the following postulate.

Postulate IV: Never lie nor bullshit if you can deceive your way out of it.
Let $\delta, \lambda, \sigma \in \Phi^\star$. Then,
(i) $B_a(DEC_{ab}(\sigma, \delta) \supset B_b\delta) \wedge B_a(O\text{-}LIE_{ab}(\lambda, \delta) \supset B_b\delta) \supset \neg O\text{-}LIE_{ab}(\lambda, \delta)$.
(ii) $B_a(DEC_{ab}(\sigma, \delta) \supset B_b\delta) \wedge B_a(O\text{-}BS_{ab}(\lambda, \delta) \supset B_b\delta) \supset \neg O\text{-}BS_{ab}(\lambda, \delta)$.

The postulates I–IV are statements that agents should try to satisfy, both for moral reasons and for self-interested reasons (lower punishments if caught). If we assume that agents try to satisfy the dishonesty postulates, and that lying is worse than BS, which is again worse than deception, then one can characterize an agent by the worst level of dishonesty it is willing to commit in order to achieve a goal. For instance, a lawyer agent might be willing to deceive (providing only information favorable to his client) but not to BS nor to lie. So if one detects that an agent is deceiving, one cannot infer that it is also willing to BS or lie. However, the opposite is the case. If an agent is willing to lie, then from the dishonesty postulates, it can also be assumed to be willing to BS or to deceive. So an agent who is caught on deceiving can perhaps still be trusted not to lie (if trust is the default attitude), but an agent that is caught on lying cannot be trusted at all anymore (also regarding BS and deception). In multiagent systems if agents have implemented the dishonesty postulates, then this helps one to reason about the possible dishonesty of other agents, and about the extent to which they can still be trusted.

5 Discussion

Some attempts have been made to formulate lying using modal logic. O'Neill [17] provides logical definitions of lies and deception based on the logic of [7]. In contrast

to our formulation with the logic BI_0, he uses the logic BI_2 which has four different modalities of belief, intention, common belief, and communication. His primary interest is to formulate various types of speech acts in an epistemic logic, and he does not investigate inference mechanisms behind the act of lying and other dishonesty. Different epistemic approaches are also reported in [22], but they just provide definitions of lies or deceptive utterances. Bonatti et al. [3] study databases that could lie to users to preserve security. They introduce a propositional modal logic to reason about databases, secrets, and users' beliefs. Their goal is formulating not lying but query answering in secure databases. Sklar et al. [20] formulate lying with argument-based dialogues. Their goal is capturing lies as contradictory dialogues, and they do not consider various types of lying, BS and deception. Caminada [4] provides a comparative study between lies, BS and deception and shows how these can be formalized using abstract argumentation. The paper provides philosophical arguments, but no logical theory is given.

This paper considered deductive and abductive lies, while lying can be combined with other types of inference. For instance, one may devise *inductive lies* by telling untrue evidences to make a hearer learn wrong inductive hypotheses. This paper focused attention in providing an ontology of dishonesty and explained how various forms of dishonesty are related to each other. It is also important to investigate how one can learn dishonesty attitudes in a multiagent society. Recent studies show that robots which compete for foods learn to conceal food information [15]. Staab and Caminada [21] design and implement an MAS-based software simulator and observe that the incentives for dishonesty emerge for economical agents to have good performance.

6 Conclusion

We have provided a logical analysis of various concepts of dishonesty as they appear in the literature in philosophy and elsewhere. The issue of logical foundation of dishonesty is a topic that has received little attention until now. Our aim is to analyze this issue using a relatively simple logical formalization. Although some formal properties were provided, the strength of the current paper is conceptual rather than purely technical. The postulates can be seen as having a normative value, and should ideally be implemented for individual agents in multiagent systems. In future work, we elaborate the formulation and plan to build a formal system based on it.

References

1. Adler, J.E.: Lying, deceiving, or falsely implicating. J. Philosophy 94(9), 435–452 (1997)
2. Augustine, S.: Lying. In: Treatises on Various Subjects, Fathers of the Church, vol. 56, pp. 45–110 (1952)
3. Bonatti, P.A., Kraus, S., Subrahmanian, V.S.: Foundations of secure deductive databases. IEEE Transactions on Knowledge and Data Engineering 7(3), 406–422 (1995)
4. Caminada, M.: Truth, lies and bullshit, distinguishing classes of dishonesty. In: Proc. IJCAI Workshop on Social Simulation (2009)
5. Carson, T.L.: The definition of lying. Noûs 40(2), 284–306 (2006)
6. Chisholm, R.M., Feehan, T.D.: The intent to deceive. Journal of Philosophy 74(3), 143–159 (1997)

7. Colombetti, M.: A modal logic of intentional communication. Mathematical Social Sciences 38, 171–196 (1999)
8. Demos, R.: Lying to oneself. Journal of Philosophy 57(18), 588–595 (1960)
9. Fallis, D.: What is lying? Journal of Philosophy 106(1), 29–56 (2009)
10. Frankfurt, H.G.: On Bullshit. Princeton Univ. Press, Princeton (2005)
11. Halpern, J., Moses, J.: A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence 54, 349–379 (1992)
12. Kupfer, J.: The moral presumption against lying. Review of Metaphysics 36, 103–126 (1982)
13. Mahon, J.E.: A definition of deceiving. J. Applied Philosophy 21(2), 181–194 (2007)
14. Mahon, J.E.: Two definitions of lying. J. Applied Philosophy 22(2), 211–230 (2008)
15. Mitri, S., Floreano, D., Keller, L.: The evolution of information suppression in communicating robots with conflicting interests. Proc. National Academy of Sciences 106(37), 15786–15790 (2009)
16. Morris, J.: Can computers ever lie? Philosophy Forum 14, 389–401 (1976)
17. O'Neill, B.: A formal system for understanding lies and deceit. In: Jerusalem Conference on Biblical Economics (2003)
18. Peirce, C.S.: Collected Papers of Charles Sanders Peirce. Harvard University Press, Cambridge (1958)
19. Searle, J.R.: Speech Acts. Cambridge University Press, Cambridge (1969)
20. Sklar, E., Parsons, S., Davies, M.: When is it okay to lie? A simple model of contradiction in agent-based dialogues. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS (LNAI), vol. 3366, pp. 251–261. Springer, Heidelberg (2005)
21. Staab, E., Caminada, M.: Assessing the impact of informedness on a consultant's profit. In: Proc. 21st Benelux Conf. on AI (BNAIC 2009), Eindhoven, pp. 397–398 (2009)
22. Urchs, M.: Just lying. Logic and Logical Philosophy 15, 67–89 (2006)

Tabling with Answer Subsumption: Implementation, Applications and Performance

Terrance Swift[1] and David S. Warren[2]

[1] CENTRIA — Universidade Nova de Lisboa
[2] Stony Brook University, Stony Brook, NY

Abstract. Tabled Logic Programming (TLP) is becoming widely available in Prolog systems, but most implementations of TLP implement only *answer variance* in which an answer A is added to the table for a subgoal S only if A is not a variant of any other answer already in the table for S. While TLP with answer variance is powerful enough to implement the well-founded semantics with good termination and complexity properties, TLP becomes much more powerful if a mechanism called *answer subsumption* is used. XSB implements two forms of answer subsumption. The first, partial order answer subsumption, adds A to a table only if A is greater than all other answers already in the table according to a user-defined partial order. The second, lattice answer subsumption, may join A to some other answer in the table according to a user-defined upper semi-lattice. Answer subsumption can be used to implement paraconsistent and quantitative logics, abstract analysis domains, and preference logics. This paper discusses the semantics and implementation of answer subsumption in XSB, and discusses performance and scalability of answer subsumption on a variety of problems.

1 Introduction

Tabled Logic Programming (TLP) currently supports a number of applications in agent frameworks, reasoning over the semantic web, machine learning, and probabilistic logic programming; and TLP is supported by several Prolog systems, including XSB, YAP, B Prolog, Ciao, and ALS. However, an important feature called *answer subsumption* has been little studied in the literature, and is missing from most TLP systems. Most TLP systems add an answer A to a table T only if A is not a variant of some other answer already in T, a technique termed *answer variance*. While answer variance is sufficient to allow tabling to compute the well-founded semantics and to terminate for programs with bounded term-depth, other choices of when and how to add an answer can be made. Using *partial order answer subsumption*, A would be added to T only if A is maximal with respect to other answers in T according to a given partial order $>_O$. Furthermore if A is added, any answers in T that A subsumes (i.e., is greater than in $>_O$) are deleted. When using *lattice answer subsumption*, A itself may not be added to T, rather the join is taken of A and another answer A' in T, with A' being deleted. Despite its conceptual simplicity, answer subsumption can be a powerful tool. Partial order answer subsumption allows a table to retain only answers that are maximal according to a metric or to a preference relation; lattice answer subsumption can form

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 300–312, 2010.
© Springer-Verlag Berlin Heidelberg 2010

the basis of multi-valued logics, quantitative logics, and of abstract interpretations for programs and process logics.

A version of answer subsumption has been available in XSB for over a decade, but its implementation was never described, and only recently was its implementation optimized and declarations provided to make it easy for programmers to use. [10] described how lattice answer subsumption can implement Generalized Annotated Programs [6], but did not provide any details of implementation or benchmarks. Recently, [8] used answer subsumption to implement probabilistic inference, but the benchmark times in that paper were dominated by the cost of maintaining BDDs that represented probabilistic explanations. Beyond related work for XSB, the mode-specific tabling of B Prolog can be seen a restricted form of answer subsumption that allows only min and max over the Prolog term order, and constrains the modes of aggregated tabled subgoals.

This paper makes two main contributions: first, it describes the implementation of partial order and lattice answer subsumption in XSB; and second, it analyzes performance and demonstrates scalability of answer subsumption for applications in social network analysis, abstract interpretation, and query justification through multi-valued logics. The structure of the paper is as follows. Section 2 informally presents the semantics of partial order and lattice answer subsumption. Section 3 then describes the underlying implementation of answer subsumption using the trie-based tabling data structures of XSB. Finally, Section 4 analyzes the performance and scalability of answer subsumption in various applications.

2 An Informal Semantics for Answer Subsumption

Terminology and Conventions. Informally, an answer is simply an atom derived via some fixed-point evaluation of a program P – using tabling or bottom-up evaluation. For simplicity of presentation, we assume that all queries are safe – i.e. that any answer to a query will be ground (the implementation in XSB allows non-ground answers in certain cases). Within this paper, answer subsumption is restricted to occur on a single argument of a predicate; however since answer subsumption is defined for arbitrary terms this restriction does not affect expressibility. For purposes of space, we restrict our description to definite programs. However, the implementation described in Section 3 supports stratified programs, and so can form the basis of formalisms that use negation such as annotated or residuated programs [6, 1]. Finally, all examples use standard Prolog syntax.

Partial Order Answer Subsumption. For simplicity, our first examples make use of a shortest-path predicate (Figure 1) that counts the number of edges between two vertices; more sophisticated uses of answer subsumption are presented in Section 4.

As mentioned above, partial-order answer subsumption retains in a table T only those answers that are maximal according to a given partial order $>_O$. In the case of the shortest-path predicate of Figure 1, $sp(A_1, A_2, A_3) >_O sp(B_1, B_2, B_3)$ if, $A_1 = B_1$,

```
sp(X,Y,1):- edge(X,Y).
sp(X,Z,N):- sp(X,Y,N1),edge(Y,Z),N is N1 + 1.
```

Fig. 1. A Shortest Path Predicate

$A_2 = B_2$, and $A_3 < B_3$. Note that that minimal distances are maximal in $<_O$, and that $<_O$ is undefined if A_3 or B_3 is non-numeric. In XSB, partial order answer subsumption is specified for `sp/3` using the declaration

```
:- table sp(_,_,po(</2)).
```

In a given state of computation, only those answers that are maximal according to $>_O$ are available for resolution. Thus, for a finite graph with cycles, `sp/3` will terminate using answer subsumption, but not with answer variance. Other partial orders beyond distance metrics may be useful. For instance, $>_O$ may specify a preference ordering between derived atoms so that answer subsumption provides an alternative to default-based methods for computing preferences (cf. Section 4.1 for a discussion).

Lattice Answer Subsumption. An upper semi-lattice is a partial order for which any two elements have a unique least upper bound. Because the ordering for the third argument of sp/3 is total, it also forms an upper semi-lattice, and so can be computed using lattice answer subsumption. In XSB lattice answer subsumption for `sp/3` is declared as

```
:- table sp(_,_,lattice(min/3)).
```

with `min/3` defined as `min(X,Y,Z):- Z is min(X,Y).` Operationally, this means that whenever an answer $sp(A_1, A_2, A_3)$ is derived, if there is another answer $sp(B_1, B_2, B_3)$ where $A_1 = B_1$ and $A_2 = B_2$ the join J_3 of A_3 and B_3 is taken, and only $sp(A_1, A_2, J_3)$ is available for resolution. As with a partial order, the join operation ensures termination for shortest path over a finite graph with cycles.

As the following proposition shows, lattice answer subsumption can be modeled either starting with a lattice, or starting with a function with appropriate properties.

Proposition 1. *Let* op *be an associative, commutative, and idempotent binary function. Then there is a partial order P, such that P is an upper semi-lattice with join* op.

Conversely, if a function does not have the above properties, it is not suitable for lattice answer subsumption. Accordingly the aggregate functions count and sum cannot be computed using lattice answer subsumption[1]. Lattice answer subsumption has a variety of applications: Section 4.3 shows how it is used for social-network analysis and for an application of multi-valued logics, [10] describes how a similar formalism can implement a quantitative logic, and [8] describes an implementation of probabilistic logic based on answer subsumption.

Partial Order Answer Subsumption with Abstraction. Computation over an abstract domain may require certain maximal answers to be abstracted. In many cases, abstraction can be modeled by a join operation, but in others the abstraction represents an implicit induction step in the following sense. Given a set $\mathcal{A}$ of answers, it may be detected that the program computed does not have a finite model. An abstraction operation then is applied so that $\mathcal{A}$ and its extensions can be symbolically represented by a single answer A. Using answer subsumption, this abstraction can be taken only if needed during program execution. Abstractly, partial order answer subsumption with abstraction uses the declaration

[1] Since count and sum are not idempotent their semantics is based on multi-sets, rather than sets. Incorporating these as tabling features requires modifying their semantics to be set-based, in a manner similar to aggregation ASP systems (cf. e.g. [2]) .

```
:- table p(_,_,po(rel/2,abs/3)).
```

where `rel/2` is a partial order, and `abs/3` is the abstraction operation. Section 4.2 provides a detailed example of how such an approach is used to analyze a process logic.

Complexity. Consider a ground program P where some predicate p/n is declared to use lattice answer subsumption with join predicate $op/3$. Note that any answer to a subgoal of p/n need be compared to at most one other answer to compute a join. Thus if $op/3$ has constant cost, lattice answer subsumption adds no overhead in terms of complexity to evaluating P. However, for partial order answer subsumption, an answer to a subgoal of p/n might in principle be compared to all other answers for p/n, which in the worst case is $atoms(P)$, the number of atoms in P. Accordingly, if $rel/2$ has constant cost, the complexity of evaluating P will be $size(P) \times atoms(P)$, regardless of whether P is definite, or is being evaluated using negation over the well-founded semantics.

3 Implementation

Both lattice and partial-order subsumption are implemented through a compiler translation that introduces specialized code to manipulate answers in the table.

We first describe the implementation of lattice answer subsumption. As discussed, for simplicity of presentation, we assume that the predicate tabled using answer subsumption returns only ground answers. Consider again the example of shortest path using `min/3` as a join operator (Figure 1), in which the query finds all distances from a single source – e.g. a query such as `sp(a,Y,M)`. The XSB compiler transforms `sp/3` to the code in Figure 2. The first two subgoals in the body of the transformed version of `sp/3` (line 3) gain access to the table created on the call to `sp(a,Y,M)`; access in XSB is through the generator choice point for the table, obtained through the choice point register, `Breg` (see [9] for details). `Cs` is a pointer to the table entry for the current call, and `Skel` is a term containing the free variables of the query, which

```
   :- table sp/3.
   sp(X,Y,M) :-
        '_$savecp'(Breg), breg_retskel(Breg,3,Skel,Cs),
        (nonvar(M) -> instantiation_error ; true),
5       excess_vars(Skel,[M],[],Vars),
        copy_term(t(Vars,Skel,M),t(Vars,OSkel,OM)),
        'sp$$'(X,Y,NM),
        ('_$$get_answers'(Cs,OSkel,AnsPtr)
         -> min(OM,NM,M),
10          M \== OM,
            delete_answer(Cs,AnsPtr)
         ;  M = NM ).

   'sp$$'(X,Y,1) :- edge(X,Y).
15 'sp$$'(X,Y,N) :- sp(X,Z,N1),e(Z,Y),N is N1+1.
```

Fig. 2. Example Code for Lattice Answer Subsumption

for sp(a,Y,M) is the term ret(Y,M). Since tabled answers in XSB contain only bindings to variables in the call, the free variables are necessary to retrieve answers from the table. Line 4 throws an error if the argument using answer subsumption is not a variable, as the code of Figure 2 is not correct in that case. Lines 5 and 6 generate variants of other terms that will be needed to retrieve answers from the table. In our example, OSkel is ret(Y,OM) – note that Y is in the call, but OM is free. After this setup, line 7 calls the original code (transformed to 'sp$$'/3) to derive answers. On success of 'sp$$'/3 (line 8), a previous answer whose bindings unify with OSkel is obtained from the table, if it exists. For instance, if the success of sp$$/3 in line 7 bound Y to b, the answer in the table for sp(a,Y,M) that has Y bound to b is obtained if it exists, binding OM to the third argument of that answer. Note that the use of lattice answer subsumption, together with the safety assumption ensure that there is at most one such answer. If the answer does exist, the old value OM is joined with the new NM from the answer just returned (line 9). If the join differs from the old answer (line 10), the old answer is deleted (line 12) and the clause succeeds. Further compilation into byte code ensures that an answer is added to the table whenever a clause of a tabled predicate succeeds (here, in line 10 or 12). If the joined value M is the same as the value OM in the old answer, the computation fails in order to search further. If there is no previous answer in the table (line 12), then the clause succeeds. Note that the setup portion, (lines 1-6) are executed once per call; lines 8-12 are executed for each answer.

Next we describe the implementation of the same program and query using partial order answer subsumption. Again the compiler transforms the program to perform the table manipulations (Figure 3). The first 6 lines of setup are identical to the lattice case; partial order subsumption differs only in how it treats answers. In lines 8-9 the table is checked to see if any previous answer is the same as or subsumes the new answer. If so, then the computation fails. (Note that if the new answer is subsumed by an answer already in the table, then the table will not contain any answer subsumed by the new one.) Assuming the new answer is not subsumed by any old answer, lines 10-12 use findall/3 to collect pointers to all answers subsumed by the new one, and in line

```
  :- table sp/3.
  sp(X,Y,M) :-
       '_$savecp'(Breg), breg_retskel(Breg,3,Skel,Cs),
       (nonvar(M) -> instantiation_error ; true),
5      excess_vars(Skel,[M],[],Vars),
       copy_term(t(Vars,Skel,M),t(Vars,OSkel,OM)),
       'sp$$'(X,Y,NM),
       \+ ('_$$get_answers'(Cs,OSkel,_),
           (OM == NM ; '<'(OM,NM)) ),
10     findall(AnsPtr,
                ('_$$get_answers'(Cs,OSkel,AnsPtr), '<'(NM,OM)),
                AnsPtrs),
       (member(AnsPtr,AnsPtrs), delete_answer(Cs,AnsPtr), fail
        ;
15      M=NM ).
```

Fig. 3. Example Code for Partial Order Answer Subsumption

```
  :- table reachable/2.
  reachable(S,M) :-
      '_$savecp'(Breg), breg_retskel(Breg,2,Skel,Cs),
      (nonvar(M) -> instantiation_error ; true),
5     excess_vars(Skel,[M],[],Vars),
      copy_term(t(Vars,Skel,M),t(Vars,OSkel,OM)),
      'reachable$$'(S,NM),
      findall(OM-AnsPtr,'_$$get_answers'(Cs,OSkel,AnsPtr),OldAnswerPtrs),
      collect_ans(OldAnswerPtrs,OldAnswers),
10    omega_abs(OldAnswers,NM,AbsM),
      \+ (member(OM-_,OldAnswerPtrs),
          (OM == AbsM ; omega_gte(OM,AbsM)) ),
      (member(p(OM,AnsPtr),OldAnswerPtrs),
       omega_gte(AbsM,OM)), delete_answer(Cs,AnsPtr), fail
15     ;
       M=AbsM ).
```

Fig. 4. Example Code for Partial Order Answer Subsumption with Abstraction

13, they are deleted from the table. The new answer is added to the table upon the clause's success in line 15.

Finally we describe the transformation for Partial Order Subsumption with Abstraction. The example transformation for PT Net Reachability (Section 4.2) is shown in Figure 4. The declaration for this example is assumed to be `:- table reachable(_,po(omega_gte/2,omega_abs/3)).` Again the setup and call in line 7 are the same as the previous cases. On return of a newly computed answer, line 8 collects all old answers and the pointers to them, and line 9 separates out just the old answers, which are input to the abstraction operator in line 10. Then in the rest of the code, the abstracted answer is used in place of the computed answer, as follows. First, lines 11-12 check whether the new answer is already subsumed by an existing answer, in which case the clause fails. Otherwise, lines 13-14 delete all old answers subsumed by (the possible abstraction of) the new answer. And in line 15, we return the new (possibly) abstracted answer.

4 Performance and Applications

In this section we benchmark and analyze application TLP programs.[2]

4.1 Answer Subsumption in Support of Social Network Analysis

The field of Social Network Analysis (SNA) (cf. [13]) studies the behavior of groups through the relations among their members. In SNA a social network is a graph that is

[2] All benchmarks were performed on a MacBook pro laptop, with a 2 Ghz Intel Core Duo CPU and 2 GB of RAM. Multi-threading was not used for these benchmarks, so only one core was utilized. All times are in seconds, and all measures of space are in bytes. Table space in XSB includes storage space for subgoals and answers along with space allocated for copying areas, answer hash buckets, etc. All benchmark programs are available by anonymous CVS from xsb.sourceforge.net in the `benches` directory of the module `mttests`.

analyzed to determine measures of connectivity or of balance, partitioned into subcomponents according to an optimality criterion, or analyzed in other ways. Logic programming offers promise for SNA: it is easy to specify properties of vertices ("male,"lives-in-city") and of edges ("father-of","exchanges-needles-with"); and SNA properties can be declaratively analyzed by TLP or ASP systems. A factor in many types of SNA (e.g. [11]) is the *coherence* of a (sub-)graph: a numeric measure based on the shortest paths between all vertices in the subgroup (the metric for distance may be defined on different edge types, or their combination).

We begin our benchmarking with the shortest path predicate `sp/3` of Figure 1 which uses lattice answer subsumption. In `sp/3`, distance between two vertices is defined simply as the minimal number of edges between them. While there are several well-known algorithms to determine shortest paths in graphs with non-negative edge weights, the problem offers excellent scope for analyzing various aspects of answer subsumption. Table 1 shows the scalability of the goal `sp(From,To,Dist)` on randomly generated graphs with N vertices and edges. These graphs are sparse in the sense that they are largely unconnected: the number of answers is substantially below the N^2 answers the query would return for a fully connected graph. As Table 1 shows, `sp/3` scales linearly in answers up to the amount of core memory available.

The standard algorithm for finding shortest paths to all nodes from a single source node is Dijkstra's algorithm [4]. The difference, between that algorithm and the underlying algorithm for answer subsumption, is in the scheduling. In Dijkstra's algorithm, the next node chosen to expand is the one with shortest distance from the source node. So the "wave front" of the search is expanded by choosing the nearest non-expanded node. This tabling algorithm expands the wave front based on the number of edges from the source, independent of the weights on the edges. For our examples where each edge is assumed of weight 1, the algorithm corresponds to Dijkstra's. But with varying edge weights, answer subsumption (as implemented here) may be suboptimal.

Sparse graphs are unlikely to have many different paths between two vertices: accordingly Table 1 does not check the efficiency of all aspects of lattice answer subsumption such as accessing previously derived answers to compute a join, and possibly deleting them. These factors are measured in Table 2, which benchmarks various predicates on graphs of 1000 vertices and $N = 2 \times 1000, 4 \times 1000 \ldots 512 \times 1000$ edges. In addition to benchmarking `sp/3` with lattice and partial order answer subsumption, Table 2 measures two new predicates shown in in Figure 5. The first, `reach/3` is a simple transitive closure predicate that does the same work as `sp/3` *except* for answer

Table 1. Scalability of lattice `sp/3` on sparse graphs where $|edges| = |vertices|$

Vertices	Time	Table Space	Answers
25000	1.7	44,146,000	960,588
50000	7.5	198,905,244	4,324,742
75000	12.8	307,611,736	6,683,493
100000	9.8	212,186,848	4,611,563
125000	57.6	1,128,215,852	24,617,754

```
:- table reach/3.
reach(X,Y,1):- edge(X,Y).
reach(X,Z,1):- reach(X,Y,N1),edge(Y,Z), _N is N1 + 1.

:- table sp_del(X,Y,lattice(min/3)).
sp_del(X,Y,D):- edge(X,Y,D).
sp_del(X,Z,D3):- sp_del(X,Y,D1),edge(Y,Z,D2),D3 is D1 + D2
```

Fig. 5. Predicates for shortest path and transitive closure

Table 2. Comparison of approaches on dense graphs where $|edges| = N \times |vertices|$

Avg. Verts//Node	2	8	32	128	512
`sp/3`-Lattice					
Time	2.3	13.2	52.1	211.9	880
Table Space	26,249,826	41,213,664	41,213,664	41,213,664	41,213,664
Answers	631,509	1,000,000	1,000,000	1,000,000	1,000,000
`sp/3`-PO					
Time	4.1	16.5	56	218.2	890
Table Space	26,249,860	41,213,688	41,214,084	41,214,084	41,214,084
`reach/3`					
Time	0.88	3.47	12.5	53.2	238
Table Space	26,241,796	41,205,624	41,205,624	41,205,624	41,205,624
`sp_del/3`					
Time	4.2	104.0	329	845	2392
Table Space	27,198,048	41,203,908	41,290,552	41,322,464	41,345,080
Answers	655,221	999,000	1,000,000	1,000,000	1,000,000
Deletes	281,834	2,416,658	4,917,751	6,960,565	8,407,883

subsumption; the second is a shortest path predicate, `sp_del/3`, for which distance is a function of weights for each edge. As can be seen from Table 2, once the graphs are fully connected, `sp/3` is linear in the number of edges, regardless of whether a lattice or partial order is used for answer subsumption. The space required is virtually the same for both approaches, and the times are also quite similar, indicating that the worst-case complexity of partial order answer subsumption (Section 2) is not a factor for these examples.

Tests of `reach/3` on the same graphs show a similar growth in times to `sp/3` and virtually the same space. `reach/3` is about 3-4 times faster, indicating the overhead for answer subsumption on this simple example; it should be noted that shortest path uses answer subsumption extremely heavily, and the overhead for answer subsumption on most other programs will be much smaller. Profiling of `sp/3` shows that no deletions are performed on either the sparse-graph or dense-graph benchmarks. In these experiments, shorter paths are discovered first; when non-optimal paths are derived later, so that execution of answer subsumption code fails on the comparison in line 10 of Figure 2, and a deletion need not be performed. To test the cost of deletions, `sp_del/3`

```
:- table pref_distance/4.
pref_distance(X,Y,1,_):- edge(X,Y).
pref_distance(X,Z,N,Max):-
   pref_distance(X,Y,N1,Max),
   edge(Y,Z), N is N1 + 1, N < Max,
   tnot(preferred_distance(X,Z,N,Max)).  % XSB's tabled negation

:- table preferred_distance/4.
preferred_distance(X,Y,N,Max):- pref_distance(X,Y,M,Max),M < N.
```

Fig. 6. A Program to Compute Shortest Path using Negation

was tested on graphs where each edge fact also contains a randomly-generated cost.[3] Table 2 shows that deletion imposes overhead in terms of time, but virtually no overhead in terms of space.

Comparison of answer subsumption to negation. In addition to using answer subsumption, shortest paths can also be computed through negation, as by the predicate `pref_distance/4` in Figure 6, which concludes a given path between two vertices is shortest if no other shorter path is derivable. This approach is similar to a preference-based approach, where a shorter path is prefered to a longer one. Note that when answer subsumption is not used for shortest path, a program may have an infinite model if the underlying graph has cycles. To ensure termination, `pref_distance/4` has as its fourth argument the maximum diameter of a graph. The need for a maximum distance, together with the requirement that calls to negative literals be ground, increases the complexity of determining shortest path. Not surprisingly, experiments show that `pref_distance/4` scales poorly compared to the approaches based on answer subsumption. Since many ASP grounders may require users to program shortest path in a ground manner similar to that of `pref_distance/3`, experiments on ASP grounders were also performed. The experiments showed poor scalability compared to answer subsumption. Overall, these results indicate that answer subsumption can play an important role for ASP grounding, either by implementing answer subsumption within a grounder, or by using TLP as a grounder as in XSB's XASP package.

4.2 Answer Subsumption and Abstract Interpretation

Net-style formalisms, such as Petri Nets, Workflow Nets, etc. have been used extensively for process modeling. Reachability is a central problem in analyzing properties of such nets, to which properties such as liveness, deadlock-freedom, and the existence of home states can be reduced. However, many interesting net formalisms cannot guarantee a finite number of configurations in a given net, so abstraction methods must be applied for their analysis.

For instance, the lack of finiteness is a problem in analyzing Place/Transition (PT) Nets. PT nets have no guard conditions or after-effects, and do not distinguish between

[3] The graphs used for `sp_del/3` have different randomly-generate edge relations than those for `sp/3` and so have a different number of answers.

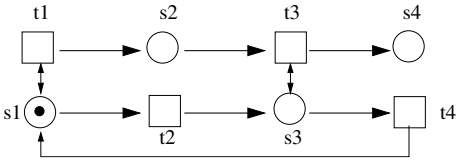


Fig. 7. A PT-net and configuration with an infinite number of reachable configurations

```
:- table reachable(_,po(omega_gte/2,omega_abs/3)).
reachable(InConf,NewConf):-
        reachable(InConf,NewConf),
        hasTransition(Conf,NewConf).
reachable(InConf,NewConf):- hasTransition(InConf,NewConf).
```

Fig. 8. Top-level predicate for PT net reachability

token types. However, PT nets do allow a place to hold more than one token, leading to a potentially infinite number of configurations. This can be seen in the simple network of Figure 7 (from [3]) in which transitions are denoted by squares and places by circles. Each transition removes one token from the places that are the sources of its input edges and adds one token to each place at the target of each of its output edges. Starting from the configuration in Figure 7, repeated application of transition `t1` leads to place `s2` containing an unbounded number of tokens; repeated application of the sequence `t1,t2,t3,t4` leads to place `s4` containing an unbounded number of tokens.

Despite such examples, reachability in PT nets is decidable and can be determined using an abstraction method called ω-sequences, (see e.g. [3]). The main idea in determining ω sequences is to define a partial order $\geq_\omega$ on configurations as follows. If configurations C_1 and C_2 are both reachable, C_1 and C_2 have tokens in the same set PL of places, C_1 has at least as many tokens in each place as C_2, and there exists a non-empty $PL_{sub} \subseteq PL$, such that for each $pl \in Pl_{sub}$ C_1 has strictly more tokens than C_2, then $C_1 >_\omega C_2$. When evaluating reachability, if C_2 is reached first, and then C_1 was subsequently reached, C_1 is abstracted by marking each place in PL_{sub} with the special token ω which is taken to be greater than any integer. If C_1 was reached first and then C_2, C_2 is treated as having already been seen.

Tabling combined with partial order answer subsumption requires slightly over 100 lines of code to model reachability in PT nets using ω-sequences. Due to space restrictions, the program cannot be fully described here, but the top-level reachability predicate is shown in Figure 8. Despite its succinctness, it can evaluate reachability in networks with millions of states in a few minutes. This use of tabling to determine reachability in PT nets can be seen as a special case of tabling for abstract interpretation (cf. [5] and other works). However the framework for answer subsumption described here allows tabling to be used to efficiently perform abstract interpretation within a general Prolog system

4.3 Scalability for Multi-valued and Quantitative Logics

The technique of program justification (cf. e.g. [7]) has been used for debugging tabled programs that cannot be debugged by traditional means. Here, we consider justification in the context of the Silk system, currently under development at Vulcan, Inc. Silk is a commercial knowledge representation and rule system built on top of Flora-2, which is implemented using XSB. One of the salient features of Silk is its default reasoning, which is based on a parameterized argumentation theory evaluated under the well-founded semantics [12]. One issue in using Silk is that knowledge engineers must have a way of understanding the reasoning of the system, a task complicated by the use of the well-founded semantics and the intricacies of the argumentation theory. We describe an experimental approach to justification of Silk-style argumentation theories using multi-valued logics.

As noted in [12], argumentation theories in Silk are usually extensions of the default theories of Courteous Logic Programs (CLP) and are based on two user-defined predicates: `opposes/2` and `overrides/2`. Two atoms *oppose* each other if no model of a program can contain both atoms: an atom and its explicit negation oppose each other, but opposition can capture many other types of contradictions. Given two opposing atoms, one atom may *override* the other, and so be given preference. For atoms A_1 and A_2, if A_1 and A_2 are both derivable and oppose each other but neither overrides the other, A_1 and A_2 mutually *rebut* each other. If in addition A_1, say, overrides A_2, A_1 *refutes* A_2[4]. Within Silk and Flora-2, the compilation of an argumentation theory ensures that rebutted atoms have an undefined truth value, as do atoms that refute themselves (i.e. if the `overrides/2` predicate is cyclic). However, for justification, it is meaningful to distinguish those facts that are undefined due to a negative loop in the argumentation theory from those that are undefined due to a negative loop in the program itself. In addition, it is meaningful to distinguish an atom that is true because it overrides some other atom, from an atom whose derivation does not depend on the argumentation theory. Similar distinctions can be made for default false literals leading to the truth lattice shown in Figure 9.

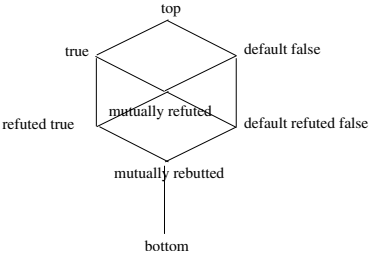


Fig. 9. A Truth Lattice for a Simplified Version of Courteous Argumentation Theory

An atom-based argumentation theory is added to a program by an easy *standard* transformation [12]. Each clause H :- B whose head is a defeasable predicate is rewritten as

[4] In [12] argumentation theories are built on named rules, here we base them on derived atoms.

H :- $B, tnot(defeated(H))$; clauses for non-defeasible predicates are not altered. To obtain support for a justification a *multi-valued* transformation was used instead of the standard transformation. First, the lattice of Figure 9 was programmed in Prolog for use by answer subsumption. Next, each clause H :- B whose head was a defeasible predicate was rewritten as H :- $B, defeated(H, Reason)$, where `defeated(H,Value)` indicates the truth value of H on the lattice of Figure 9.

Experiments were performed on synthetic programs to compare the implementation of a Silk argumentation theory using the standard transformation to the new multi-valued transformation. Synthetic programs were tested containing a large number of mutually recursive defeasible rules, together with a large proportion of refuted and rebutted atoms. These tests indicate that the use of lattices may increase the time for total query evaluation by two to three times, well within an allowable increase for a justification system. Surprisingly, the multi-valued transformation of the argumentation theory sometimes take *less* table space, due to the space overhead incurred by XSB to maintain conditional answers (i.e. answers whose truth value is undefined in the well-founded semantics). We stress that these results are preliminary in the sense that the behavior of the synthetic programs may not resemble that of practical programs that use defeasible logic. However, the heavy use of defeasibility in the synthetic programs gives reason to believe that the time overhead may well be much less in practical programs than observed here. Together the results show that multi-valued logics are a promising approach for justification of defeasible logics, whether these logics occur as part of Silk or are used directly in a TLP system such as XSB.

5 Conclusions

This paper has described how answer subsumption can be used for applications in quantitative reasoning, abstract interpretation and multi-valued logics. To use answer subsumption, a programmer need only write a join, comparison, or abstraction operation in Prolog and make the appropriate declarations. As shown in Section 3, the main implementational requirements of answer subsumption are 1) an efficient way to compare a new answer to appropriate answers in a table; and 2) an efficient way to delete subsumed answers. These features only access table space, so that they can be implemented by any tabling system, regardless of the engine architecture. Since XSB's table space is trie-based, other Prologs with trie-based tabling such as YAP or Ciao may be able to port XSB's engine code directly[5].

Answer subsumption is restricted to stratified programs in the current version of XSB. Future work includes the ability to use answer subsumption in non-stratified programs, and to add program constructs that allow non-idempotent aggregate operations to be computed, such as sum and count. However, the main work will be incorporating answer subsumption in applications such as program analysis in compilers, grounders for ASP solvers, and para-consistent and quantitative programs.

[5] A significant amount of low-level C code has been ported from XSB to YAP to support a different feature termed call subsumption.

Acknowledgements. The authors would like to thank Prasad Rao who helped implement the original version of answer subsumption, and Neng-Fa Zhou for a helpful discussion of tabling declarations.

References

1. Damásio, C.V., Pereira, L.M.: Monotonic and residuated logic programs. In: Benferhat, S., Besnard, P. (eds.) ECSQARU 2001. LNCS (LNAI), vol. 2143, pp. 748–759. Springer, Heidelberg (2001)
2. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate functions in disjunctive logic programs. In: IJCAI (2003)
3. Desel, J., Reisig, W.: Place/transition Petri nets. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 122–174. Springer, Heidelberg (1998)
4. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–277 (1959)
5. Kanamori, T., Kawamura, T.: Abstract interpretation based on OLDT resolution. JLP 15, 1–30 (1993)
6. Kifer, M., Subrahmanian, V.S.: Theory of generalized annotated logic programming and its applications. JLP 12(4), 335–368 (1992)
7. Pemmasani, G., Guo, H., Dong, Y., Ramakrishnan, C.R., Ramakrishnan, I.V.: Online justification for tabled logic programs. In: Kameyama, Y., Stuckey, P.J. (eds.) FLOPS 2004. LNCS, vol. 2998, pp. 24–38. Springer, Heidelberg (2004)
8. Riguzzi, F., Swift, T.: Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In: ICLP (2010) (to appear)
9. Sagonas, K., Swift, T.: An abstract machine for tabled execution of fixed-order stratified logic programs. ACM TOPLAS 20(3), 586 (1998)
10. Swift, T.: Tabling for non-monotonic programming. AMAI 25(3-4), 201–240 (1999)
11. Valente, T., Fujimoto, K.: Bridges: Locating critical connectors in a network. Social Networks (2010) (to appear)
12. Wan, H., Grossof, B., Kifer, M., Fodor, P., Liang, S.: Logic programming with defaults and argumentation theories. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 432–448. Springer, Heidelberg (2009)
13. Wasserman, S., Faust, K.: Social Network Analysis. Cambridge University Press, Cambridge (1994)

Embracing Events in Causal Modelling: Interventions and Counterfactuals in CP-Logic

Joost Vennekens[1,⋆], Maurice Bruynooghe[2], and Marc Denecker[2]

[1] Campus De Nayer, Jan De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
[2] Dept. Comp. Sc., K.U. Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract. This paper integrates Pearl's seminal work on probability and causality with that of Shafer. Using the language of CP-logic, it transposes Pearl's analysis of interventions and counterfactuals to the semantic context of Shafer's probability trees. This gives us definitions that work not on the level of random variables, but on the level of Humean events. There are some tangible benefits to our approach: we can elegantly handle counterfactuals in the context of cyclic causal relations, and are able to consider interventions that are both more fine-grained and more elaborate than Pearl's.

1 Introduction

Causal statements implicitly refer to things that happen. For instance, the statement "syphilis causes paresis" refers to the biological process of syphilis spirochaetes damaging certain brain cells. It is by means of this process that patients who at first exhibit only a syphilis infection will eventually come to exhibit paresis as well. The causal statement itself leaves the details of this process implicit and just asserts its existence: "somehow," it says, "syphilis causes paresis." Each causal statement of this form (e.g., dropping a vase causes it to break, a voltage drop causes an electrical current, being born in Belgium causes Belgian citizenship) implicitly refers to some such implicit process, to some implicit thing that happens to generate the effect from its cause.

In [9], Shafer recognizes the importance of this dynamic aspect of causality and introduces a specific term for these implicit things-that-happen: *Humean events*. The adjective "Humean" is added to avoid confusion with the technical meaning that the term "event" has in probability theory. That is, a Humean event is *not* a subset of some sigma algebra, nor a set of possible outcomes of some experiment, but simply, as in everyday language, something that happens. Following the line of thought from the preceding paragraph, Shafer places this concept at the centre of his work, using probability trees to model causal systems as sequences of Humean events. In this context, a random variable (RV), for instance, is simply something which gets assigned a certain value at some point in the sequence of events.

⋆ Partially supported by IWT-Vlaanderen.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 313–325, 2010.
© Springer-Verlag Berlin Heidelberg 2010

A different approach is taken by Pearl [7]. His probabilistic causal models (PCMs) represent the relations between RVs as they hold in stable states of the domain (i.e., when no more Humean events are happening). His models, therefore, do not care about which event causes the value of an RV, or when this happens. For instance, [5] uses the term "event" to refer to an assignment $\boldsymbol{X} = \boldsymbol{x}$ of values to RVs. Unlike proper Humean events, such assignments are not things that intrinsically happen at a certain point in time, as part of a larger sequence of events. Underlying one such an assignment, there may be many different Humean events, happening at different times, and it may even depend on the context (i.e., the outcome of previous events) which events are involved in a particular assignment. For instance, a patient being in hospital ($Hospital = true$) may be the result of a car crash yesterday (a Humean event), but also, if he managed to avoid the crash, of slipping on the stairs this morning (a different Humean event). Shafer's account therefore presents a more refined view on the dynamic aspects of causality than Pearl's. However, it lacks Pearl's thorough treatment of *interventions* and *counterfactuals*.

In [10] (and before at JELIA 2006) , we presented a logical language called *CP-logic* (the "CP" stands for causal and probabilistic), which offers a succinct syntax for describing classes of Shaferian probability trees. We will now use this language to provide the semantics of probability trees with a suitable notion of interventions. In the same way as Shafer's account of causality in terms of Humean events can be seen as a refinement of the RV-based model employed by Pearl, our account of interventions and counterfactuals in CP-logic will constitute a useful refinement of the account found in [7].

2 Preliminaries: CP-Logic

We assume familiarity with classical logic and briefly recall CP-logic. For simplicity, we omit function symbols and make the Herbrand assumption of identifying interpretations with sets of ground atoms. More detail can be found in [10].

A *causal probabilistic law*, or *CP-law* for short, is a statement r of the form:

$$\forall \boldsymbol{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \phi.$$

Here, ϕ is a first-order formula, the A_i are atoms, the α_i are non-zero probabilities, and the tuple of variables $\boldsymbol{x}$ contains all free variables in ϕ and the A_i. Intuitively, for each $\boldsymbol{x}$, $\phi(\boldsymbol{x})$ causes some implicit Humean event, which will result in at most one of the effects $A_i(\boldsymbol{x})$. For each i, α_i is the probability of A_i being the resulting effect. Therefore, we require $\sum \alpha_i \leq 1$. If the equality holds, exactly one A_i is caused; otherwise, it is possible that the Humean event passes without any (visible) effect on the state of the world. For mathematical uniformity, we introduce the notation $r^{=}$ to refer to r itself if the equality holds, and otherwise to the CP-law: $\forall \boldsymbol{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \vee (\text{—} : 1 - \sum_i \alpha_i) \leftarrow \phi$. Here, the dash represents the possibility of there being no (visible) effect. For a CP-law r, we denote $\boldsymbol{x}$ by $vars(r)$, refer to ϕ as $body(r)$, to the sequence $(A_i, \alpha_i)_{i=1}^n$ as $head(r)$, and to $(A_i)_{i=1}^n$ as $head_{At}(r)$. We also allow the condition $body(r)$ to be

omitted, which means that the Humean event always happens. We abbreviate $(A:1) \leftarrow \phi$ by $A \leftarrow \phi$. A *CP-theory* is a finite set of CP-laws.

We now recall the semantics of such a CP-theory. For simplicity, we will assume that each precondition $body(r)$ is a positive formula. The semantics of negation can be found in [10].

Our basic semantic construct is a probability tree [9], i.e., a finite tree in which each edge is labeled with a probability, such that the labels of all edges leaving the same internal node always sum up to one. Intuitively, such a tree $\mathcal{T}$ represents a probabilistic process: each node is a state of the process and, together, the edges leaving a node represent a Humean event that causes a probabilistic transition to one of its children. The root is the initial state and the leaves are final states. Let us denote by $\pi_{\mathcal{T}}$ the probability distribution that a tree $\mathcal{T}$ defines over its leaves, i.e., for each leaf l, $\pi_{\mathcal{T}}(l)$ is the product of the labels of the edges that lead to l. We now define which probability trees $\mathcal{T}$ correspond to a given set of CP-laws C. Basically, the events that happen in $\mathcal{T}$ should follow the blueprints given by the CP-laws in C. An *occurrence* of a CP-law $r \in C$ is the result $r[\boldsymbol{x}/\boldsymbol{c}]$ of replacing the variables $\boldsymbol{x} = vars(r)$ by constants $\boldsymbol{c}$. The *grounding* $grnd(C)$ of C consist of all occurrences of CP-laws $r \in C$ that can be thus constructed. Because we have no function symbols and only a finite set of constants, this grounding is finite.

Condition 1. *Events correspond to occurrences of CP-laws, i.e., there exists a mapping $\mathcal{E}$ from the internal nodes s of $\mathcal{T}$ to $grnd(C)$ such that we can label the nodes and the edges of $\mathcal{T}$ as: if $\mathcal{E}(s) = r[\boldsymbol{x}/\boldsymbol{c}]$ and $head((r[\boldsymbol{x}/\boldsymbol{c}])^{=}) = (A_i, \alpha_i)_{i=0}^{n}$, then the children of s are nodes $s_0, \ldots, s_n$ such that the label $lbl(s_i) = A_i$ and $lbl((s, s_i)) = \alpha_i$. For uniformity, the root is labeled with '—' ("no-effect").*

Condition 2. *An occurrence that has already happened cannot happen again. Formally, let $Anc(s)$ be the ancestors of s (not including s itself), and let $\mathcal{R}(s)$ be the set of occurrences that have not yet happened in s (i.e., $\mathcal{R}(s) = grnd(C) \setminus \{\mathcal{E}(s') \mid s' \in Anc(s)\}$). Then each $\mathcal{E}(s)$ must be in $\mathcal{R}(s)$.*

Finally, we relate the nodes in the tree to the effects and preconditions of the occurrences. Recall that each node s in a probability tree corresponds to a potential state of the domain. We represent this state by an interpretation (i.e., a set of ground atoms) as follows: $\mathcal{I}(s) = \bigcup_{t \in Anc(s) \cup \{s\}} l(t)$, where $l(t) = \{\}$ if $lbl(t) = \text{`}-\text{'}$ and $\{lbl(t)\}$ otherwise.

Condition 3. *The precondition of the occurrence of a CP-law that happens in a node must be satisfied in its state, i.e., for all internal nodes s: $\mathcal{I}(s) \models body(\mathcal{E}(s))$.*

The three conditions above describe when a probability tree unfolds according to the CP-laws of a theory C. We call such a tree an *execution model* of C if it is also complete in the following sense:

Condition 4. *$\mathcal{T}$ cannot be extended, i.e., for each leaf l and $r \in \mathcal{R}(l)$, we have that $\mathcal{I}(l) \not\models body(r)$.*

While we lack space for an example at this point, Fig. 1 on page 318 show an execution model for Example 1. By defining a probability distribution $\pi_{\mathcal{T}}$ over its leaves, an execution model $\mathcal{T}$ induces a probabilistic possible world semantics: the probability $\pi_{\mathcal{T}}(S)$ of an interpretation S is $\sum_{\mathcal{I}(l)=S} \pi_{\mathcal{T}}(l)$. The probability of a formula ϕ is then $\pi_{\mathcal{T}}(\phi) = \sum_{S \models \phi} \pi_{\mathcal{T}}(S)$. In [10], it was shown that each execution model $\mathcal{T}$ of a CP-theory C defines the *same* possible world semantics $\pi_{\mathcal{T}}$, which we therefore also denote as π_C. This result demonstrates one sense in which causal statements are indeed justified in leaving implicit the Humean events to which they refer: as long as we are only interested in the final states that will eventually be reached (and not in any intermediate states), the properties of these implicit events do not matter.

CP-logic normally distinguishes exogenous from endogenous predicates [10]. Here, however, we will save some space by writing "$\forall \boldsymbol{x}\ (Exo(\boldsymbol{x}) : *) \leftarrow$" to say that *Exo* should really be exogenous, but that we will treat it as an endogenous predicate caused with some unknown probability $*$. For the examples in this paper, it is easy to see that the proper treatment of exogenous predicates would yield the same results.

CP-logic is closely related to Probabilistic Causal Models (PCMs). A PCM that contains only boolean equations with boolean RVs can easily be translated to a CP-theory that contains for each PCM equation a single CP-law that propagates the value of the body of the equation to its head. Conversely, CP-theories can also be translated to PCMs. If the CP-theory contains no cycles (to be discussed below), then this is trivial. Otherwise, artificial RVs representing common causes underlying all of the variables in a cycle are needed. See [10] for details.

3 Interventions

The goal of this section is to provide CP-logic with definitions that capture Pearl's intuitions about interventions. Let us briefly recall some of the formal tools used by Pearl. A *structural model* is a set of *structural equations*, each of which defines the value of one RV in terms of the values of some other RVs. A *Probabilistic Causal Model (PCM)* consists of a structural model, together with a probability distribution over the values of its exogenous RVs (i.e., those without a defining equation). An important restriction is that, given any assignment of values to the exogenous RVs, this set of equations must have a unique solution. Typically, this is ensured by requiring an acyclic set of equations.

An *intervention* in a PCM is of the form $do(\boldsymbol{X} = \boldsymbol{x})$ with $\boldsymbol{X}$ of a tuple of endogenous variables and $\boldsymbol{x}$ a tuple of values. *Performing* this intervention means replacing the defining equation of each $X_i \in \boldsymbol{X}$ by $X_i := x_i$. In a logical framework, we can view a PCM as an acyclic set of equations $A := \phi$ where A is a ground atom and ϕ a sentence. An intervention $do(\boldsymbol{A} = \boldsymbol{a})$ then assigns a tuple of truth values $\boldsymbol{a} \in \{\mathbf{t}, \mathbf{f}\}^n$ to the tuple of atoms $\boldsymbol{A}$, while removing their defining equations. In this paper, we will not consider interventions at the level of random variables (i.e., atoms) as Pearl does, but look instead at interventions that add and/or prevent CP-laws to/in a theory. Because CP-logic represents a

causal system in a modular way as a set of causal laws, this kind of intervention is already built into its semantics.

Definition 1. *Let C be a CP-theory. An* intervention *is a pair (R, A) with R a subset of C (a* preemption*) and A a set of laws not in C (an* addition*). The result of* performing *(R, A) on C, denoted $C\!\not\downarrow\!(R, A)$, is the CP-theory $(C \setminus R) \cup A$.*

Let r be $\forall \boldsymbol{x}\ (A_1 : \alpha_1) \vee \cdots \vee (A_n : \alpha_n) \leftarrow \phi$. For atoms $\mathbf{A} \subseteq \{A_i \mid 1 \leq i \leq n\}$, let $r|^-_{\mathbf{A}}$ be this CP-law without the atoms $\mathbf{A}$, i.e.,

$$r|^-_{A_i} = \forall \boldsymbol{x}\ \bigvee_{A_i \notin \mathbf{A}} (A_i : \alpha_i)\ \leftarrow \phi.$$

Let $r|^+_{A_i}$ be the CP-law $A_i \leftarrow \phi$. An intervention that blocks only the possible effect A_i can be represented as $(\{r\}, \{r|^-_{\{A_i\}}\})$. At the other extreme, $(\{r\}, \{r|^+_{A_i}\})$ forces the outcome of the event caused by ϕ to be A_i.

We can use this notion of intervention to simulate Pearl's interventions. A Pearl-style intervention $Int = do(A_0 = \mathbf{t}, \ldots, A_m = \mathbf{t}, B_0 = \mathbf{f}, \ldots, B_n = \mathbf{f})$ corresponds to a pair (R, A) where $R = \bigcup_{j\geq 0}^{n} \{r \in C \mid B_j \in head_{At}(r)\}$ and $A = \bigcup_{j\geq 0}^{n} \{r|^-_{\{B_j \mid 0 \leq j \leq n\}} \mid r \in R\} \cup \bigcup_{i\geq 0}^{m} \{\text{“}A_i \leftarrow \text{”}\}$. Given a PCM P and its corresponding CP-theory C (as in [10]), performing intervention Int on P produces the same probability distribution as performing the intervention (R, A) on P. (Proof omitted because of space restrictions.)

4 Counterfactuals

Let us again start by recalling Pearl's treatment of counterfactuals. He considers the following class of statements:

$$\underbrace{\text{Given } X}_{\text{explanation}}, \underbrace{\text{would } Y \text{ have happened}}_{\text{prediction}}, \underbrace{\text{had we done } Z}_{\text{intervention}}?$$

Pearl's intuition is to read such a statement as: if we do the intervention Z, will Y then hold, assuming that, insofar as the intervention does not interfere, whatever lead to X in the first place will still happen in the same way as it did before? He therefore suggests the following three-step process for evaluating such statements in a PCM.

Explanation: update the *a priori* distribution over the exogenous variables by the observation X.

Intervention: apply the intervention Z to the model.

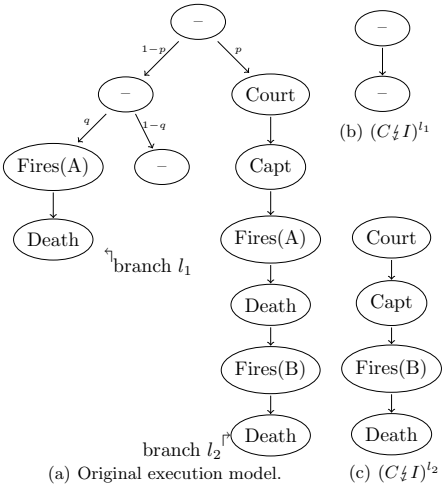
Prediction: compute the probability of Y in the resulting model, using the *a posteriori* distribution given X, i.e., we look at $P(Y \mid do(Z), X)$.

Again, our goal is to see how we can apply these intuitions in the context of CP-logic. Let us introduce our approach with an example from [7].

Example 1. A court might order the death of a prisoner. The probability of this is p. The execution is to be performed by a two person firing squad. If the court so decides, the captain of the firing squad orders both of his riflemen to fire. However, rifleman A is of the nervous type, and might shoot even if not ordered to. This happens with probability q. If at least one rifleman fires, the prisoner dies. In CP-logic, this becomes:

$$(Court : p) \leftarrow \quad (1)$$
$$Capt \leftarrow Court \quad (2)$$
$$Fires(A) \leftarrow Capt \quad (3)$$
$$(Fires(A) : q) \leftarrow \neg Capt \quad (4)$$
$$Fires(B) \leftarrow Capt \quad (5)$$
$$Death \leftarrow Fires(A) \quad (6)$$
$$Death \leftarrow Fires(B) \quad (7)$$

Note that (4) contains negation, the semantics of which was not explained in Section 2 but can be found in [10]. We could also omit $\neg Capt$ from this CP-law; the only effect would be that $Fires(A)$ might redundantly be caused twice (in branch l_2 of Fig. 1a), which does not change the semantics of this theory.



(a) Original execution model.

Fig. 1. Execution models of example 1

Pearl considers the counterfactual question: "If the prisoner is dead, what is the probability that he would still be dead if rifleman A had not shot?" Intuition puts it at the probability that the prisoner's death is due to the court ordering his execution, i.e., at $P(Court|Death) = \frac{p}{q(1-p)+p}$, because it is precisely in this case that the intervention of preventing A from firing will not save the prisoner (since B will also fire). We can reach this conclusion with a CP-logic variant of Pearl's three steps, starting from the execution model $\mathcal{T}$ in Fig. 1a.

Explanation: $\mathcal{T}$ defines a probability distribution $\pi_{\mathcal{T}}$ over its leaves. Having observed $Death$, we can update this distribution: because l_1 and l_2 are the branches where the prisoner dies, the *a posteriori* probability $P(l_1 \mid Death)$ is $q(1-p)/n$ and $P(l_2 \mid Death) = p/n$ with $n = q(1-p) + p$.

Intervention: To prevent A from firing, we apply the intervention that removes CP-laws (3) and (4). Now, as shown in Fig. 1b, if all the remaining events happen in the same way as they happened in l_1, the prisoner survives; on the other hand, as Fig. 1c shows, if they happen as they happened in l_2, he still dies.

Prediction: Having thus determined that the intervention would have prevented the prisoner's death just in case the original execution model was actually in branch

l_2 instead of l_1, we can now judge the desired probability to be $\pi_{\mathcal{T}}(l_2 \mid Death)$, which is p/n.

To make this more formal, we need a number of definitions. Let us choose an execution model $\mathcal{T}$ for our theory (it can be shown that the choice does not affect the outcome). In the explanation step, we condition on the observation X as follows.

Definition 2. *For a formula X, we define the conditional probability $\pi_{\mathcal{T}}(l \mid X)$ of a leaf l of $\mathcal{T}$ given X to be 0 if $\mathcal{I}(l) \not\models X$ and $\pi_{\mathcal{T}}(l)/\pi_{\mathcal{T}}(X)$ otherwise.*

In the intervention step, we consider what would have happened in different circumstances, under the assumption that all events that are not affected would still have happened in the same way as they originally happened.

Definition 3. *Let l be a leaf of an execution model of a CP-theory C and $B(l)$ the branch that leads to l. By C^l we denote the CP-theory that fixes the outcome of all the events that happened in $B(l)$ to the outcome they had in $B(l)$:*

$$C^l = \{r \in C \mid r \in \mathcal{R}(l)\} \cup \{r|^+_H \mid \exists s \in B : lbl(s) = H \text{ and } \mathcal{E}(parent(s)) = r\}.$$

For instance, if we fix the outcomes that occurred in l_1:

$$C^{l_1} = \left\{ \begin{array}{lll} - \leftarrow, & Fires(A) \leftarrow Capt, & Death \leftarrow Fires(A), \\ Capt \leftarrow Court, & Fires(A) \leftarrow \neg Court, & Death \leftarrow Fires(B) \end{array} \right\}$$
$$Fires(B) \leftarrow Capt,$$

We now define counterfactual probabilities as follows.

Definition 4. *Let X, Y be formulas (respectively observation and prediction) and Z an intervention. The* counterfactual probability *of Y after Z given X is*

$$Cfl(X, Y, Z) = \sum_{l \text{ is leaf of } \mathcal{T}} \pi_{(C \lightning Z)^l}(Y) \cdot \pi_{\mathcal{T}}(l \mid X).$$

In our example, performing the intervention $Z = (\{(3), (4)\}, \{\})$ on the theory produces the following $C \lightning Z$:

$$\left\{ \begin{array}{lll} (Court : p) \leftarrow, & Fires(B) \leftarrow Capt, & Death \leftarrow Fires(A), \\ Capt \leftarrow Court, & & Death \leftarrow Fires(B) \end{array} \right\}$$

In $(C \lightning Z)^{l_1}$, the first of these CP-laws reduces to "$- \leftarrow$", whereas in $(C \lightning Z)^{l_2}$, it reduces to "$Court \leftarrow$". The reader can verify that the branch in Fig. 1b indeed corresponds to $(C \lightning Z)^{l_1}$ and that Fig. 1c corresponds to $(C \lightning Z)^{l_2}$. So, $Cfl(Death, Death, Z) = 0 \cdot \frac{q(1-p)}{q(1-p)+p} + 0 + 1 \cdot \frac{p}{q(1-p)+p}$.

5 Causal Cycles

Let us now amend the previous example, by adding that each of the two soldiers will also fire when he hears the guy next to him fire. In CP-logic, this is a pretty innocuous change; we simply add:

$$Fires(A) \leftarrow Fires(B). \qquad Fires(B) \leftarrow Fires(A).$$

The only effect of this change is that now two soldiers will fire in circumstances where previously only one would, which raises the probability of $Fires(B)$ from p to $p + (1 - p)q$. Because one soldier firing already suffices to kill the prisoner, however, this does not affect the probability of his death.

The purpose of this section is to demonstrate that such cyclic causality cannot be adequately handled in Pearl's framework. To this end, we will examine a number of ways of trying to do this, and discuss what is wrong with each of them.

First, let us note that it obviously does not suffice to leave the equation for $Fires(A)$ untouched and change only the equation for $Fires(B)$ into:

$$Fires(B) := Captain \vee Fires(A).$$

Indeed, when we intervene with $Fires(B)$, for instance by sabotaging his rifle so that it will go off even without the soldier pulling the trigger, the effect should be that A also fires, which the current equation for $Fires(A)$ will not accomplish. One is therefore tempted to also make this change:

$$Fires(A) := Caption \vee Nervous \vee Fires(B).$$

Together with our modification of the equation for $Fires(B)$, however, this clearly violates the acyclicity restriction. In the appendix to [4], Halpern and Pearl present a way of lifting this restriction. However, their semantics is not what is needed for this example: they impose an equilibrium condition, where every assignment that satisfies all equations is considered possible. Therefore, it is possible for A to fire for no reason than that B does, and at the same time for B to fire for no other reason than that A does. Clearly, this is not what we want for this example: if the soldiers fire, then at least one of them should have a reason for firing that is not his comrade firing first.

This problem with the semantics of course has consequence for the results that are produced. For instance, even the tautological counterfactual "given that the prisoner survived, he would have survived" cannot be deemed true. Moreover, it also becomes impossible to judge the probability of the prisoner dying any more accurate than that it must be somewhere in the interval $[p + (1 - p)q, 1]$.

The transformation from CP-logic to Bayesian networks given in [10] would attempt to solve this problem by introducing an intermediate RV $BothFire$, replacing the equations for $Fires(A)$ and $Fires(B)$ by:

$$BothFire := Nervous \vee Captain \qquad \begin{array}{l} Fires(A) := BothFire \\ Fires(B) := BothFire \end{array}$$

This works insofar as that it generates the right probability distribution for $Death$, but it breaks down when intervention come into play. The reason is of course that this model has removed the asymmetry between $Fires(A)$ and $Fires(B)$: it no longer has the information that A, and not B, is the soldier who might fire out of nervousness. Consequently, if we ask: "given that the captain did not give the order to fire, would B have fired if we had prevented A from

firing?", then the above model has no way of knowing that the answer should be "no".

We are therefore forced into more complicated options, such as including multiple copies of our original RVs. For instance, we can have $Fires_i(A)$ and $Fires_i(B)$ for $i \in \{1, 2\}$:

$$Fires_2(A) := Fires_1(A) \vee Fires_1(B) \qquad Fires_1(A) := Captain \vee Nervous$$
$$Fires_2(B) := Fires_1(B) \vee Fires_1(A) \qquad Fires_1(B) := Captain$$

We can think here of the indices as a timestamp for the RVs, which explicitly encodes the small time delay between hearing your neighbour firing and firing yourself. In general, for a firing squad of n soldiers, in which a soldier firing causes his two neighbours to fire too, we would need n copies of each RV, allowing a "falling domino"-style propagation through the squad. An intervention such as preventing soldier S from firing should then be interpreted as an intervention with all RVs S_i, $i = 1..n$.

This solution is correct, but has the downside of blowing up the representation: simply adding the domino-effect forces us to abandon the original representation in terms of n RVs, in favour of a new representation in term of n^2 RVs. This is neither concise, nor elaboration tolerant. By contrast, in CP-logic, one just needs to add the obvious n CP-laws, in terms of the original vocabulary:

$$\begin{aligned} Fires(S_1) &\leftarrow Fires(S_2). \\ &\dots \\ Fires(S_i) &\leftarrow Fires(S_{i-1}) \vee Fires(S_{i+1}). \\ &\dots \\ Fires(S_n) &\leftarrow Fires(S_{n-1}). \end{aligned}$$

In this case, one can still defend the PCM solution as an accurate picture of reality: even though the problem description does not mention it, the propagation of shots down the firing line would in fact not be instantaneous, since each soldier takes some time to fire. However, this is not always possible. Consider, for instance, a bicycle which has a big and a small gear wheel connected by a chain. If we turn one of these wheels, the other will turn too. Moreover, this effect will be instantaneous; there is no perceptible delay between turning one and seeing the other turn. Again, CP-logic handles this fine:

$$\begin{aligned} Turn(Big) &\leftarrow Turn(Small). \\ Turn(Small) &\leftarrow Turn(Big). \end{aligned} \qquad Turn(Big) \leftarrow Peddle.$$

The reader can easily check that this behaves correctly in the face of all conceivable interventions (e.g., manually turning a gear wheel, removing the chain, blocking a gear wheel).

To represent this system as a PCM, we would need to perform the same trick as before:

$$Turn_2(Big) := Turn_1(Big) \vee Turn_1(Small). \tag{8}$$

$$Turn_2(Small) := Turn_1(Big) \vee Turn_1(Small). \tag{9}$$

$$Turn_1(Big) := Peddle \tag{10}$$

However, the newly introduced RVs are now truly artificial: one can no longer explain the difference between $Turn_1(Big)$ and $Turn_2(Big)$ in real-world terms, because, unlike in the firing squad, it cannot be the case that these RVs refer to the condition of the same gear wheel at different points in time. In this case, $Turn_2$ will tell us which wheels will turn, while $Turn_1$ has no real-world meaning. The need to invent artificial RVs in order to model the perfectly intuitive causal relation between these two gear wheels makes this an inadequate representation.

6 More Interventions

As Pearl's book explains, one of the reasons for wanting formal definitions of interventions and their effects is that human experts tend to misjudge such things. Thanks to Pearl, however, such judgments are no longer needed: all you need to do is (1) come up with a causal model of the domain, and (2) figure out how to formulate the intervention you want to consider in terms of the vocabulary of this causal model. All else, i.e., actually figuring out the effect of the intervention, can then be left up to the formal definitions.

One obvious limitation of this methodology is that it is of course not necessarily possible to formulate the intervention you want to consider in terms of the vocabulary of the causal model. In such a case, you cannot blindly let the definitions do the work, but you must still get actively involved and make some changes to the original model. While this cannot be avoided, we may hope to make this need for manual intervention as small as possible. That is, we would like our causal models and formal tools to allow as many reasonable interventions as possible to be applied "unthinkingly", without the need to tinker with the original causal model.

Some examples of reasonable interventions and associated counterfactuals:

- If we were to remove the chain from our bicycle, peddling would still cause the big gear wheel to turn, but the small wheel would no longer turn with it. Therefore, given that you originally were peddling, the big wheel would still have been turning, even if you had removed the chain.
- Suppose we could send soldier A to some additional training, which decreases his nervousness from q to $q/2$. If we had done this, the probability of the prisoner dying would have dropped from $p + (1-p)q$ to $p + (1-p)\frac{q}{2}$.
- In the example of the n person firing squad, suppose that A fired out of nervousness, which caused all other rifleman to fire as well. Would this still have happened if we had somehow managed to make our soldiers a little more stress resistant, such that they would only have fired themselves if *both* their neighbours fired, instead of at least one?

In CP-logic, each of these three examples corresponds to a straightforward intervention with the original theory, namely:

$$\Big(\{Turn(Big) \leftarrow Turn(Small), Turn(Small) \leftarrow Turn(Big)\}, \quad \{\}\Big)$$

$$\Big(\{Fires(A) : q \leftarrow\}, \quad \{Fires(A) : \frac{q}{2} \leftarrow\}\Big)$$

$$\Big(\{Fires(S_i) \leftarrow Fires(S_{i-1}) \vee Fires(S_{s+1}) \mid 1 < i < n\},$$
$$\{Fires(S_i) \leftarrow Fires(S_{i-1}) \wedge Fires(S_{s+1}) \mid 1 < i < n\}\Big)$$

Pearl, however, considers only interventions that replace the defining equations for some RVs $\boldsymbol{X}$ by truth assignments $\boldsymbol{X} = \boldsymbol{x}$. If we start from the PCMs for these example as we gave them in the previous section, then none of the three interventions listed above actually corresponds to such an intervention $\boldsymbol{X} = \boldsymbol{x}$:

- To remove the effect of the small gear wheel on the big one, Pearl would have to preempt the equation that defines whether the big wheel turns (equation (8)). However, this also removes the effect of peddling.
- Because Pearl's interventions fix RVs to a specific value, they cannot contain a probability distribution.
- For the same reason, they can also not introduce a new relation between existing RVs.

In all of these cases, the fix is to somehow already include the intervention that we wish to perform in the model. For instance, for the second case:

$$Fires(A) := Captain \vee NervousWithoutTraining \wedge \neg Training \vee NervousWithTraining \wedge Training.$$

In itself, this is not hard, but for the reasons outlined at the beginning of this section, the CP-logic way of handling such interventions without changing the original causal model is preferable.

7 Related Work

Shortcomings of PCMs have already been recognized in the literature. For instance, Hopkins and Pearl [6] join us in observing that: "In structural causal models, everything is represented as a random variable. Thus, one cannot distinguish between an enduring condition (e.g. the man is dead) versus a transitional event (e.g. the man dies)." They then attempt to fix this and other problems, by means of Situation Calculus. To us, this seems like overkill. SitCalc is an expressive action language, which contains many features that go beyond what is traditionally expressed in a causal model (e.g., preconditions for actions, fluents that might spontaneously change value, and frame axioms for prohibiting fluents from spontaneously changing values). For typical causal reasoning problems, these features are not needed: for instance, one is always free to consider

any intervention (= action) whatsoever, and the value of the fluents is fully bound by the causal laws. In any case, Hopkins' approach requires that all "intervenable" properties be represented as SitCalc actions, which means that (1) it cannot handle causal cycles such as the gear wheels any better than regular PCMs (Section 5), and (2) it is equally limited in the kind of interventions that can be considered without changing the original model (Section 6).

The semantics of CP-logic is closely related to the well-founded model construction of logic programming (LP). There are a number of other LP languages that deal with probability and causality, on which we will now briefly comment. We discuss only issues related to the specific topic of this paper (interventions and counterfactuals); for a more general comparison, we refer to [10].

P-log [1] is a language that performs probabilistic reasoning with answer sets. It comes equipped with a *do*-operator for performing interventions, which, as shown in [2], can be used to perform counterfactual reasoning in P-log. Essentially, this *do*-operator is the same as Pearl's. Therefore, we could repeat here the comments made earlier. P-log translates its causal models, together with interventions and observations, to an ASP program, which is then combined with an ASP knowledge base. Instead of relying on the *do*-operator, one can also update such a program by adding additional ASP rules. These "non-monotonic updates", as they are called, provide a much more flexible system for interventions, but they do require knowledge of how the high-level probabilistic construct are translated into the ASP encoding. A second similarity between P-log and Pearl is P-log's *coherence* criterion, which is similar to Pearl's condition that an assignment of values to the exogenous variables should uniquely determine the values of the endogenous ones. Moreover, like Pearl, Baral et al. suggest ensuring this criterion by means of an acyclicity condition. The arguments we gave earlier regarding the advantages of CP-logic when it comes to representing cyclic causal relations also carry over to P-log.

As shown in [10], Poole's Independent Choice Logic (ICL) [8] is a sublanguage of CP-logic, to which the whole logic can be mapped in a polynomial and modular way. [3] examines interventions (as well as related notions such as actual causes and explanations) in this logic. However, they arrive at their definitions by means of a transformation into Pearl's causal models. This is the opposite of our approach: they take the fine-grained, event-based representation (restricted to the acyclic case), compile it into the coarser RV-based representation, and then do the interventions there. As we have argued above, there are significant advantages to defining interventions directly on CP-laws.

8 Conclusions

This paper has presented an analysis of interventions and counterfactuals, that reformulates Pearl's intuitions in the Shaferian framework of CP-logic. Our treatment has some attractive features: we can elegantly handle cyclic causality and can deal with several kinds of interventions that, for various reasons, can only by handled by Pearl at the cost of tinkering with the original model.

Besides these practical advantages, our work also makes philosophical contributions. First, it ties together different approaches to causality from the literature: we investigate Pearl's interventions and counterfactuals in Shafer's semantic context, using the LP-based language of CP-logic to syntactically describe classes of probability trees.

Second, we also add a touch of clarity to the picture painted by Pearl. His book considers causal models in two different languages: Bayesian networks and PCMs. While they are formally very similar—every Bayesian network is easily transformed into a PCM—they embody views on the nature of causality that are ontologically quite different: Bayesian networks represent causal relations as inherently probabilistic, while PCMs express the *Laplacian* view that causal relations are completely deterministic and uncertainty stems solely from a lack of knowledge about their "inputs". Pearl's book adopts Bayesian networks throughout the chapters that first introduce the idea of interventions. When eventually the topic of counterfactuals arises, however, a switch is made to the Laplacian view of PCMs. It is peculiar that, on the one hand, interventions should be easiest to explain under the assumption that causal relations are inherently probabilistic, while on the other hand, their use for counterfactual reasoning requires the assumption that causal relations are deterministic. Our paper shows that CP-logic's event-based view on causality reconciles these two views: *whether* a Humean event happens is deterministic (in any given state of the world), but its *outcome* can be probabilistic. In this way, CP-logic can match Bayesian networks as a natural representation for probabilistic causal relations, while also, as this paper has shown, surpassing PCMs as a counterfactual reasoning tool.

References

1. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory and Practice of Logic Programming 9(1) (2009)
2. Baral, C., Hunsaker, M.: Using the probabilistic logic programming language P-log for causal and counterfactual reasoning and non-naive conditioning. In: Proceedings of IJCAI (2007)
3. Finzi, A., Lukasiewicz, T.: Structure-based causes and explanations in the independent choice logic. In: Uncertainty in Artificial Intelligence, UAI (2003)
4. Halpern, J., Pearl, J.: Causes and explanations: A structural model approach – part I: Causes. In: Uncertainty in Artificial Intelligence, UAI (2001)
5. Halpern, J., Pearl, J.: Causes and explanations: A structural-model approach. part I: Causes. The British Journal for the Philosophy of Science 56(4) (2005)
6. Hopkins, M., Pearl, J.: Causality and counterfactuals in the situation calculus. J. Log. Comput. 17(5), 939–953 (2007)
7. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge Press, Cambridge (2000)
8. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artificial Intelligence 94(1-2), 7–56 (1997)
9. Shafer, G.: The art of causal conjecture. MIT Press, Cambridge (1996)
10. Vennekens, J., Denecker, M., Bruynooghe, M.: CP-logic: A language of probabilistic causal laws and its relation to logic programming. Theory and Practice of Logic Programming 9(3), 245–308 (2009)

An Approximative Inference Method for Solving ∃∀SO Satisfiability Problems

Hanne Vlaeminck, Johan Wittocx, Joost Vennekens,
Marc Denecker, and Maurice Bruynooghe

Department of Computer Science, K.U. Leuven

Abstract. The fragment $\exists\forall SO(ID)$ of second order logic extended with inductive definitions is expressive, and many interesting problems, such as conformant planning, can be naturally expressed as *finite domain* satisfiability problems of this logic. Such satisfiability problems are computationally hard (Σ_2^P). In this paper, we develop an approximate, sound but incomplete method for solving such problems that transforms a $\exists\forall SO(ID)$ to a $\exists SO(ID)$ problem. The finite domain satisfiability problem for the latter language is in NP and can be handled by several existing solvers. We show that this provides an effective method for solving practically useful problems, such as common examples of conformant planning. We also propose a more complete translation to $\exists SO(FP)$, existential SO extended with nested inductive and coinductive definitions.

1 Introduction

Several declarative problem solving frameworks for solving search problems are based on the computational task of finite model generation. Prominent examples of such frameworks are Answer Set Programming (ASP) [1] and model expansion [10]. In ASP, finite Herbrand models of an answer set program are computed [1]. Model expansion (MX) generalizes Herbrand model generation and aims at computing one or more models of a theory T that expand a finite structure I_0 for a (possibly empty) subset of symbols of T. MX can be applied for arbitrary logics with a model theoretic semantics. In [10], it is shown that MX for first order logic (MX(FO)) is complete for NP problems ("it captures NP"). This property is preserved for rich extensions of FO, such as FO extended with inductive definitions (FO(ID)) [5] and with aggregates. By contrast, disjunctive ASP is complete for Σ_2^P [1]. Formally, $MX(FO)$ is equivalent to the finite domain satisfiability problem for existential second-order logic ($SAT(\exists SO)$). An overview of state-of-the-art ASP and MX(FO($\cdot$)) solvers is in found in [6] (here, FO($\cdot$) refers to arbitrary extensions of FO).

As a running example, consider the following dynamic domain:*a glass may be clean or not, and can be cleaned by the action of wiping.* This is expressed in the following FO theory T_{action}:

$$\begin{array}{rl} & \forall t : (Clean(t+1) \Leftrightarrow Clean(t) \vee Wipe(t)). \\ \wedge & \quad Clean(0) \Leftrightarrow InitiallyClean. \end{array} \tag{1}$$

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 326–338, 2010.
© Springer-Verlag Berlin Heidelberg 2010

The bounded planning problem to turn a dirty glass in a clean one in n steps is expressed by the satisfiability problem of the following $\exists SO$ formula in the range $[0 \dots n]$ of time points:

$$\exists Wipe, Clean, InitiallyClean : (T_{action} \wedge \neg InitiallyClean \wedge Clean(n)). \quad (2)$$

For $n > 0$, this formula is indeed satisfied in the suitable interpretation of $0, n, +/1$ and each *witness* W for its satisfiability provides a plan. E.g., wiping at time point 0 will do the job, as is verified by the witness W for which $Wipe^W = \{0\}$ and $Clean^W = \{1, \dots, n\}$.

In this paper, we are not interested in NP, but in the next level Σ_2^P of the polynomial hierarchy. A well-known such problem is finite domain satisfiability for $\exists\forall SO$: satisfaction in finite interpretations is in Σ_2^P for every $\exists\forall SO$ sentence and is Σ_2^P-hard for some such sentences [8]. The same holds for $\exists\forall SO(ID)$. An interesting Σ_2^P problem is that of *conformant planning*, which we discuss in detail in Section 6. Extending our example, suppose that we do not know whether the object is initially clean or dirty, but still want a plan that is guaranteed to make it clean, *no matter what* the initial situation was. This can be formulated as:

$$\exists\, Wipe \forall InitiallyClean, Clean : \quad (T_{action} \Rightarrow Clean(n)). \quad (3)$$

In words, we need an assignment to the action $Wipe$ such that the goal is satisfied for every initial situation $InitiallyClean$ and fluent $Clean$ that satisfy the action theory. Note that instead of a conjunction as in (2), in formula (3) we find an implication. Indeed, the condition $T_{action} \wedge Clean(n)$ does not solve the problem as there are many interpretations for the $Clean$ predicate that do not satisfy the action theory, e.g., when $InitiallyClean$ is true and $Clean(0)$ is false.

While Σ_2^P problems can be solved in principle, e.g., by solvers for disjunctive ASP, in practice they are often too hard. In this paper, we present an *approximate* method that consists of reducing a $\exists\forall SO(ID)$ problem to a $\exists SO(ID)$ problem. This method is sound but not complete, in the sense that a witness of the approximating $\exists SO(ID)$ formula is a witness of the $\exists\forall SO$ formula, but not necessarily the other way around. Our method exploits the techniques for *constraint propagation* in FO$(\cdot)$ proposed in [17,16]. This propagation operates on a three-valued structure that approximates all models of an FO theory and makes it more and more precise. It was shown in [16] that the propagation process can be captured in a formal FO(ID) inductive definition, and we use it here to build the $\exists SO(ID)$ formula. This approach has the advantage that the translation can be automated and that any existing satisfiability solver for $\exists SO(ID)$ or any MX(FO(ID)) solver can be plugged in. Finally, we exploit a result of [11] to develop a more accurate translation of $\exists\forall SO(ID)$ formulas with inductive definitions to $\exists SO(FP)$, $\exists SO$ with nested least and greatest fixpoint definitions. Our method is inspired by interpolation in Logic Programming [2,12] and approximate query answering in locally closed databases [4].

2 Preliminaries

We assume familiarity with standard first order logic (FO) and second order logic (SO). We define the extensions FO(ID) and FO(FP) of FO, and the corresponding extensions of SO as follows. Given is a vocabulary Σ. A rule (over Σ) is an expression of the form $\forall \bar{x}\ P(\bar{t}) \leftarrow \varphi$ where $P(\bar{t})$ is an atomic formula and φ an FO formula. The symbol $\leftarrow$ is a new connective, called the *definitional implication*, to be distinguished from the FO material implication symbol $\Leftarrow$ (or its more standard inverse $\Rightarrow$). A definition Δ is a finite set of rules. A predicate symbol P in the head of a rule of Δ is called a *defined predicate*; all other predicate and function symbols in Δ are called *open symbols* or the *parameters* of the definition; the set of defined predicates is denoted $Def(\Delta)$, the remaining symbols $Open(\Delta)$. An FO(ID) formula is defined using the standard induction defining an FO formula, augmented with one extra case:

– A definition Δ over Σ is an FO(ID) formula (over Σ)).

FO(ID) formulas are quantified boolean combinations of atoms and definitions. Notice that rule bodies do not contain definitions, that rules only occur inside definitions and are not FO(ID) formulas themselves. The satisfaction relation $I \models \varphi$ of FO(ID) is defined using the standard inductive rules of FO, augmented with one extra rule:

– $I \models \Delta$ if $I = (I|_{Open(\Delta)})^{\Delta}$.

where $(I|_{Open(\Delta)})^{\Delta}$ is the well-founded model of Δ extending the restriction of I to the open symbols of Δ. Here we use the parameterized version of the well-founded semantics that was introduced in the context of deductive databases [15]; it defines *intensional view predicates* (i.e., defined predicates) in terms of a database of *extensional predicates* (i.e., a structure defining open symbols).

The formal notion of a definition as defined here is a faithful syntactic formalisation of informal inductive definitions as used in mathematics [3].

We now define the logic FO(FP). A rule is called *positive* in a set of predicate symbols σ if each predicate of σ has only positive occurrences in rule bodies (i.e., is in the scope of an even number of $\neg$). A *fixpoint definition* is defined inductively as either a *least fixpoint definition* $\lfloor S, \Delta_1, \ldots, \Delta_n \rfloor$ or a *greatest fixpoint definition* $\lceil S, \Delta_1, \ldots, \Delta_n \rceil$, where in both cases S is a set of rules, and $\Delta_1, \ldots, \Delta_n$ are fixpoint definitions. Define the defined predicates $Def(\Delta)$ of a fixpoint definition Δ inductively, as $Def(S) \cup Def(\Delta_1) \cup \cdots \cup Def(\Delta_n)$. We require that predicates of $Def(\Delta)$ have only positive occurrences in rule bodies anywhere in Δ, and also that defined predicates of Δ_i do not occur in Δ_j, $i \neq j$. An FO(FP) formula is defined as in FO with one extra case:

– A fixpoint definition $\mathcal{D}$ over Σ is an FO(FP) formula (over Σ).

With each fixpoint definition $\mathcal{D}$, a monotonic operator $\Gamma_{\mathcal{D}}$ can be associated. This operator is essentially an extension of the standard operator of (unnested) inductive definitions defined by induction on the subdefinition structure of $\mathcal{D}$. We then define:

- $I \models \Delta$ (where Δ is a least fixpoint definition) if I is the least fixpoint of Γ_Δ.
- $I \models \nabla$ (where ∇ is a greatest fixpoint definition) if I is the greatest fixpoint of Γ_∇.

This notion of fixpoint definition is a syntactic variant of the notion of nested least and greatest fixpoint expressions, the difference being that predicate symbols are defined instead of fixpoint expressions denoting relations, and a rule-based syntax is used.

The techniques introduced in the following sections implicitly use concepts of three-valued logic. We assume familiarity with three-valued interpretations and (Kleene's) three-valued truth evaluation. Three-valued interpretations $\mathcal{I}$ assign three-valued relationships to predicate symbols and are used here as approximations of two-valued interpretations I. The precision order $\mathcal{I} \leq_p \mathcal{I}'$ holds if $\mathcal{I}$ and $\mathcal{I}'$ share domain and interpretation of function symbols and $P^{\mathcal{I}}(d_1, \ldots, d_n) = P^{\mathcal{I}'}(d_1, \ldots, d_n)$, for each tuple $(d_1, \ldots, d_n)$ and predicate P/n such that $P^{\mathcal{I}}(d_1, \ldots, d_n) \neq \mathbf{u}$. Two-valued interpretations are maximally precise three-valued interpretations.

We will use a well-known technique to encode three-valued interpretations by two-valued interpretations of an extended language. In particular, let σ be a subvocabulary of Σ such that $\Sigma \setminus \sigma$ contains only predicate symbols. Each three-valued Σ-interpretation $\mathcal{I}$ that is two-valued in every symbol of σ can be encoded as a two-valued I^{tf} of the vocabulary $\Sigma^{tf} = (\sigma \cup \{Q^{ct}, Q^{cf} | Q \in \Sigma \setminus \sigma\}$ such that $\mathcal{I}|_\sigma = I^{tf}|_\sigma$ and such that $(Q^{ct})^{I^{tf}} = \{(d_1, \ldots, d_n) | Q^{\mathcal{I}}(d_1, \ldots, d_n) = \mathbf{t}\}$ and $(Q^{cf})^{I^{tf}} = \{(d_1, \ldots, d_n) | Q^{\mathcal{I}}(d_1, \ldots, d_n) = \mathbf{f}\}$. For a formula φ in negation normal form, we denote by $\varphi^{\mathrm{ct}}_\sigma$ the result of replacing atoms $P(\bar{t})$ by $P^{ct}(\bar{t})$ and negative literals $\neg P(\bar{t})$ by $P^{cf}(\bar{t})$, for every $P \in \Sigma \setminus \sigma$. This encoding has the property that $\varphi^I = \mathbf{t}$ iff $(\varphi^{\mathrm{ct}}_\sigma)^{I^{tf}} = \mathbf{t}$.

3 Propagation for FO

Suppose we have a finite three-valued structure $\mathcal{I}$ that represents some (incomplete) knowledge about the symbols appearing in an FO theory T. We would now like to know the implications of this knowledge. To find this out, we look at the set $\mathcal{M}$ of all models of T that complete this three-valued structure, i.e., $\mathcal{M} = \{M \mid M \models T \text{ and } \mathcal{I} \leq_p M\}$. Given the partial information $\mathcal{I}$, everything that is true in all $M \in \mathcal{M}$ must certainly be true according to T, while everything that is false in all such M must certainly be false according to T. In other words, we can derive from $\mathcal{I}$ the more precise three-valued structure $\mathcal{G}$ that is the greatest lower bound $\mathrm{glb}_{\leq_p}\mathcal{M}$. For instance, let T_{action} be as in (1) and $\mathcal{I}$ the three-valued interpretation that knows that $InitiallyClean$ is $\mathbf{f}$ and $Clean(1)$ is $\mathbf{t}$. In every model of T_{action} that extends this $\mathcal{I}$, it is the case that $Wipe(0)$ holds, and we can figure this out by computing $\mathcal{G}$.

In general, this computation may be too expensive (Δ_2^P) to be of practical use. However, we may still achieve useful results by computing some approximation $\tilde{\mathcal{M}}$ such that $\mathcal{I} \leq_p \tilde{\mathcal{M}} \leq_p \mathcal{G}$. For instance, consider again example (1).

If we know that $\neg InitiallyClean$, we can derive from the second conjunct that also $\neg Clean(0)$, which, according to the first conjunct, implies in turn that the only way to achieve $Clean(1)$ is by $Wipe(0)$. Using this idea, [17] developed a polynomial propagation method to compute such a $\tilde{\mathcal{M}}$. This method was implemented as a C++ program, and is now used in, among others, the grounder of the FO(ID) finite model generator IDP [9].

A recent result in [16] that will prove key to our enterprise here, is that the propagation process can be captured *symbolically* by an FO(ID) definition that defines $\tilde{\mathcal{M}}$. This compilation of an FO theory T into an inductive definition consists of three steps. First, T is rewritten to a theory T' containing only sentences of the form $\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow \varphi)$. Second, these equivalences are split into several sentences of the form $\forall \bar{x}\ (\psi \Rightarrow L[\bar{x}])$, where L is a literal. Finally, these implications are rewritten to rules of an inductive definition.

Definition 1. *An FO sentence φ is in* equivalence normal form *(ENF) if it is of the form $\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow \psi[\bar{x}])$, and ψ is of the form L, $(L_1 \wedge L_2)$, $(L_1 \vee L_2)$, $(\forall v\ L)$ or $(\exists v\ L)$, where L, L_1 and L_2 are literals.*

For the first step, we assume without loss of generality that T is in negation normal form and contains only one formula. The theory T' in ENF is easily obtained from T through a process akin to the Tseitin transformation for propositional logic [14]. Consider the parse-tree of T. For each node $\psi[\bar{x}]$ in this tree that is not a literal, we introduce a new symbol A_ψ/n and add the equivalence $\forall \bar{x}(A_\psi(\bar{x}) \Leftrightarrow \psi')$ where ψ' is obtained from ψ by substituting the Tseitin predicates $A_\phi(\bar{y})$ for non-literal immediate subformulas $\phi[\bar{y}]$ of $\psi[\bar{y}]$.

Proposition 1. *The ENF theory T' is linear in the size of T. Also, M is a model of T iff there exists an expansion M' of M to the vocabulary of T' such that $M' \models T'$ and $M' \models A_T$ where A_T is the Tseitin predicate for T.*

In the second step, we rewrite each ENF formula $\varphi \in T'$ to an equivalent set $T^{\Rightarrow}$ of implications $\mathrm{INF}(\varphi)$ of the form $\forall \bar{x}\ (\psi \Rightarrow L[\bar{x}])$, where L is a literal. The idea is that these implications exhaustively enumerate all the inferences that one could make on the basis of the ENF formula. For instance, if $\varphi = \phi \wedge \psi$, then A_φ implies both φ and ψ, $\varphi \wedge \psi$ implies A_φ, $\neg\varphi$ and $\neg\psi$ both imply $\neg A_\varphi$, $\neg A_\varphi \wedge \varphi$ implies $\neg\psi$, and similarly $\neg A_\varphi \wedge \psi$ implies $\neg\varphi$. Table 1 specifies the corresponding implications for all types of ENF formulas.

The theory $T^{\Rightarrow}$ allows us to characterize the algorithm of [17] in a convenient way: whenever it has inferred the antecedent of such an implication, it infers the consequent. This is reminiscent of Stickel's encoding of clauses in his Prolog Technology Theorem Prover [13]. In the third step, we encode this propagation process explicitly in an inductive definition. Given a theory T over Σ and let $\sigma \subseteq \Sigma$ be a set of symbols on which we have full knowledge in the form of a σ-interpretation I. Assume also that $\bar{Q} = \Sigma \setminus \sigma$ consists of predicate symbols only. We can now use the propagation rules in $T^{\Rightarrow}$ to expand I with three-valued interpretations for the predicates $P \in \bar{Q}$. Recall that three-valued interpretations can be encoded using predicates P^{ct} and P^{cf}. The propagation process in these

Table 1. The implications INF(φ) for an ENF formula φ

φ	INF(φ)
$\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow L[\bar{x}])$	$\forall \bar{x}\ (P(\bar{x}) \Rightarrow L[\bar{x}])$ $\forall \bar{x}\ (\neg L[\bar{x}] \Rightarrow \neg P(\bar{x}))$ $\forall \bar{x}\ (L[\bar{x}] \Rightarrow P(\bar{x}))$ $\forall \bar{x}\ (\neg P(\bar{x}) \Rightarrow \neg L[\bar{x}])$
$\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow \forall y\ L[\bar{x}, y])$	$\forall \bar{x} \forall y\ (P(\bar{x}) \Rightarrow L[\bar{x}, y])$ $\forall \bar{x}\ ((\exists y\ \neg L[\bar{x}, y]) \Rightarrow \neg P(\bar{x}))$ $\forall \bar{x}\ ((\forall y\ L[\bar{x}, y]) \Rightarrow P(\bar{x}))$ $\forall \bar{x} \forall y\ (\neg P(\bar{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow L[\bar{x}, y'])) \Rightarrow \neg L[\bar{x}, y])$
$\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow \exists y\ L[\bar{x}, y])$	$\forall \bar{x} \forall y\ (P(\bar{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow \neg L[\bar{x}, y'])) \Rightarrow L[\bar{x}, y])$ $\forall \bar{x}\ ((\forall y\ \neg L[\bar{x}, y]) \Rightarrow \neg P(\bar{x}))$ $\forall \bar{x}\ ((\exists y\ L[\bar{x}, y]) \Rightarrow P(\bar{x}))$ $\forall \bar{x} \forall y\ (\neg P(\bar{x}) \Rightarrow \neg L[\bar{x}, y])$
$\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (P(\bar{x}, \bar{y}, \bar{z})$ $\Leftrightarrow L_1[\bar{x}, \bar{y}] \wedge L_2[\bar{x}, \bar{z}])$	$\forall \bar{x} \forall \bar{y}((\exists \bar{z}\ P(\bar{x}, \bar{y}, \bar{z})) \Rightarrow L_1[\bar{x}, \bar{y}])$ $\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (\neg L_1[\bar{x}, \bar{y}] \Rightarrow \neg P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{z}((\exists \bar{y}\ P(\bar{x}, \bar{y}, \bar{z})) \Rightarrow L_2[\bar{x}, \bar{z}])$ $\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (\neg L_2[\bar{x}, \bar{z}] \Rightarrow \neg P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (L_1[\bar{x}, \bar{y}] \wedge L_2[\bar{x}, \bar{z}] \Rightarrow P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{y}((\exists \bar{z}\ (\neg P(\bar{x}, \bar{y}, \bar{z}) \wedge L_2[\bar{x}, \bar{z}])) \Rightarrow \neg L_1[\bar{x}, \bar{y}])$ $\forall \bar{x} \forall \bar{z}((\exists \bar{y}\ (\neg P(\bar{x}, \bar{y}, \bar{z}) \wedge L_1[\bar{x}, \bar{y}])) \Rightarrow \neg L_2[\bar{x}, \bar{z}])$
$\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (P(\bar{x}, \bar{y}, \bar{z})$ $\Leftrightarrow L_1[\bar{x}, \bar{y}] \vee L_2[\bar{x}, \bar{z}])$	$\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (\neg L_1[\bar{x}, \bar{y}] \wedge \neg L_2[\bar{x}, \bar{z}] \Rightarrow \neg P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{y}((\exists \bar{z}\ (P(\bar{x}, \bar{y}, \bar{z}) \wedge \neg L_2[\bar{x}, \bar{z}])) \Rightarrow L_1[\bar{x}, \bar{y}])$ $\forall \bar{x} \forall \bar{z}((\exists \bar{y}\ (P(\bar{x}, \bar{y}, \bar{z}) \wedge \neg L_1[\bar{x}, \bar{y}])) \Rightarrow L_2[\bar{x}, \bar{z}])$ $\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (L_1[\bar{x}, \bar{y}] \Rightarrow P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{y}((\exists \bar{z}\ \neg P(\bar{x}, \bar{y}, \bar{z})) \Rightarrow \neg L_1[\bar{x}, \bar{y}])$ $\forall \bar{x} \forall \bar{y} \forall \bar{z}\ (L_2[\bar{x}, \bar{z}] \Rightarrow P(\bar{x}, \bar{y}, \bar{z}))$ $\forall \bar{x} \forall \bar{z}((\exists \bar{y}\ \neg P(\bar{x}, \bar{y}, \bar{z})) \Rightarrow \neg L_2[\bar{x}, \bar{z}])$

predicates is described by an inductive definition of predicates P^{ct} and P^{cf}, for every P appearing in $T^{\Rightarrow}$ but not in σ, i.e., for every symbol of $\bar{Q}$ and also for every Tseitin predicate A_φ.

Definition 2. *For a theory T, set of predicates $\bar{Q}$ such that $\sigma = \Sigma \backslash \bar{Q}$, we define $Approx_\sigma(T)$ as the inductive definition that contains, for every sentence $\forall \bar{x}\ (\psi \Rightarrow L[\bar{x}])$ of $T^{\Rightarrow}$ in which L is a literal of a predicate not in σ, the definitional rule $\forall \bar{x}(L[\bar{x}]^{\mathrm{ct}}_\sigma \leftarrow \psi^{\mathrm{ct}}_\sigma)$.*

Example 1. For the cleaning example, consider the formula $T := T_{action} \Rightarrow Clean(n)$. Take σ to be $\{Wipe, +/1, 0, n\}$. Then $Approx_\sigma(T)$ defines $InitiallyClean^{ct}$, $InitiallyClean^{cf}$, $Clean^{ct}$ and $Clean^{cf}$ as well as A^{ct}_φ and A^{cf} for any introduced Tseitin predicate A_φ, in particular for the Tseitin predicate A_T that is equivalent to the formula $T_{action} \Rightarrow Clean(0)$ and $A_{T_{action}}$ that is equivalent to the conjunction of formulas in the cleaning theory (1). $Approx_\sigma(T)$ then contains amongst others the following definitional rules:

$$\left\{ \begin{array}{ll} A^{ct}_T & \leftarrow A^{cf}_{T_{action}} \vee Clean^{ct}(n). \\ A^{cf}_T & \leftarrow A^{ct}_{T_{action}} \wedge Clean^{cf}(n). \\ A^{ct}_{T_{action}} & \leftarrow A^{ct}_{\varphi_1} \wedge A^{ct}_{\varphi_2} \\ Clean^{ct}(t) & \leftarrow \ldots \\ & \ldots \end{array} \right\},$$

where φ_1 and φ_2 are the two formulas of (1). $Wipe$ is the only open predicate of this definition. In a given σ-interpretation, the definition will compute what

is certainly true and what is certainly false. E.g., when $Wipe(t)$ is false for all t, it will compute that both $Clean^{ct}(t)$ and $Clean^{cf}(t)$ are false for all t.

Proposition 2. *Given T, $\bar{Q}$, $\sigma = \Sigma \setminus \bar{Q}$, a σ-interpretation I, $P \in \bar{Q}$ and a tuple of domain elements $\bar{d}$, the algorithm in [17] will derive the literal $P(\bar{d})$ (respectively $\neg P(\bar{d})$) exactly when $P^{ct}(\bar{d})$ (respectively $P^{cf}(\bar{d})$) holds in the unique model M of the definition $Approx_\sigma(T) \cup \{T^{ct} \leftarrow\}$ expanding I.*

From the correctness of this algorithm, it therefore follows that if $P^{ct}(\bar{d})$ (or $P^{cf}(\bar{d})$) holds in this model M, then $P(\bar{d})$ holds (does not hold) in all models of T that extend I.

4 Approximating $\exists\forall SO$-Satisfiability Problems

Consider the problem whether an $\exists\forall SO$ formula $\exists\bar{P}\forall\bar{Q} : T$ is satisfied in some finite interpretation I of the non-variable symbols of the formula. We call a *witness* for its satisfiability an expansion of I to all existentially quantified variables $\bar{P}$ that satisfies $\forall\bar{Q} : T$. We assume that $\bar{Q}$ consists only of predicate variables (while $\bar{P}$ might include also function variables). We now use the transformation of the previous section to this formula into a stronger $\exists SO(ID)$ formula, i.e., one with less witnesses for $\bar{P}$. Take σ to be the set of all symbols in T except those in $\bar{Q}$.

Definition 3. *For a formula $F = \exists\bar{P}\forall\bar{Q} : T$, we define $\mathcal{APP}(F)$ as the $\exists SO$ formula $\exists\bar{P}\exists\bar{R} : Approx_\sigma(T) \wedge A_T^{ct}$, where $\bar{R} = \{X^{ct}, X^{cf} | X \notin \sigma\}$ (i.e., X a Tseitin symbol A_ϕ or a $Q \in \bar{Q}$) and A_T is the Tseitin symbol for T.*

The intuition here is that for any σ-interpretation I, $Approx_\sigma(T)$ will tell us what the consequences of this choice are, regardless of the value of the universal predicates $\bar{Q}$. If one of these consequences is that the entire FO formula T is true, then we therefore know that I is a witness for the satisfiability of the entire formula F.

Proposition 3. *For each $\exists\forall SO$ formula F of the form $\exists\bar{P}\forall\bar{Q} : T$, $\mathcal{APP}(F)$ is a sound approximation of F, i.e. if $\mathcal{APP}(F)$ is satisfied in interpretation I, then F is satisfied too. Moreover, if I is a witness for the satisfiability of $\mathcal{APP}(F)$, then $I|_\sigma$ is a witness for the satisfiability of F.*

For example, this is the translation of the formula $F = \exists P\forall Q : P \vee Q$:

$$\exists P, Q^{ct}, Q^{cf} : \left\{ \begin{array}{l} T^{ct} \leftarrow P \vee Q^{ct} \\ T^{cf} \leftarrow \neg P \wedge Q^{cf} \\ Q^{ct} \leftarrow T^{ct} \wedge \neg P \\ Q^{cf} \leftarrow T^{cf} \end{array} \right\} \wedge T^{ct}.$$

If we choose P to be true, then the definition forces T^{ct} to be true and T^{cf} to be false. Hence, neither Q^{ct} nor Q^{cf} become true. In other words, choosing P true

implies nothing about Q, but it makes the disjunction $T = P \vee Q$ true for each possible value for Q. This choice for P is therefore a witness for the satisfiability of $\mathcal{APP}(F)$, and it is indeed also a witness for the satisfiability of the original formula $\exists P \forall Q : P \vee Q$.

This approximation method is sound, but for many applications still too incomplete. In particular, it will rarely manage to detect satisfiability of formulas of the form $\exists \bar{P} \forall \bar{Q} : (T_1 \Rightarrow T_2)$. This is because it can only derive that an implication $T_1 \Rightarrow T_2$ holds for all $\bar{Q}$ by either deriving that T_1 is certainly false (i.e., false for all $\bar{Q}$) or that T_2 is certainly true (i.e., true for all $\bar{Q}$). For conformant planning problems (which are of this form, as we will see further), this will never be the case. We will illustrate this with our running example. If we have a look at the definition in example 1, we see that the only way to make A_T^{ct} true is if $A^{cf}_{T_{action}} \vee Clean^{ct}(n)$ is true. However, for a certain choice of Wipe, there are always interpretations for the fluents (e.g. Clean) for which the action theory is satisfied, but also for which it is not satisfied (namely, one of many in which the fluents are simply incorrect for the actions). On the other hand it is also clear that not in all interpretations of the fluents $Clean(n)$ holds. Thus, we will never be able to derive that A_T^{ct} is true. However, we can make our method more complete for problems of the form $\exists \bar{P} \forall \bar{Q} : (T_1 \Rightarrow T_2)$ by postulating the truth of T_1 while checking T_2, as follows.

Definition 4. *For an an $\exists\forall SO$ formula F of the form $\exists \bar{P} \forall \bar{Q} : T_1 \Rightarrow T_2$, we define $\mathcal{APP}^{\Rightarrow}(F)$ as $\exists \bar{P} \exists \bar{R} : \Delta \wedge T_2^{ct})$, where $\Delta = Approx_\sigma(T_1 \Rightarrow T_2) \cup \{T_1^{ct} \leftarrow\}$.*

Note that we add T_1^{ct} as a definitional rule, and T_2^{cf} as a constraint. If we take T_1 to be the trivial formula $\mathbf{t}$, we get back Def. 3 as a special case of this definition. This approximation method is still sound, as the following proposition states.

Proposition 4. *Given a formula F of the form $\exists \bar{P} \forall \bar{Q} : T_1 \Rightarrow T_2$, the $\exists SO(ID)$ formula $\mathcal{APP}^{\Rightarrow}(F)$ is a sound approximation of F, i.e. if $\mathcal{APP}^{\Rightarrow}(F)$ is satisfiable, then F is satisfiable too. Moreover, if I is a witness of the satisfiability of $\mathcal{APP}^{\Rightarrow}(F)$, then $I|_\sigma$ is a witness for the satisfiability of F.*

5 Approximating Definitions

In this section, we extend our approximation method to formulas including definitions. We will not consider the general case where definitions may appear at arbitrary locations in a formula, but instead restrict attention to formulas of the form $\exists \bar{P} \forall \bar{Q} : \Delta_1 \wedge \cdots \wedge \Delta_n \wedge \phi \Rightarrow T_2$, where the Δ_i are definitions such that $Def(\Delta_i) \subseteq \bar{Q}$ and ϕ and T_2 are FO formulas. This covers the way in which definitions are typically used: under the assumption that all predicates indeed are what the definitions Δ_i (and the formula ϕ) say they should be, T_2 then states what properties they should satisfy. For instance, in conformant planning, the action theory could include a definition of the fluents in terms of the actions that are performed. By restriction attention to formulas of this form, we avoid the

need for approximation rules that infer that a definition as a whole is certainly true/false.

A first approach is based on the fact that a model of a definition Δ is also a model of the FO completion $compl(\Delta)$. Let us assume w.l.o.g. that each defined predicate P of Δ_i is defined by a single definitional rule $\forall \bar{x}\ P(\bar{x}) \leftarrow \phi$. Then $compl(\Delta)$ consists of all formulas $\forall \bar{x}\ P(\bar{x}) \Leftrightarrow \phi$, for each $P \in Def(\Delta)$. Since definitions occur only negatively in the $\exists \forall SO(ID)$ formula (i.e. in the body of the implication), it is sound to replace each Δ_i by the weaker theory $compl(\Delta_i)$. That is, each witness to $\exists \bar{P} \forall \bar{Q} : compl(\Delta_1) \wedge \cdots \wedge compl(\Delta_n) \wedge \phi \Rightarrow T_2$ is a witness to $\exists \bar{P} \forall \bar{Q} : \Delta_1 \wedge \cdots \wedge \Delta_n \wedge \phi \Rightarrow T_2$. The first formula is $\exists \forall SO$ and we can apply the technique of the previous section.

This method is sound and works fine for non-recursive definitions (where completion is equivalent with FO(ID) semantics) but in case of recursive(=inductive) definitions, it might result in unacceptable loss of precision. For illustration, consider the inductive definition $\{P \leftarrow P\}$. It entails $\neg P$ but this conclusion cannot be derived from its completion. Indeed, the rules in $Approx(P \Leftrightarrow P)$ will obviously fail to derive P^{cf}. As a consequence, the satisfiability of the formula $\forall P : \{P \leftarrow P\} \Rightarrow \neg P$ could not be detected using the above method (since $\forall P : (P \Leftrightarrow P) \Rightarrow \neg P$ is not satisfiable).

As a more complete method, we propose the following. As explained in the preliminaries, for each three-valued interpretation $\mathcal{I}$ of $Open(\Delta)$, the definition Δ has a (three-valued) well-founded model $\mathcal{W}$ extending $\mathcal{I}$. This $\mathcal{W}$ has the interesting property that $\mathcal{W} \leq_p M$, for every model M of Δ such that $\mathcal{I} \leq_p M|_{Open(\Delta)}$. Our aim is now to use this well-founded model $\mathcal{W}$ to make the additional propagations.

In [11], it was shown how the computation of the well-founded model extending a two-valued $Open(\Delta)$-interpretation I can be encoded by a nested fixpoint expression in FO(FP). The following definition extends this to the case of three-valued $Open(\Delta)$-interpretations $\mathcal{I}$ that we need in this context. Let $\sigma \subseteq Open(\Delta)$ such that σ contains all function symbols in Δ. Assume that $\mathcal{I}$ is three-valued only on symbols of $Open(\Delta) \setminus \sigma$.

Definition 5. *For a definition Δ, we define $FP_\sigma(\Delta)$ as $\lfloor \mathcal{R}^{ct}, \lceil \mathcal{R}^{cf} \rceil \rfloor$ where $\mathcal{R}^{ct}$ consists of, the rules*

$$\forall \bar{x}(P^{ct}(\bar{x}) \leftarrow \varphi^{ct}_\sigma)$$

and $\mathcal{R}^{cf}$ consists of the rules

$$\forall \bar{x}(P^{cf}(\bar{x}) \leftarrow (\neg\varphi)^{ct}_\sigma)$$

for every definitional rule $\forall \bar{x}\ P(\bar{x}) \leftarrow \varphi \in \Delta$.

This FO(FP) expression $FP_\sigma(\Delta)$ now does precisely what we want.

Proposition 5. *Given Δ, σ and $\mathcal{I}$ as specified above, let I' be the encoding of $\mathcal{I}$ in terms of the symbols P^{ct}, P^{cf}. Then the unique model of $FP_\sigma(\Delta)$ extending I' encodes the well-founded model of Δ extending $\mathcal{I}$.*

In the case of the definition $\{P \leftarrow P\}$, $FP(\Delta)$ is the following definition: $\lfloor P^{ct} \leftarrow P^{ct}, \lceil P^{cf} \leftarrow P^{cf} \rceil \rfloor$. This definition has a unique model where $P^{ct} = \mathbf{f}$ and $P^{cf} = \mathbf{t}$. This correctly encodes the well-founded model of the original definition, and we see that $FP(\Delta)$ indeed lets us infer that P has to be false.

On the one hand we can now approximate definitions by its completion. Even though we already argued that the approximation of the completion on it's own is not strong enough in the case of recursive definitions, it still does useful propagation. E.g. it allows to propagate information from the defined predicates back to the open predicates. On the other hand we have defined an encoding of the well-founded model of a definition that allows us to minimize predicates. Both of these ways to approximate definitions thus have their own use and we would like to put them together. As defined in the previous section, each approximating definition $Approx_\sigma(F)$ is a positive definiton. This means that we can equally see them as a least fixpoint definition of FO(FP). The following definition then shows how we can put the approximation of the completion of a definition Δ and the encoding of it's well-founded model together.

Definition 6. *The FO(FP) approximation $Approx^{FP}_\sigma(\Delta)$ of an FO(ID) definition Δ is the nested least fixpoint definition $\lfloor Approx_\sigma(Compl(\Delta)) \cup \mathcal{R}', FP_\sigma(\Delta') \rfloor$, where*

- *$Approx_\sigma(Compl(\Delta))$ is the rule set as defined in the previous section,*
- *Δ' is obtained from Δ by replacing all defined predicates P of $Def(\Delta)$ by new symbols P'.*
- *$\mathcal{R}'$ are the rules $P^{ct} \leftarrow P'^{ct}$ and $P^{cf} \leftarrow P'^{cf}$ for every defined predicate P of Δ.*

Note that $FP_\sigma(\Delta')$ is nested in $Approx^{FP}_\sigma(\Delta)$ and is itself a nested definition.

Finally, we now put everything together into an approximation for $\exists\forall SO(ID)$.

Definition 7. *Let F be an $\exists\forall SO(ID)$ formula of the form $\exists\bar{P}\forall\bar{Q} : \Delta_1 \wedge \cdots \wedge \Delta_n \wedge \phi \Rightarrow T_2$, where the Δ_i are definitions and ϕ and T_2 are FO formulas. We then define $\mathcal{APP}^{\Rightarrow}(F)$ as the $\exists SO(FP)$ formula*

$$\exists\bar{P}\,\bar{Q} : \lfloor Approx_\sigma(\phi) \cup Approx_\sigma(T_2) \cup \{A^{ct}_\phi \leftarrow\} \cup \bigcup_{i=1}^{n} Approx^{FP}_\sigma(\Delta_i) \rfloor \wedge T^{ct}_2.$$

Proposition 6. *Given an interpretation I interpreting the non-variable symbols of F. The above defined approximation is sound, i.e. if $\mathcal{APP}^{\Rightarrow}(F)$ is satisfied in I, then F is satisfied in I and moreover, the restriction to σ of a witness of $\mathcal{APP}^{\Rightarrow}(F)$ is a witness of F.*

6 Applications and Related Work

In the literature, many examples can be found of algorithms that perform some kind of approximate reasoning about the models of a logical theory. Typically, these algorithms, which are specific to the problem at hand, seem to boil down to an instantiation of the general methods presented here. We give some examples.

Conformant Planning. In general, a *conformant planning problem* is a planning problem in a non-deterministic domain, where, e.g., the initial state is not fully known. The goal is to come up with a plan that is nevertheless guaranteed to work. This is a hard problem (determining whether there exists a conformant plan of length $\leq k$ is Σ_2^P complete, even if k is assumed to be polynomial in the size of the problem). Therefore, one typically attempts to solve it approximately. [12] starts from a description of the planning problem in the action language $\mathcal{AL}$ and then derives from this an Answer Set Prolog program that searches for solutions in an approximated version of the corresponding transition diagram.

Our method can solve such problems in the following way. Let T_{action} be a theory that defines the fluent predicates $\bar{F}$ in terms of action predicates $\bar{A}$ and initial state predicates $\bar{I_F}$ in the context of a linear time line (possibly a finite interval). This could be an FO theory or an inductive definition as in [5]. Let T_{prec} be a theory describing preconditions $\forall \bar{x} \forall t : A(\bar{x}, t) \Rightarrow \Psi_A[\bar{x}, t]$ of each action predicates A. Finally, let the formula G specify the goal that must be achieved. The problem of conformant planning is then to decide the satisfiability of the following formula:

$$\exists \bar{A} \forall \bar{I_F} \forall \bar{F} : T_{action} \Rightarrow T_{prec} \wedge G$$

In words, there must be a plan ($\exists \bar{A}$), such that no matter how the nondeterministic aspects turn out ($\forall \bar{I}, \bar{F}$), as long as the specification of the effects of the actions is obeyed (T_{action}), the plan will be executable (T_{prec}) and achieve the goal (G). Applying the method of this paper to this formula yields a incomplete algorithm: it may not find all solutions, but if it finds one then that solution is correct. Even though more experiments are needed, preliminary results indicate that this algoritm is comparable to that of [12], both in completeness and runtime.

Querying and reasoning in open databases. Approximate methods similar to ours have been used in the context of open databases, databases with CWA [2,7]. In [4], query answering is considered in the context of databases that are as a whole incomplete, but that nevertheless contain partial, local forms of closed world assumption. The goal is to compute certain answers to queries. Because this task has a high complexity (Δ_2^P), approximate methods are presented which translate an FO query into an approximate FO or FO(FP) query that can be solved directly against the database tables using standard (polynomial) query methods. It is shown that these methods often provide optimal solutions.

The method presented in this paper can provide a similar functionality. Let DB be a set of ground literals, representing an incomplete database. Let Ψ be a background theory: it may contain integrity constraints, view definitions (datalog view programs are FO(ID) definitions), local closed world statements in FO, etc. For a given FO query $Q[\bar{x}]$, the goal is to compute all terms $\bar{t}$ such that $Q[\bar{t}]$ holds in all Herbrand models of $DB \cup \Psi$. The problem of deciding whether a tuple $\bar{t}$ is an answer is the satisfiability problem of $\forall \bar{R}(DB \wedge \Psi \Rightarrow Q[\bar{t}])$, and to this problem our approximate method applies.

While this allows us to decide whether a tuple $\bar{t}$ is a certain answer to the query, it does not yet provide a reasonable method to compute (an approximation of) all such tuples. This can be done as follows. Consider the definition $Approx(DB \wedge \Psi) \cup Approx(Q[\bar{x}])$, consisting of rules describing propagations allowed by the database and rules defining the predicate symbol A_Q^{ct} (A_Q being the Tseitin predicate representing the query $Q[\bar{x}]$). In the unique Herbrand model of this definition, the interpretation of this A_Q^{ct} contains those tuples for which our propagation can derive that they certainly satisfy the query—a sound approximation of the full set of answers. Standard deductive database techniques or techniques from fixpoint logics can be used to compute this relation in polynomial time.

7 Conclusions and Future Work

Even if a problem is computationally hard in general, specific instances of it might still be solved efficiently. This is why approximate methods are important: they cannot solve every instance, but the instances they can solve, they solve quickly. In computational logic, hard problem arise quite readily. It is therefore not surprising that the literature contains numerous examples of algorithms that perform approximate reasoning tasks for various logical formalisms in various specific contexts. Since many of these algorithms share common ideas, it is a natural question whether they can be seen as instances of some more general method for a more general language.

This paper tries to present such a method. We start from the propagation method for FO($\cdot$) developed in [17] and its symbolic expression in [16] and generalize this to a method for approximating the Σ_2^P-complete $\exists\forall SO(ID)$ satisfiability problem by solving an NP problem. Importantly, this is a syntactic method that transforms the $\exists\forall SO(ID)$ formula into either a $\exists SO(ID)$ or a $\exists SO(FP)$ formula. This affords us the freedom to use any off-the-shelf solver for these languages to perform the approximative reasoning. Moreover, it also makes it significantly easier to update the method by adding (or removing) specific propagations.

In detail, the contributions of this paper are that (1) we have extended the logical representation describing the propagation process to a general method for approximating $SAT(\exists\forall SO)$ problems; (2) we have also added approximations for inductive definitions, using a translation to FO(FP). A final, if somewhat preliminary, contribution is that we have examined how existing approximation methods fit into our general framework. In future work, we hope to extend this analysis, by investigating more thoroughly the relation to these methods, both in terms of efficiency and completeness. Moreover, we are confident that a further literature study will reveal more instances of approximation algorithms that are covered by our results.

References

1. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge university press, Cambridge (2003)
2. Baral, C., Gelfond, M., Kosheleva, O.: Expanding queries to incomplete databases by interpolating general logic programs. J. Log. Program. 35(3), 195–230 (1998)
3. Denecker, M.: The well-founded semantics is the principle of inductive definition. In: Dix, J., Fariñas del Cerro, L., Furbach, U. (eds.) JELIA 1998. LNCS (LNAI), vol. 1489, pp. 1–16. Springer, Heidelberg (1998)
4. Denecker, M., Cortés-Calabuig, A., Bruynooghe, M., Arieli, O.: Towards a logical reconstruction of a theory for locally closed databases. ACM Transactions on Database Systems (2010) (accepted)
5. Denecker, M., Ternovska, E.: A logic of nonmonotone inductive definitions. ACM Trans. Comput. Log. 9(2) (2008)
6. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczynski, M.: The second answer set programming competition. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 637–654. Springer, Heidelberg (2009)
7. Doherty, P., Magnusson, M., Szalas, A.: Approximate databases: a support tool for approximate reasoning. Journal of Applied Non-Classical Logics 16(1-2), 87–118 (2006)
8. Immerman, N.: Descriptive Complexity. Springer, Heidelberg (1998)
9. Mariën, M., Wittocx, J., Denecker, M.: The IDP framework for declarative problem solving. In: Search and Logic: Answer Set Programming and SAT, pp. 19–34 (2006)
10. Mitchell, D.G., Ternovska, E.: A framework for representing and solving np search problems. In: AAAI, pp. 430–435 (2005)
11. Ping, H., De Cat, B., Denecker, M.: Fo(fd): Extending classical logic with rule-based fixpoint definitions. In: International Conference on Logic Programming, ICLP 2010 (2010)
12. Son, T.C., Tu, P.H., Gelfond, M., Ricardo Morales, A.: An approximation of action theories of and its application to conformant planning. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 172–184. Springer, Heidelberg (2005)
13. Stickel, M.E.: A prolog technology theorem prover: Implementation by an extended prolog compiler. J. Autom. Reasoning 4(4), 353–380 (1988)
14. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic II. Seminars in Mathematics: Steklov Mathematical Institute, vol. 8, pp. 115–125. Consultants Bureau, New York (1968)
15. Van Gelder, A.: The alternating fixpoint of logic programs with negation. Journal of Computer and System Sciences 47(1), 185–221 (1993)
16. Wittocx, J.: Finite Domain and Symbolic Inference Methods for Extensions of First-Order Logic. PhD thesis, K.U.Leuven (May 2010)
17. Wittocx, J., Mariën, M., Denecker, M.: Approximate reasoning in first-order logic theories. In: KR, pp. 103–112 (2008)

Horn Contraction via Epistemic Entrenchment

Zhi Qiang Zhuang and Maurice Pagnucco

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

Abstract. Belief change studies the way in which a reasoner should maintain its beliefs in the face of newly acquired information. The AGM account of belief change assumes an underlying logic containing classical propositional logic. Recently, there has been interest in studying belief change, specifically contraction, under the Horn fragment of propositional logic (i.e., *Horn logic*). In this paper we continue this line of research, and propose a Horn contraction that is based on the *Epistemic Entrenchment* (EE) construction of AGM contraction. The standard EE construction refers to arbitrary disjunctions which are not available in Horn logic. Therefore, we make use of a Horn approximation technique called *Horn strengthening*. An ideal Horn contraction should be as plausible as an AGM contraction. In other words it should performs identically with AGM contractions when restricted to Horn logic. We demonstrate that no EE based Horn contraction satisfies this criterion unless we apply certain restrictions to the AGM contraction. A representation theorem is proved which identifies the characterising postulates for our Horn contraction.

1 Introduction

Belief change, simply put, studies the way a reasoner should alter its corpus of beliefs as it acquires new information. The benchmark approach in this area is the AGM [1] framework, named after its authors, which introduces three types of belief change: *belief expansion* where the reasoner simply adds the new information to its belief corpus; *belief revision* where the reasoner adds the new information to its belief corpus but in so doing may remove previously held beliefs in order to maintain consistency; and *belief contraction* where the reasoner gives up some of its beliefs to which it no longer gives credence or to contemplate other possibilities.

The AGM approach to belief change assumes an underlying logic that includes classical propositional logic. This paper investigates belief contraction where the underlying logic is founded on the Horn fragment of propositional logic, that is Horn logic. A *Horn clause* is a disjunction of literals consisting of at most one positive literal, e.g., $\neg p \vee \neg q \vee r$. Horn logic is merely propositional logic but restricted to Horn clauses and conjunctions of Horn clauses.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 339–351, 2010.
© Springer-Verlag Berlin Heidelberg 2010

Horn contraction has been studied in [2,3,4]. [2] points out that the topic is of interest because it gives insight into the belief change process by considering a weaker underlying logic and because Horn logic has been applied numerous times in both AI and in databases. Additionally [3] indicates that ideas from Horn contraction are applicable in repairing ontologies in *description logics* [5], as Horn clauses correspond closely to subsumption statements in description logics and removal of problematic subsumptions can be achieved by a contraction operation.

Unlike previous approaches which are analogues of the AGM *partial meet contraction* (PMC) [1], we develop a Horn contraction that is analogous to the AGM *epistemic entrenchment based contraction* (EEC) [6].

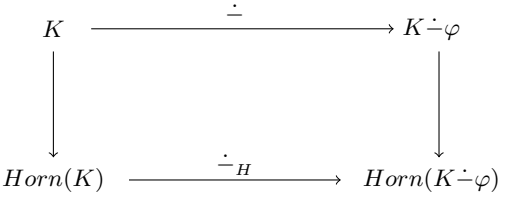


Fig. 1.

Ideally, Horn contractions should be as plausible as AGM contractions. Since we are working with Horn logic, it is reasonable to conclude that a Horn contraction is as plausible as an AGM contraction if the two contractions perform identically in terms of Horn formulas. This idea is illustrated in Figure 1, where $\dot{-}_H$ is the Horn contraction and $\dot{-}$ is the standard AGM contraction. Given a belief set K, $Horn(K)$ is the set of all Horn formulas in K and furthermore if K is associated with an EE-relation $\leq$ then the corresponding EE-relation associated with $Horn(K)$ is the Horn subset of $\leq$. To be more precise, the Horn contraction $\dot{-}_H$ which applies to the Horn formulas in a belief set K and a Horn formula φ has the same behaviour as applying (an equivalent)[1] AGM contraction to K by φ if the same Horn formulas are returned in both cases. Thus the following equality is desirable:

$$Horn(K)\dot{-}_H\varphi = Horn(K\dot{-}\varphi)$$

However, our investigation shows that in general the equality does not hold for EE based Horn contractions. It will be clear that the problem lies not in how the Horn contraction is defined, but in the limited expressiveness of Horn logic.

[1] While we are using the term "equivalent" loosely here, this will become clearer later. Our approach is based on the standard EE construction which imposes an ordering over the formulas of the underlying language. $\dot{-}$ and $\dot{-}_H$ are considered equivalent if $\dot{-}$ is based on an EE-relation $\leq$ which orders Horn formulas in exactly the same way as $\leq_H$ on which $\dot{-}_H$ is based. $\leq$ otherwise orders the non-Horn formulas only according to the restrictions of the entrenchment relation.

We demonstrate that certain restrictions can be applied to the (equivalent) AGM contraction to make it Horn equivalent to our Horn contraction.

In a more general setting, the problem of adapting the AGM framework to a wider class of logics is initiated by the work of [7]. They study belief change in a general logic and give a condition (i.e., *Decomposability*, see [7] for details) for the existence of a contraction operator satisfying the *Recovery* postulate of AGM. [8] has adopted the method in [7] and applied it to Horn logic, where they conclude that the condition does not hold in Horn logic so that any Horn contraction defined does not satisfy Recovery. [9] proposes the postulate of *Core-retaintment* and shows that it is equivalent to the Recovery postulate. Since Horn contraction is not compatible with Recovery we instead investigate the Core-retaintment property.

2 Technical Preliminaries

We assume a propositional language $\mathcal{L}$, over a set of atoms $\mathbf{P} = \{p, q, \ldots\}$, with standard model-theoretic semantics. Lower case Greek characters $\varphi, \psi, \ldots$, denote formulas and upper case Roman characters $X, Y, \ldots$, denote sets of formulas. A *Horn clause* is a clause with at most one positive atom. A *Horn formula* is a conjunction of Horn clauses. A *Horn theory* is a set of Horn formulas. The Horn language $\mathcal{L}_H$ is the restriction of $\mathcal{L}$ to Horn formulas. The Horn logic obtained from $\mathcal{L}_H$ has the same semantics as the propositional logic obtained from $\mathcal{L}$, but restricted to Horn formulas and Horn derivability.

Classical logical consequence and logical equivalence are denoted by $\vdash$ and $\equiv$ respectively. Cn is the Tarskian consequence operator such that $Cn(X) = \{\varphi \mid X \vdash \varphi\}$. Logical consequence under Horn logic is denoted by $\vdash_H$ and thus the consequence operator Cn_H under Horn logic is such that $Cn_H(X) = \{\varphi \mid X \vdash_H \varphi\}$.

Selman and Kautz [10] proposed a notion of Horn approximation called *Horn strengthening*, where they consider a clause as a set of atoms and compare them by set inclusion.

Definition 1. *Given a clause φ, its set of Horn strengthenings, denoted by $\varphi_{\mathcal{H}}$ is such that $\varphi^H \in \varphi_{\mathcal{H}}$ iff φ^H is a Horn clause and there is no Horn clause φ' such that $\varphi^H \subset \varphi' \subseteq \varphi$. Given a set of clauses $\{\varphi_1, \ldots, \varphi_n\}$, its set of Horn strengthenings, denoted by $\{\varphi_1, \ldots, \varphi_n\}_{\mathcal{H}}$ is such that $\{\varphi_1^H, \ldots, \varphi_n^H\} \in \{\varphi_1, \ldots, \varphi_n\}_{\mathcal{H}}$ iff $\varphi_i^H \in \varphi_{i\mathcal{H}}$ for $1 \leq i \leq n$.*

A Horn strengthening of a non-Horn clause is formed by removing all but one positive atom. We extend the notion to conjunctions of clauses by treating them as sets of clauses and applying this definition. We also take the convention that the Horn strengthening of an arbitrary formula, logically equivalent to a Horn formula, is the singleton set which contains the Horn formula. For example, let φ be $(\neg a \vee b) \wedge (\neg p \vee q \vee r)$. Since $\neg a \vee b$ is already Horn, two Horn strengthenings are obtained for φ by removing either q or r from $\neg p \vee q \vee r$, that is $\varphi_{\mathcal{H}} = \{(\neg a \vee b) \wedge (\neg p \vee q), (\neg a \vee b) \wedge (\neg p \vee r)\}$.

3 AGM Contraction

In the AGM framework of belief change, belief states are modelled by sets of sentences, called *belief sets*, which are closed under logical deduction. That is, if K is a belief set then $K = Cn(K)$. Three change operations, namely *expansion*, *revision* and *contraction* are studied. Expansion is the incorporation of new beliefs into the belief set without taking any provisions for maintaining consistency, thus expansion of K by φ, denoted $K + \varphi$, is simply the deductive closure of $K \cup \{\varphi\}$. Revision is the incorporation of new beliefs while maintaining consistency of the belief set, thus if the new belief is inconsistent with the original beliefs, some beliefs need to be dropped. Contraction is required when an agent has to give up beliefs from its belief set.

AGM provides *rationality postulates* and specific constructions for the change operations. The postulates are intended to capture the behaviours of a rational agent in expanding, revising, and contracting its beliefs. The constructions specify, abstractly, how the operation is to be performed. Moreover, AGM shows that the operator obtained through a particular construction satisfies all the corresponding postulates and that any operator satisfying the postulates can be represented by the construction. This kind of formal result is known as a *representation theorem*. Following the AGM tradition, similar theorems are proved to demonstrate the appropriateness of the change operator defined in this paper.

This paper focuses on belief contraction. AGM contraction is a function from $2^{\mathcal{L}} \times \mathcal{L}$ to $2^{\mathcal{L}}$. It is characterised by the following set of postulates.

$(K \dot{-} 1)$ $K \dot{-} \varphi = Cn(K \dot{-} \varphi)$.	(Closure)
$(K \dot{-} 2)$ $K \dot{-} \varphi \subseteq K$.	(Inclusion)
$(K \dot{-} 3)$ If $\varphi \notin K$, then $K \dot{-} \varphi = K$.	(Vacuity)
$(K \dot{-} 4)$ If $\nvdash \varphi$, then $\varphi \notin K \dot{-} \varphi$.	(Success)
$(K \dot{-} 5)$ $K \subseteq (K \dot{-} \varphi) + \varphi$.	(Recovery)
$(K \dot{-} 6)$ If $\varphi \equiv \psi$, then $K \dot{-} \varphi = K \dot{-} \psi$.	(Extensionality)
$(K \dot{-} 7)$ $K \dot{-} \varphi \cap K \dot{-} \psi \subseteq K \dot{-} \varphi \wedge \psi$.	(Conjunction overlap)
$(K \dot{-} 8)$ If $\psi \notin K \dot{-} \varphi \wedge \psi$ then $K \dot{-} \varphi \wedge \psi \subseteq K \dot{-} \psi$.	(Conjunction inclusion)

$(K \dot{-} 1)$–$(K \dot{-} 6)$ are known as the *basic* postulates, and $(K \dot{-} 7)$–$(K \dot{-} 8)$ are known as the *supplementary* postulates. In accordance with the basic postulates, contraction returns another belief set $(K \dot{-} 1)$ not implying φ unless φ is a tautology $(K \dot{-} 4)$. Additionally the belief set does not contain any new beliefs $(K \dot{-} 2)$. If φ is not believed originally then the contraction has no effect $(K \dot{-} 3)$. Furthermore, the contraction should be independent of the syntax $(K \dot{-} 6)$ and remove as little information from K as possible $(K \dot{-} 5)$. Detailed explanations of the full set of postulates can be found in [1,11].

Several constructions for contraction exist, making use of different logical notions such as *remainder sets* [1], *epistemic entrenchment* (EE) [6] and *system of spheres* [12].

We review here the construction method via EE. EEC follows the intuition that when forced to give up beliefs we tend to give up the less preferred. Formally, an EE-relation is a total pre-order $\leq$ over the sentences of a belief set and is

intended to capture the degree of plausibility of a sentence; the more entrenched the belief, the more reluctant we are to give it up during contraction. The relation $\varphi \leq \psi$ represents the situation that ψ is at least as entrenched as φ, hence during a contraction the agent is at least as reluctant to give up ψ as they are to give up φ. Moreover, $\varphi < \psi$ means $\varphi \leq \psi$ and $\psi \not\leq \varphi$, and $\varphi = \psi$ means $\varphi \leq \psi$ and $\psi \leq \varphi$. The following constraints are imposed on $\leq$ for it to capture its intended meaning.

$(EE1)$ If $\varphi \leq \psi$ and $\psi \leq \chi$ then $\varphi \leq \chi$	(Transitivity)
$(EE2)$ If $\varphi \vdash \psi$ then $\varphi \leq \psi$	(Dominance)
$(EE3)$ $\varphi \leq \varphi \wedge \psi$ or $\psi \leq \varphi \wedge \psi$	(Conjunctiveness)
$(EE4)$ If $K \nvdash \bot$ then $\varphi \notin K$ iff $\varphi \leq \psi$ for every ψ	(Minimality)
$(EE5)$ If $\varphi \leq \psi$ for every φ then $\vdash \psi$	(Maximality)

It follows from $(EE1)$–$(EE5)$ that $\leq$ satisfies transitivity and connectivity, in which tautologies are most entrenched and non-beliefs are least entrenched.

AGM provides two conditions that establish connections between EE-relations and contraction operators, one of which determines an EE-relation through a contraction operator, and the other which determines a contraction operator through an EE-relation. Given a contraction operator $\dot{-}$, an EE-relation $\leq$ can be obtained through condition $(\mathbf{C} \leq)$:

$(\mathbf{C} \leq)$: $\varphi \leq \psi$ iff $\varphi \notin K\dot{-}\varphi \wedge \psi$ or $\vdash \varphi \wedge \psi$.

The idea is that, in the contraction of $\varphi \wedge \psi$ from K, we are forced to give up φ or ψ (or both). If φ is retracted then it must be the case that ψ is at least as entrenched as φ. In the limiting case that φ and ψ are tautologies, by $(EE2)$ they are equally entrenched. Now given an EE-relation $\leq$, a contraction operator $\dot{-}$ can be obtained through condition $(\mathbf{C}\dot{-})$:

$(\mathbf{C}\dot{-})$: $\psi \in K\dot{-}\varphi$ iff $\psi \in K$ and either $\varphi < \varphi \vee \psi$ or $\vdash \varphi$.

$(\mathbf{C}\dot{-})$ states that belief ψ is retained if it was originally believed ($\psi \in K$) and there is "sufficient evidence" for retaining it ($\varphi < \varphi \vee \psi$) or if it is not possible to remove φ ($\vdash \varphi$). As pointed out by Gardenförs and Makinson [6], the condition presumes the controversial postulate of Recovery and $(\mathbf{C} \leq)$. We already mentioned in Section 1 that Horn logic does not satisfy Recovery. Recasting this condition for Horn logic is the main challenge in this paper.

In [6] we find the following representation theorem giving the correctness of the conditions above.

Theorem 1. *If* $\leq$ *satisfies* $(EE1)$–$(EE5)$*, the contraction* $\dot{-}$ *determined by* $(\mathbf{C}\dot{-})$ *satisfies* $(K\dot{-}1)$–$(K\dot{-}8)$ *and condition* $(\mathbf{C} \leq)$*. If* $\dot{-}$ *satisfies* $(K\dot{-}1)$–$(K\dot{-}8)$ *then* $\leq$ *determined by* $(\mathbf{C} \leq)$ *satisfies* $(EE1)$–$(EE5)$ *and* $(\mathbf{C}\dot{-})$.

4 Horn Contraction via EE

In this section we construct an EE based Horn contraction (EEHC) which is an analogue of EEC in the AGM framework. Moreover we identify the set of postulates that fully characterises the EEHC.

4.1 Construction of EEHC

Our EEHC is constructed by adapting the key ingredients of EEC—the EE-relation, the $(\mathbf{C} \leq)$ condition, and the $(\mathbf{C}\dot{-})$ condition—to Horn logic. The Horn EE-relation is like the standard one satisfying $(EE1)$–$(EE5)$, but contains only relations between Horn formulas. $(\mathbf{C} \leq)$ is already applicable to Horn logic, thus it requires no adaptation. $(\mathbf{C}\dot{-})$ is problematic as it refers to arbitrary disjunctions which may not be Horn formulas. Therefore, a modified condition $(\mathbf{HC}\dot{-})$ is used where non-Horn disjunctions are replaced by their Horn strengthenings.

$(\mathbf{HC}\dot{-})$: $\psi \in H\dot{-}\varphi$ iff $\psi \in H$ and either there exists a Horn strengthening $\chi \in (\varphi \vee \psi)_{\mathcal{H}}$ such that $\varphi < \chi$ or $\vdash \varphi$.

Similar to $(\mathbf{C}\dot{-})$, a belief ψ is retained if it was originally believed and there is "sufficient evidence" for retaining it or if it is not possible to remove φ. For the principal case of $(\mathbf{C}\dot{-})$, retainment of ψ is determined by the relative entrenchment of φ and $\varphi \vee \psi$; if $\varphi \vee \psi$ is strictly more entrenched than φ then we have "sufficient evidence" to retain ψ. Since $\varphi \vee \psi$ may not be a Horn formula, we uses its Horn strengthening (which is the closest Horn approximation that logically implies $\varphi \vee \psi$) in $(\mathbf{HC}\dot{-})$. Furthermore, to retain ψ, it suffices to have *only one* of the Horn strengthenings being strictly more entrenched than φ. Obviously, another option is to require all Horn strengthenings being strictly more entrenched than φ. However, as long as the *minimal change principle* is concerned, the former option which always retains more of the original beliefs during contraction is more appropriate.

Figure 2 illustrates the contraction of $\neg p \vee r$ from $H = Cn_H(\{\neg p \vee q, \neg q \vee r\})$, while H is associated with EE-relations $\leq_1, \leq_2$ and $\leq_3$. Each rectangle represents the formulas in the belief set H and their relative entrenchment. Clauses in the same level are equally entrenched. Clauses in a level higher are strictly more entrenched than those in a level lower. For instance with $\leq_2$, we have $\neg p \vee q = \neg q \vee r = \neg p \vee \neg r \vee q < \neg p \vee r < \neg p \vee \neg q \vee r$. Non-beliefs, tautologies and conjunctions are not shown as their level of entrenchment are uniquely determined by the clauses shown. The underlined clauses are retained after the contraction. $\neg p \vee \neg r \vee q$ is retained with all EE-relation as its disjunction with $\neg p \vee r$ is a tautology which is most entrenched. With $\leq_1$ only $\neg p \vee \neg r \vee q$ is retained. With $\leq_2$, $\neg q \vee r$ and $\neg p \vee \neg q \vee r$ are also retained. As both of them

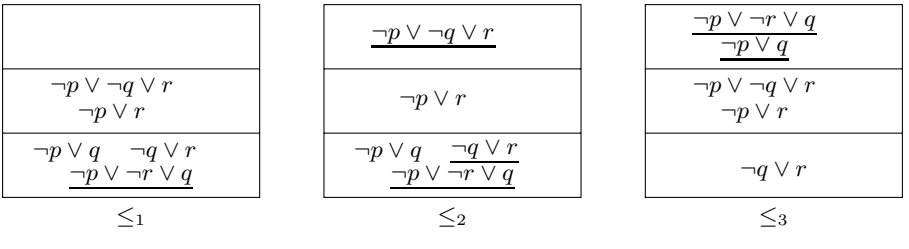


Fig. 2. $H\dot{-}(\neg p \vee r)$

when disjunct with $\neg p \vee r$ yield $\neg p \vee \neg q \vee r$ which is already Horn and is strictly more entrenched than $\neg p \vee r$. With $\leq_3$, the disjunction of $\neg p \vee q$ with $\neg q \vee r$ is $\neg p \vee q \vee r$ which has two Horn strengthenings $\neg p \vee q$ and $\neg q \vee r$. Since $\neg p \vee q$ is strictly more entrenched than $\neg q \vee r$, $\neg p \vee q$ is retained.

4.2 Horn Equivalence of EEHC and EEC

As set out in Section 1, the Horn contraction to be defined should be as plausible as AGM contractions. To be more precise, we want to guarantee that for any belief set K and Horn formula ϕ, the Horn contraction of ϕ from the Horn subset of K ($Horn(K)$) yields a resulting belief set ($Horn(K)\dot{-}_H\phi$)) that is exactly the Horn subset of the resulting belief set ($K\dot{-}\phi$) yielded by the AGM contraction of ϕ from K. This can be formalised by the equality: $Horn(K)\dot{-}_H\varphi = Horn(K\dot{-}\varphi)$. Since ($\mathbf{HC}\dot{-}$) is stronger than ($\mathbf{C}\dot{-}$), one half of the equality holds for EEHC:

Proposition 1. *Let K be a belief set and $\varphi \in \mathcal{L}_H$. If $\dot{-}$ and $\dot{-}_H$ are EEC and EEHC respectively then*

$$Horn(K)\dot{-}_H\varphi \subseteq Horn(K\dot{-}\varphi)$$

Proof. It suffices to show if $\psi \in Horn(K)\dot{-}_H\varphi$ then $\psi \in Horn(K\dot{-}\varphi)$. We sketch the proof for the principal case when $\psi \in K$ and $\nvdash \varphi$. By ($\mathbf{HC}\dot{-}$) there is a $\chi \in (\varphi \vee \psi)_{\mathcal{H}}$ s.t. $\varphi < \chi$. It follows from $\chi \vdash \varphi \vee \psi$ (Definition 1) and ($EE2$) that $\chi \leq \varphi \vee \psi$. We have by ($EE1$) and $\varphi < \chi$ that $\varphi < \varphi \vee \psi$ in which case $\psi \in K\dot{-}\varphi$ follows from ($\mathbf{C}\dot{-}$). Finally as $\psi \in \mathcal{L}_H$, $\psi \in Horn(K\dot{-}\varphi)$. □

However, as demonstrated in Figure 3, the other half of the equality does not always hold for EEHC. Figure 3 illustrates the contraction of $\neg p \vee r$ from $K = Cn(\{\neg p \vee q, \neg q \vee r\})$ (with the associated EE-relations $\leq_1$ and $\leq_2$) by EEC and the contraction of $\neg p \vee r$ from $H = Cn_H(\{\neg p \vee q, \neg q \vee r\}$ (with the associated EE-relation $\leq$ i.e., $\leq_1$ of Figure 2) by EEHC. H is the Horn subset of K (i.e., $H = Horn(K)$). Horn formulas are entrenched identically in $\leq_1$, $\leq_2$ and $\leq$. Non-Horn formula $\neg p \vee q \vee r$ is entrenched differently in $\leq_1$ and $\leq_2$. We can see that the EEC based on $\leq_1$ retains the same Horn formulas as the EEHC whereas the EEC based on $\leq_2$ retains more Horn formulas than the EEHC.

In fact the equality does not hold for any EE based Horn contraction unless certain restriction is applied to the corresponding EE-relation. The reason is straightforward. An EE based Horn contraction is specified by a Horn EE-relation which is obviously not sufficient to specify an EEC as it does not contain relations between non-Horn formulas. There may be several standard EE-relations (such as $\leq_1$, $\leq_2$ in Figure 3) that are identical in terms of Horn formulas with a Horn EE-relation (such as $\leq$ in Figure 3) yet are different in terms of non-Horn formulas. The standard EE-relations give rise to different EECs and the EE based Horn contraction based on the Horn EE-relation corresponds to only some of the EECs by means of the equality.

In what follows we present under what condition the equality holds for our EEHC. We can draw intuitions from the contractions in Figure 3. In $\leq_1$, the

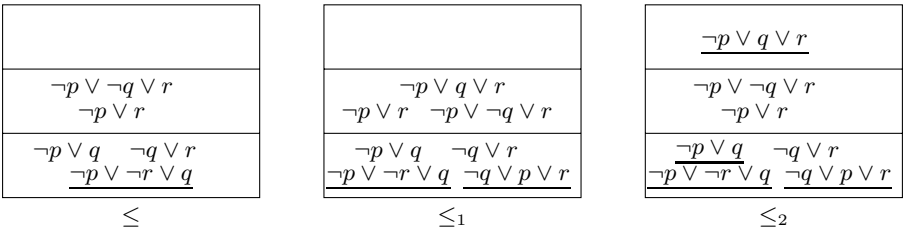


Fig. 3. $H \dot{-}_H (\neg p \vee r)$ and $K \dot{-} (\neg p \vee r)$

non-Horn clause $\neg p \vee q \vee r$ is as entrenched as one of its Horn strengthenings $\neg p \vee r$, whereas in $\leq_2$, $\neg p \vee q \vee r$ is strictly more entrenched than all its Horn strengthenings. Moreover the equality holds with $\leq_1$ but not with $\leq_2$. As we will prove, given a standard EE-relation $\leq$ and a Horn EE-relation $\leq_H$ such that they are identical in terms of Horn formulas, in $\leq$ if each non-Horn formula is equally entrenched to one of its Horn strengthenings, then the EEC based on $\leq$ is equivalent to the EEHC based on $\leq_H$ in terms of Horn formulas. Since the above condition is a constraint for EE-relations we refer it as $(EE6)$.

$(EE6)$ For each φ, there is a $\psi \in \varphi_{\mathcal{H}}$ such that $\varphi \leq \psi$.

The formal result is as follows:

Proposition 2. *Let K be a belief set with associated EE-relation $\leq$ and $\varphi \in \mathcal{L}_H$. Let $\dot{-}$ and $\dot{-}_H$ be the EEC and the EEHC respectively. If $\leq$ satisfies $(EE6)$ then*

$$Horn(K) \dot{-}_H \varphi = Horn(K \dot{-} \varphi)$$

Proof. If $\vdash \varphi$, the equality holds immediately. So suppose $\nvdash \varphi$. We already have by Proposition 1 that $Horn(K) \dot{-}_H \varphi \subseteq Horn(K \dot{-} \varphi)$. It remains to prove if $\psi \in Horn(K \dot{-} \varphi)$ then $\psi \in Horn(K) \dot{-}_H \varphi$. Assumes $\psi \in Horn(K \dot{-} \varphi)$, we have by $(\mathbf{C}\dot{-})$ that $\psi \in K$ and $\varphi < \varphi \vee \psi$. Then from $(EE6)$ there is a $\chi \in (\varphi \vee \psi)_{\mathcal{H}}$ s.t. $\varphi \vee \psi \leq \chi$. But it follows from $(EE1)$, $\varphi < \varphi \vee \psi$, and $\varphi \vee \psi \leq \chi$ that $\varphi < \chi$. Hence, $\psi \in Horn(K) \dot{-}_H \varphi$ follows from $(\mathbf{HC}\dot{-})$. □

4.3 Characterising Postulates

The following set of postulates as we will show formally characterise our EEHC.

$(H\dot{-}1)$ $H \dot{-} \varphi = Cn_H(H \dot{-} \varphi)$.
$(H\dot{-}2)$ $H \dot{-} \varphi \subseteq H$.
$(H\dot{-}3)$ If $\varphi \notin H$ or $\vdash \varphi$, then $H \dot{-} \varphi = H$.
$(H\dot{-}4)$ If $\nvdash \varphi$, then $\varphi \notin H \dot{-} \varphi$.
$(H\dot{-}5)$ If $\psi \in H \dot{-} \varphi \wedge \psi$ then $\psi \in H \dot{-} \varphi \wedge \psi \wedge \delta$
$(H\dot{-}6)$ $\vdash \varphi \equiv \psi$, then $H \dot{-} \varphi = H \dot{-} \psi$.
$(H\dot{-}7)$ $H \dot{-} \varphi \cap H \dot{-} \psi \subseteq H \dot{-} \varphi \wedge \psi$.
$(H\dot{-}8)$ If $\psi \notin H \dot{-} \varphi \wedge \psi$ then $H \dot{-} \varphi \wedge \psi \subseteq H \dot{-} \psi$.
$(H\dot{-}9)$ If $\psi \in H$ and $\psi \notin H \dot{-} \varphi$ then $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H \dot{-} \varphi$
$(H\dot{-}10)$ If $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H \dot{-} \varphi \wedge \chi$ then $\psi \notin H \dot{-} \varphi$

Postulates $(H\dot{-}1)$, $(H\dot{-}2)$, $(H\dot{-}4)$, $(H\dot{-}6)$–$(H\dot{-}8)$ are the Horn analogues of the AGM contraction postulates. $(H\dot{-}3)$ is the *Success* postulate but contains an additional antecedent for capturing the failure property which states the contraction of a tautology has no effect on the belief set. $(H\dot{-}5)$ is the well know *Conjunctive trisection* postulate [13,14]. Given the set of AGM postulates, $(H\dot{-}5)$ is derivable from $(K\dot{-}7)$, but the proof needs Recovery. $(H\dot{-}5)$ is crucial in showing the completeness of EEHC. $(H\dot{-}9)$ and $(H\dot{-}10)$ capture the relationship between a formula in the belief set and the formula to be contracted by means of their disjunction. $(H\dot{-}9)$ states that if ψ is removed in the contraction by φ then so are all Horn strengthenings of $\phi \vee \psi$. $(H\dot{-}10)$ states that if all Horn strengthenings of $\phi \vee \psi$ are removed in the contraction by φ[2] then so is ψ. These relationships turn out to be unique in EE based Horn contractions.

We start with a soundness theorem.

Theorem 2. *If $\dot{-}$ is an EEHC then it satisfies $(H\dot{-}1)$–$(H\dot{-}10)$ and the condition $(\mathbf{C}\leq)$.*

Proof. We omit the proofs for $(H\dot{-}1)$–$(H\dot{-}8)$. By Proposition 3, $\dot{-}$ is Horn equivalent to a special case of EEC, so obviously $\dot{-}$ satisfies all postulates EEC satisfies when the postulates are restricted to Horn logic (Recovery is an exception).

$(\mathbf{C}\leq)$: $(\Rightarrow)$ Suppose that $\varphi \leq \psi$ whilst $\varphi \in H\dot{-}\varphi \wedge \psi$; we need to show $\vdash \varphi \wedge \psi$. Since $\varphi \in H\dot{-}\varphi \wedge \psi$ we have by $(\mathbf{HC}\dot{-})$ that $\varphi \in H$ and either $\vdash \varphi \wedge \psi$ or $\exists \chi \in ((\varphi \wedge \psi) \vee \varphi)_{\mathcal{H}}$ s.t. $\varphi \wedge \psi < \chi$, that is $\varphi \wedge \psi < \varphi$ as by Definition 1 and $(\varphi \wedge \psi) \vee \varphi \equiv \varphi$, we have $((\varphi \wedge \psi) \vee \varphi)_{\mathcal{H}} = \varphi_{\mathcal{H}} = \{\varphi\}$. But since $\varphi \leq \psi$ we have by $(EE2)$ and $(EE3)$ that $\varphi \leq \varphi \wedge \psi$, which implies $\varphi \wedge \psi \not< \varphi$. $(\Leftarrow)$ Suppose either $\varphi \notin H\dot{-}\varphi \wedge \psi$ or $\vdash \varphi \wedge \psi$, we need to show $\varphi \leq \psi$. $\vdash \varphi \wedge \psi$ implies $\vdash \psi$, so $\varphi \leq \psi$ as required by $(EE2)$. Suppose then $\nvdash \varphi \wedge \psi$. Since $\varphi \notin H\dot{-}\varphi \wedge \psi$ and $\nvdash \varphi \wedge \psi$ we have by $(\mathbf{HC}\dot{-})$ that either $\varphi \notin H$ or $\forall \chi \in ((\varphi \wedge \psi) \vee \varphi)_{\mathcal{H}}$ s.t. $\varphi \wedge \psi \not< \chi$, that is $\varphi \wedge \psi \not< \varphi$ (with the same argument as in $(\Rightarrow)$). It follows from connectivity of $\leq$ that $\varphi \wedge \psi \not< \varphi$ implies $\varphi \leq \varphi \wedge \psi$. Finally $\varphi \notin H$ gives us $\varphi \leq \psi$ as required by $(EE4)$ and $\varphi \leq \varphi \wedge \psi$ also gives us, $\varphi \leq \psi$ as required by $(EE1)$, $(EE2)$ and $\varphi \wedge \psi \vdash \psi$.

$(H\dot{-}9)$: Assume $\psi \in H$ and $\psi \notin H\dot{-}\varphi$, we need to show $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H\dot{-}\varphi$. It follows from $(\mathbf{HC}\dot{-})$ and $\psi \notin H\dot{-}\varphi$ that $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\varphi < \chi$. Moreover for all such χ, $(\varphi \vee \chi)_{\mathcal{H}} \subseteq (\varphi \vee \psi)_{\mathcal{H}}$ follows from Definition 1. Hence by $(\mathbf{HC}\dot{-})$, $\chi \notin H\dot{-}\varphi$ for all χ.

$(H\dot{-}10)$: Suppose $\psi \notin H$ then $\psi \notin H\dot{-}\varphi$ follows from $(H\dot{-}3)$. So suppose $\psi \in H$. Assume $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H\dot{-}\varphi \wedge \chi$. We have by $(\mathbf{C}\leq)$ that $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \leq \varphi$ which implies $\varphi \not< \chi$. So $\psi \notin H\dot{-}\varphi$ follows from $(\mathbf{HC}\dot{-})$. □

Also a completeness theorem can be shown.

[2] *Conjunctive factoring* [1] states either $K\dot{-}\varphi \wedge \psi = K\dot{-}\varphi$, $K\dot{-}\varphi \wedge \psi = K\dot{-}\psi$, or $K\dot{-}\varphi \wedge \psi = K\dot{-}\varphi \cap K\dot{-}\psi$ which is satisfied by EEHC. Thus $\chi \notin H\dot{-}\varphi$ implies $\chi \notin H\dot{-}\varphi \wedge \chi$

Theorem 3. *If $\dot{-}$ satisfies $(H\dot{-}1)$–$(H\dot{-}10)$ then $\dot{-}$ is an EEHC.*

Proof. We need to show
1) EE-relation $\leq$ determined by $(\mathbf{C}\leq)$ and $\dot{-}$ satisfies $(EE1)$–$(EE5)$.
2) $\dot{-}$ satisfies $(\mathbf{HC}\dot{-})$.

The proof for $(EE1)$–$(EE5)$ follows exactly as in [15, pages 191–192]. For $(\mathbf{HC}\dot{-})$, we sketch the proof for the principal case when $\psi \in H$ and $\nvdash \varphi$. We prove the contrapositive for $(\mathbf{HC}\dot{-})$ which is: $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \leq \varphi$ iff $\psi \notin H\dot{-}\varphi$.

($\Rightarrow$) Assume $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \leq \varphi$. Then we have by $(\mathbf{C}\leq)$, $\chi \notin H\dot{-}\varphi \wedge \chi$. So by $(H\dot{-}10)$, $\psi \notin H\dot{-}\varphi$.

($\Leftarrow$) Assume $\psi \notin H\dot{-}\varphi$. Then we have by $(H\dot{-}9)$, $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H\dot{-}\varphi$. It then follows from *conjunctive factoring* [1] and $\chi \notin H\dot{-}\varphi$ that $\chi \notin H\dot{-}\varphi \wedge \chi$. Hence, we have by $(\mathbf{C}\leq)$, $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \leq \varphi$. □

4.4 Postulate of Core-Retainment

A well known postulate for contraction, namely *Core-retainment* [9] captures the intuition that formulas not contributing to the fact that a belief set K implies a formula φ, should be retained in $K\dot{-}\varphi$. [9] refers to these formulas as the *φ-cores* of K.

Definition 2. *[9] ψ is a φ-core of K iff $\psi \in K$ and for all $X \subseteq K$: if $\varphi \notin Cn(X)$ then $\varphi \notin Cn(X \cup \{\psi\})$.*

Core-retainment then ensures that if a formula is removed then it must not be a core.

Core-retainment: If $\psi \in K$ and $\psi \notin K\dot{-}\varphi$ then there is a set X such that $X \subseteq K$, $\varphi \notin Cn(X)$ and $\varphi \in Cn(X \cup \{\psi\})$

As shown in [9], given the other AGM postulates, Core-retainment is equivalent to Recovery. Since Horn logic does not satisfy Recovery, it is tempting to see how Horn contractions perform against Core-retainment. The Horn contraction defined in [3] is further refined in [16] in which the authors conclude that their contraction satisfies Core-retainment. It is not difficult to show that the Horn contraction from [2] also satisfies Core-retainment. However, this is not the case for our EEHC. Figure 4 illustrates the contraction of $\neg p \vee q$ from $H = Cn_H(\{\neg r \vee s, \neg p \vee q\})$. Since $\neg r \vee s$ does not contribute to the implication of $\neg p \vee q$ in H, it is a $(\neg p \vee q)$-core of H, yet it is discarded.

It turns out that Core-retainment, while not necessary in our approach, is compatible with it. An important property of *cores* is pointed out by [9].

Proposition 3. *ψ is a φ-core of K iff $\psi \in K$ and $\vdash \varphi \vee \psi$.*

This property does not hold in Horn logic. To regain Core-retainment in Horn contraction it is necessary to force retainment of formulas like $\neg r \vee s$ which is a $(\neg p \vee q)$-*core* only in Horn logic. The alternative property of *core*s in Horn contraction is as follows:

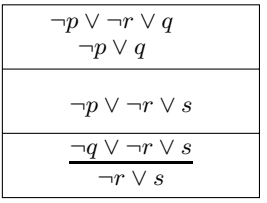


Fig. 4. $H \dot{-}_H(\neg p \vee q)$

Proposition 4. *Let H be a Horn belief set and $\psi, \varphi \in \mathcal{L}$. ψ is a φ-core of H iff $\psi \in H$ and one of the following holds:*
1). $\vdash \varphi \vee \psi$
2). $\forall \chi \in (\psi \rightarrow \varphi)_{\mathcal{H}}$ either $\chi \vdash \varphi$ or $\chi \notin H$.

The following example illustrates this proposition.

Example 1. Continuing with the contraction in Figure 4. Since $\nvdash \neg p \vee q \vee \neg r \vee s$ condition *1)* does not hold, we check condition *2)*. As $(\neg r \vee s) \rightarrow (\neg p \vee q) \equiv (\neg p \vee q \vee r) \wedge (\neg p \vee \neg s \vee q)$ we have $((\neg r \vee s) \rightarrow (\neg p \vee q))_{\mathcal{H}} = \{\beta_1, \beta_2\}$ for $\beta_1 = (\neg p \vee q) \wedge (\neg p \vee \neg s \vee q)$ and $\beta_2 = (\neg p \vee r) \wedge (\neg p \vee \neg s \vee q)$. For β_2 we have $\beta_2 \notin H$ follows from $H \nvdash \neg p \vee r$. For β_1 we have $\beta_1 \vdash \neg p \vee q$. Hence condition *2)* holds and we have $\neg r \vee s$ is a $(\neg p \vee q)$-*core* of H.

(**HC**$\dot{-}$) is now revised to include the extra condition in Proposition 4.

(**HC**$\dot{-}$**Core**): $\psi \in H \dot{-} \varphi$ iff $\psi \in H$ and one of the following holds:
1). $\exists \chi \in (\varphi \vee \psi)_{\mathcal{H}}$ s.t. $\varphi < \chi$.
2). $\forall \chi \in (\psi \rightarrow \varphi)_{\mathcal{H}}$ either $\chi \vdash \varphi$ or $\chi \notin H$.
3). $\vdash \varphi$.

Note that condition *1)* of Proposition 4 is covered by *1)* of (**HC**$\dot{-}$**Core**). The modified EEHC now satisfies Core-retainment.

Theorem 4. *If $\leq$ satisfies $(EE1)$–$(EE5)$ then the contraction $\dot{-}$ determined by* (**HC**$\dot{-}$**Core**) *satisfies $(H\dot{-}1)$–$(H\dot{-}9)$, $(H\dot{-}10C)$, $(H\dot{-}C)$, and* (**C** $\leq$).

$(H\dot{-}C)$ is the core-retainment postulate in Horn logic and $(H\dot{-}10C)$ includes in its antecedent conditions that ψ not be a φ-*core*.

$(H\dot{-}10C)$ If $\forall \chi \in (\varphi \vee \psi)_{\mathcal{H}}$, $\chi \notin H \dot{-} \varphi \wedge \chi$, and $\exists \chi \in (\psi \rightarrow \varphi)_{\mathcal{H}}$ s.t. $\chi \nvdash \varphi$ and $\chi \in H$ then $\psi \notin H \dot{-} \varphi$.

5 Related Work and Concluding Remark

The Horn contractions[3] studied in [2,3] and [4] are analogues of the AGM PMC. They differ in the way of adapting the notion of remainder set to Horn logic.

[3] Package contraction and base contraction are also studied in these works but we consider only belief set contraction which is the focus of the current paper.

[2] sticks to the original definition of remainder sets. The Horn contraction thus defined is usually referred to as *orderly maxichoice contraction*. One property of PMC is lost when turning to Horn logic. That is, every belief set in between (by set inclusion) the one returned by the *full meet contraction* and the one returned by a *maxichoice contraction* is obtainable by taking intersections of some remainder sets (i.e., covered by some PMC). In [3], all such belief sets are considered as appropriate resulting belief sets, so that the notion of remainder sets is generalised for their Horn contraction to return all appropriate results. Most recently, [4] adopted the model theoretic way of obtaining remainder sets and applied it to Horn contraction. The distinguishing feature of their contraction is that it avoids a triviality result similar to the AGM maxichoice contraction:[4]

For p not mentioned in H, we have $(H \dot{-} \varphi) + p \vdash \varphi$

[4] also shows that both [2] and [3] suffer from this triviality result.

There is a close connection between PMC and EEC in the AGM framework [17]. However, this connection is unexplored for Horn contractions and is left as future work. We believe that such a connection is essential for comparing our EEHC to the existing ones and for determining the most appropriate form of Horn contraction. In other aspects, our Horn contraction also avoids the triviality result. However, it can be easily shown any Horn contraction satisfying Core-retainment suffers from the triviality result and so does our contraction when amended for satisfying Core-retainment. Through Theorem 3 we find that in order to characterise our EE based Horn contraction, two extra postulates (i.e., $(H \dot{-} 9)$ and $(H \dot{-} 10)$) are required, which capture the relative entrenchment of some closely related Horn clauses during contraction. The existing Horn contractions do not satisfy $(H \dot{-} 9)$ and $(H \dot{-} 10)$. The reason is that in defining our Horn contraction we have in mind the full set of AGM postulates, whereas [3] and [4] focus only on the basic set of postulates. [2] also works with the full set of AGM postulates, but restricted to maxichoice contraction.

In this paper we also propose a criterion as depicted in Figure 1 for checking plausibility of a Horn contraction. Our investigation shows that EE based Horn contraction fails this criterion due to the inexpressiveness of Horn logic. So it is interesting to see how the Horn contractions based on remainder sets perform against this criterion which we also leave for future work.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. J. Symb. Logic 50(2), 510–530 (1985)
2. Delgrande, J.P.: Horn clause belief change: Contraction functions. In: Proc. KR 2008, pp. 156–165 (2008)
3. Booth, R., Meyer, T., Varzinczak, I.J.: Next steps in propositional Horn contraction. In: Proc. IJCAI 2009, pp. 702–707 (2009)

[4] It is easy to show their approach avoids the triviality result only when the formula to be contracted is not a disjunction of negative atoms.

4. Delgrande, J.P., Wassermann, R.: Horn clause contraction function: Belief set and belief base approaches. In: Proc. KR 2010 (2010)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. CUP, Cambridge (2003)
6. Gärdenfors, P., Makinson, D.: Revisions of knowledge systems using epistemic entrenchment. In: Proc. TARK 1988, pp. 83–95 (1988)
7. Flouris, G., Plexousakis, D., Antoniou, G.: Generalizing the AGM postulates: preliminary results and applications. In: Proc. NMR 2004, pp. 171–179 (2004)
8. Langlois, M., Sloan, R.H., Szörényi, B., Turán, G.: Horn complements: Towards Horn-to-Horn belief revision. In: Proc. AAAI 2008, pp. 466–471 (2008)
9. Hansson, S.O.: Belief contraction without recovery. Studia Logica 50(2), 251–260 (1991)
10. Selman, B., Kautz, H.: Knowledge compilation using Horn approximations. In: Proc. AAAI 1991, pp. 904–909. MIT Press, Cambridge (1991)
11. Gärdenfors, P.: Knowledge in Flux: Modelling the Dynamics of Epistemic States. MIT Press, Cambridge (1988)
12. Grove, A.: Two modellings for theory change. Journal of Philosophical Logic 17(2), 157–170 (1988)
13. Rott, H.: Preferential belief change using generalized epistemic entrenchment. JoLLI 1(1), 45–78 (1992)
14. Hansson, S.O.: Changes of disjunctively closed bases. JoLLI 2(4), 255–284 (1993)
15. Hansson, S.O.: A Textbook of Belief Dynamics Theory Change and Database Updating. Kluwer, Dordrecht (1999)
16. Booth, R., Meyer, T., Varzinczak, I., Wassermann, R.: A contraction core for Horn belief change: Preliminary report. In: Proc. NMR 2010 (2010)
17. Rott, H.: Two methods of constructing contractions and revisions of knowledge systems. Journal of Philosophical Logic 20(2), 149–173 (1991)

The DMCS Solver for Distributed Nonmonotonic Multi-Context Systems*

Seif El-Din Bairakdar, Minh Dao-Tran, Thomas Eiter,
Michael Fink, and Thomas Krennwallner

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{bairakdar,dao,eiter,fink,tkren}@kr.tuwien.ac.at

1 Introduction

The DMCS system is an implementation of the equilibrium semantics for heterogeneous and nonmonotonic multi-context systems (MCS) [3], which feature contexts with heterogeneous and possibly nonmonotonic logics. Each context in an MCS comprises of two parts: a local knowledge base and a set of bridge rules that can access the beliefs of other contexts and add new information to the knowledge base. In this setting, contexts are loosely coupled, and may model distributed information linkage applications; thus it is natural to have a system that allows for the distributed evaluation of MCS.

In an MCS $M = (C_1, \ldots, C_n)$, each context C_i is characterized by a knowledge base kb_i and a set of bridge rules br_i. In our implementation, each kb_i is in DLV syntax as in [6]. The br_i are sets of nonmonotonic rules

$$p_0 \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathbf{not}\ (c_{j+1} : p_{j+1}), \ldots, \mathbf{not}\ (c_m : p_m).$$

where the $(c_k : p_k)$ are bridge atoms; the index c_k refers to a context C_{c_k} and p_k is a possible belief of C_{c_k}; intuitively, the atom is true if p_k is in the belief set of context C_{c_k}. If the body evaluates to true with respect to a belief state, which is a sequence $S = (S_1, \ldots, S_n)$ of belief sets S_i of C_i, $1 \leq i \leq n$, then p_0 has to be added to kb_i. The semantics of M is then given in terms of stable belief sets (called equilibria). *Partial Equilibria* are equilibria in a sub-MCS of M induced by a single context C_k resp. a collection $C_{k_1}, \ldots, C_{k_j}$ of contexts.

Example 1. Consider an MCS $M = (C_1, C_2)$, where C_1, C_2 have answer set programs in the local knowledge bases; specifically

$kb_1 = \{a_1 \leftarrow b_1; \bot \leftarrow \text{not } b_1\}$ and $br_1 = \{b_1 \leftarrow (2 : a_2)\}$;

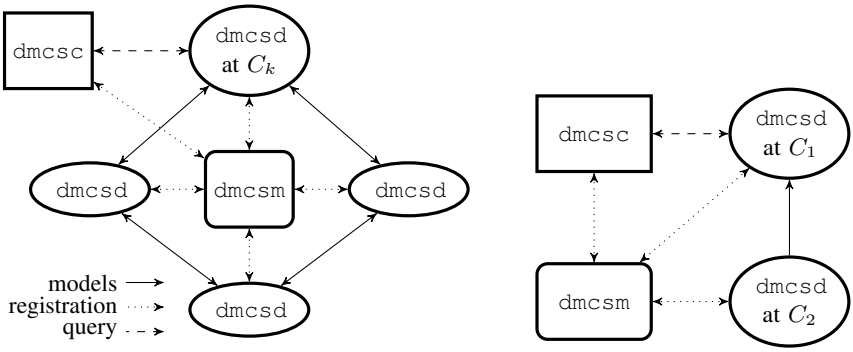
$kb_2 = \{a_2 \vee b_2 \leftarrow\}$ and $br_2 = \emptyset$.

Then $S = (\{a_1, b_1\}, \{a_2\})$ is the only equilibrium of M; as well as the only partial equilibrium of M w.r.t. C_1. Note that w.r.t. C_2, M has two partial equilibria: $S^{(1)} = (\epsilon, \{a_2\})$ and $S^{(2)} = (\epsilon, \{b_2\})$ (here ϵ means the context is not reachable).

The algorithm in [4] describes a generic distributed procedure for evaluating (partial) equilibria of multi-context systems. It has been refined with an effective decomposition

* This research has been supported by the Austrian Science Fund (FWF) project P20841 and by the Vienna Science and Technology Fund (WWTF) project ICT 08-020.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 352–355, 2010.
© Springer-Verlag Berlin Heidelberg 2010



(a) System Architecture **(b)** Components of Ex. 2

Fig. 1. DMCS System Architecture

technique in [1]. Our DMCS system comprises both algorithms and allows for MCS with contexts that have propositional answer set programs as knowledge bases. Initial experimental results with the DMCS system were shown in [1,4].

The basic idea for our system [4] is to take the bridge rules and the knowledge base of a context, compile them to a propositional theory, and use a SAT solver to compute the models. The distributed algorithms then take care of combining the models and generate equilibria at the context that initiated the computations.

DMCS is a purely distributed framework written in C++. It uses *clasp* [5] for local model building. The system is available at

`http://www.kr.tuwien.ac.at/research/systems/dmcs/`.

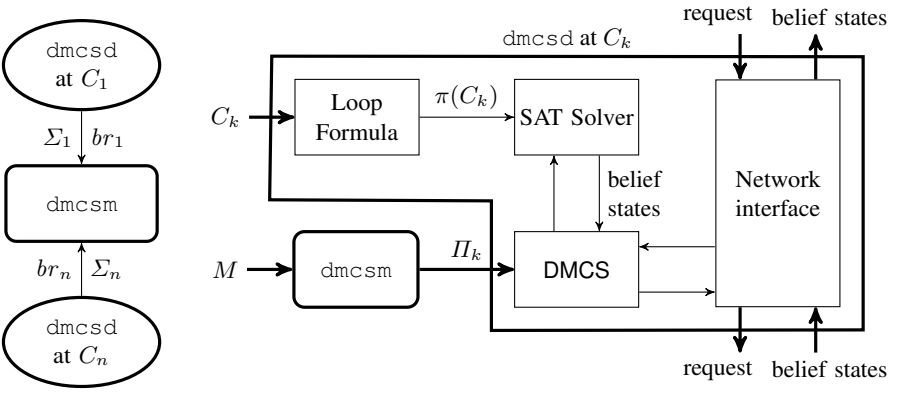
2 System Architecture and Evaluation

The architecture of DMCS is outlined in Figure 1a, which has the following main components: (i) a front-end `dmcsc` for querying the multi-context system; (ii) daemons `dmcsd`, where each of them represents a context and interacts with the others; a daemon has four modules, namely Loop Formula, SAT Solver, DMCS, and Network Interface (cf. Figure 2b); and (iii) a component `dmcsm` holding meta information about the MCS that has been collected from each context. The system has four stages which are briefly described as follows:

System start-up. At this stage, all running `dmcsd` processes register at the `dmcsm`, provide their own set of bridge rules, alphabet as well as port and host name (Figure 2a). With this information, the `dmcsm` component identifies the topology of the system and gets ready to answer any question regarding this meta knowledge.

Initialization of `dmcsd`. Upon initialization, each `dmcsd` utilizes the Loop Formula module to transform its local knowledge base and bridge rules into a SAT theory denoted by $\pi(C_k)$ in DIMACS format (see [4] for details). Then, it starts listening for incoming requests from other daemons, or from queries of `dmcsc` described next.

Querying the system. When the user wants to know all partial equilibria of the system w.r.t. a starting context C_k, she uses `dmcsc` to pose the query. She may specify



(a) Start-up **(b)** dmcsd System Architecture

Fig. 2. DMCS System

variables, which will be provided as the initial request to C_k. First, dmcsc inquiries the dmcsm component about C_k and gets back the connection settings of this context. Then, dmcsc sends the query to the respective dmcsd representing C_k and waits for the results.

Evaluating the System. After dmcsc has sent a query to the dmcsd process that represents the starting context C_k, the daemon computes partial belief states w.r.t. interface variables and projects unwanted variables away. If C_k needs beliefs from neighboring contexts, it sends a request to them and awaits their belief states, which will be consistently combined with the local beliefs of C_k. Essentially, those requests look just as queries sent from dmcsc, and every dmcsd will process them in a uniform manner. After all neighbors have been addressed, C_k returns the partial equilibria to the client, who presents them to the user.

The algorithm used in dmcsd is an ASP logic instance of the generic DMCS algorithm presented in [4]. Alternatively, dmcsd may use an adapted version of this algorithm, DMCSOPT, which exploits dependencies in the MCS, uses economically small representations of them, and uses minimal interface variables needed for minimizing data transmission (see [1]). Here, query plans Π_k w.r.t. C_k are key for guiding the evaluation process and may be provided by dmcsm.

3 System Usage

For a concrete usage scenario of DMCS, we reconsider the MCS in the example above.

Example 2 (cont'd). In order to evaluate our example, one has to set up a system as illustrated in Figure 1b, by executing two start up calls, possibly on different machines.

```
$ dmcsd --context=1 --kb=C1.kb --br=C1.br --manager=HOST:PORT
$ dmcsd --context=2 --kb=C2.kb --br=C2.br --manager=HOST:PORT
```

The command-line argument `--context` tells the daemon the context id that it will represent. The knowledge base and the bridge rule files are provided via `--kb` and `--br`, resp. The `--manager` option is used to set up the location of the `dmcsm` component.

To compute equilibria of M w.r.t. context C_1, the user queries the `dmcsm` to get the connection settings for the `dmcsd` representing C_1 using the following command (the parameters have the same meaning as above).

```
$ dmcsc --context=1 --manager=HOST:PORT
```

After `dmcsd` at C_1 finishes its computations, it delivers the result back to `dmcsc`. A list of the equilibria is then enumerated to the user:

```
( {a1,b1}, {a2} )
Total Number of Equilibria: 1
```

It is possible to specify for DMCS a set of atoms of interest; other atoms will then be discarded. Unless such a set is given, `dmcsc` will assume its default settings and proceed with standard operations.

4 Conclusions

The DMCS system is, to the best of out knowledge, the first implementation of a fully distributed algorithm to evaluate heterogeneous and nonmonotonic multi-context systems. Other related systems like the one in [7] does not allow cyclic references in bridge rules, and the system in [2] is based on a query evaluation approach.

The method for computing partial equilibria induced by some context can be easily extended to compute equilibria of the whole system; this, however, may be of less interest from the perspective of an individual context (e.g., in a peer-to-peer style evaluation).

Our ongoing work aims at further extending the implementation and optimization, as well as on dynamic configuration of MCS by instantiating generic bridge rules.

References

1. Bairakdar, S., Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Decomposition of distributed nonmonotonic multi-context systems. In: JELIA 2010. Springer, Heidelberg (September 2010)
2. Bikakis, A., Antoniou, G., Hassapis, P.: Strategies for contextual reasoning with conflicts in ambient intelligence. Knowl. Inf. Syst. (2010) (published online: April 9, 2010)
3. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI 2007, pp. 385–390. AAAI Press, Menlo Park (July 2007)
4. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Distributed nonmonotonic multi-context systems. In: KR 2010, pp. 60–70. AAAI Press, Menlo Park (May 2010)
5. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007, pp. 386–392. AAAI Press, Menlo Park (January 2007)
6. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Logic 7(3), 499–562 (2006)
7. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 361–376. Springer, Heidelberg (2005)

The MCS-IE System for Explaining Inconsistency in Multi-Context Systems*

Markus Bögl, Thomas Eiter, Michael Fink, and Peter Schüller

Institute of Information Systems
Vienna University of Technology
Favoritenstrasse 11, A-1040 Vienna, Austria
markus.boegl@student.tuwien.ac.at, {eiter,fink,schueller}@kr.tuwien.ac.at

Abstract. The Multi-Context System Inconsistency Explainer allows for evaluation of semantics and explanation of inconsistencies in systems where heterogeneous knowledge bases are linked via nonmonotonic rules. The implementation is based on the dlvhex tool, which is an extension of answer set programming with external atoms and higher order features.

1 Introduction

Nonmonotonic multi-context systems (MCSs) were introduced in [1], mainly but not exclusively as an extension of [5,2]. They are a formalism for interlinking heterogeneous knowledge bases (called contexts), whose semantics is defined via (possibly non-unique) belief sets, via bridge rules that may be non-monotonic. For example, $(2 : b) \leftarrow (1 : a), not(3 : a)$ expresses that context 2 should add b to its knowledge base, if context 1 has a in its belief set while context 3 has not. The semantics of MCSs is then defined in terms of belief states, which contain one belief set per context, that satisfy a stability condition (called equilibria).

Inconsistency is the absence of such equilibria. An inconsistent MCS yields no information, therefore our aim is to explain reasons for inconsistency in MCSs in order to support users in dealing with inconsistency. For this purpose, the notions of *diagnosis* and *inconsistency explanation* were introduced in [3].

A diagnosis (D_1, D_2) points out a set of bridge rules D_1 which must be removed from an MCS M, and a set of bridge rules D_2 whose unconditional form (i.e., $\alpha \leftarrow$ for $\alpha \leftarrow \beta$) must be added to M to restore consistency in M. An explanation (E_1, E_2) points out a set of bridge rules E_1 which is required in M, and a set of bridge rules E_2 whose unconditional forms must not be added to M to ensure inconsistency in M. Diagnoses allow to find overall system repairs, whereas inconsistency explanations separate individual sources of inconsistency.

A method for evaluating these notions by a rewriting to logic programming was introduced in [3]. This work describes the MCS Inconsistency Explainer system MCS-IE, which realizes and extends the implementation concepts of [3].[1]

* This work was supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-020.

[1] http://www.kr.tuwien.ac.at/research/systems/mcsie/

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 356–359, 2010.
© Springer-Verlag Berlin Heidelberg 2010

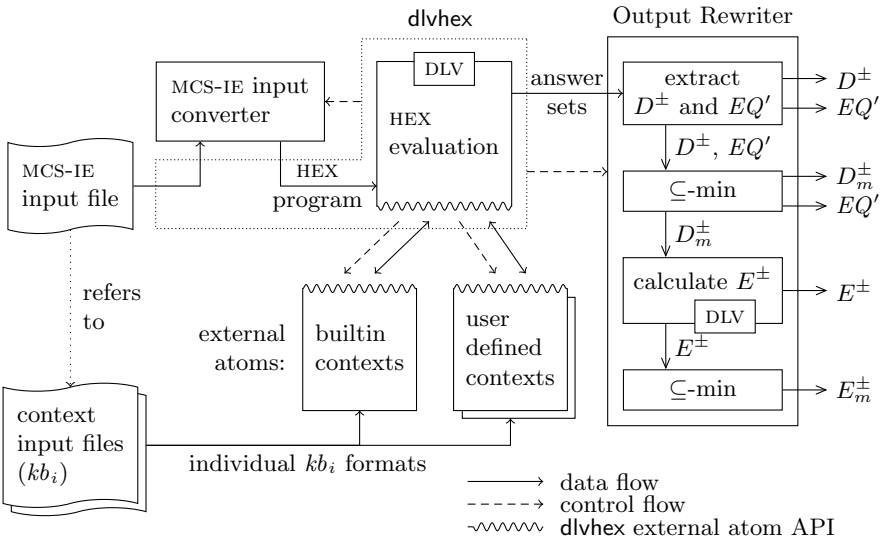


Fig. 1. Data flow between MCS-IE components, dlvhex and DLV

Our system uses the dlvhex solver[2] and HEX programs, which are an extension of answer set programs (ASPs) with external atoms [4]. We use external atoms for capturing context semantics, since each context of an MCS can be based on its own logic, and in general cannot be rewritten to ASPs.

The MCS-IE system provides the following features:

- computation of MCS semantics (output projected equilibria),
- computation of diagnoses, explanations, and their subset-minimal notions,
- support for contexts formalized in ASP (specifically: DLV programs), and
- an API in C++ for integrating user-defined contexts.

2 System Architecture

Figure 1 shows the architecture of the MCS-IE system. It is implemented as a plugin to dlvhex, therefore dlvhex controls all MCS-IE components.

To analyze inconsistency in an MCS, a master input file which describes the MCS topology (bridge rules and contexts) must be provided by the user. dlvhex rewrites the master file into a HEX program using the MCS-IE input converter. For space reasons, we refer to the MCS-IE website for details of this rewriting.[1] The main idea is to guess a diagnosis using auxiliary predicates, to guess a belief state, and to evaluate bridge rule semantics based on the guesses. Context semantics are then evaluated using dlvhex external atoms, which may require additional context input files (e.g., ASP program files, or databases).

[2] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

Each answer set of the rewritten HEX program describes a diagnosis of the given MCS and a corresponding equilibrium. The MCS-IE output rewriter component extracts this information and converts it into a human readable format (see the example below). For the conversion of diagnoses to inconsistency explanations the output converter generates an answer set program from the extracted diagnosis, and evaluates it using DLV. Depending on command-line parameters,[1] sets of (subset-minimal) explanations for inconsistency in the MCS are displayed.

3 Example

In the following, we give an example MCS along with its encoding for MCS-IE.

Our MCS is a health care decision support system, which consists of a database of lab test results C_1, a patient record database C_2, an ontology C_3 for disease classification, and an expert system C_4 suggesting suitable treatments.

The MCS topology is specified in file `master.hex`, contexts C_1, C_2, and C_4 are realized in corresponding DLV program files:

master.hex:
```
#context(1,"dlv_asp_context_acc", "kb1.dlv").
#context(2,"dlv_asp_context_acc", "kb2.dlv").
#context(3,"ontology_context3_acc", "").
#context(4,"dlv_asp_context_acc", "kb4.dlv").
r1: (3:pneum)           :- (2:xraypneum).
r2: (3:marker)          :- (2:marker).
r3: (4:need_ab)         :- (3:pneum).
r4: (4:need_strong)     :- (3:atyppneum).
r5: (4:allow_strong_ab) :- not (1:allergystrong).
```

kb1.dlv:
```
allergystrong.
```

kb2.dlv:
```
marker. xraypneum.
```

kb4.dlv:
```
give_strong v give_weak :- need_ab.
give_strong             :- need_strong.
give_nothing            :- not need_ab, not need_strong.
:- give_strong, not allow_strong_ab.
```

Intuitively, C_1 and C_2 provide information that the patient is allergic to strong antibiotics, that a certain blood marker is present, and that pneumonia was detected in an X-ray examination. C_4 suggests a treatment which is either a strong antibiotic, a weak antibiotic, or no medication at all.

Context C_3 is implemented in C++ and uses the MCS-IE API. The following source code (namespaces and includes are omitted) builds into a `dlvhex` plugin:

```
DLVHEX_MCSEQUILIBRIUM_PLUGIN(MedExamplePluginContext3,0,1,0)
DLVHEX_MCSEQUILIBRIUM_CONTEXT(Context3,"ontology_context3_acc")
set<set<string> > Context3::acc(
      const string& param, const set<string>& input) {
  set<set<string> > ret;
  set<string> s(input.begin(), input.end());
  if( input.count("pneum") == 1 && input.count("marker") == 1 )
```

```
    s.insert("atyppneum");
  ret.insert(s); return ret;
}
void MedExamplePluginContext3::registerAtoms()
  { registerAtom<Context3>(); }
```

The first two lines creates a dlvhex plugin and an external atom usable in `#context(...)`. Context semantics is implemented in function `acc`, which gets bridge rule heads as `input` and returns accepted belief sets. Finally, the external atom is registered in the plugin using `registerAtoms`.

Roughly, C_3 specifies that presence of pneumonia together with a blood marker (stemming from r_2) yields atypical pneumonia in the belief state.

Note that this system is inconsistent, as the expert system concludes that a patient must be given a special drug, but the patient record states that she is allergic to that drug, a counter-indication. This inconsistency can be explained using MCS-IE as follows (assuming the MCS-IE plugin in the current directory):

```
$ dlvhex --plugindir=./ --ieenable --ieexplain=Dm,Em master.hex
```

MCS-IE calculates the following output, containing minimal diagnoses plus witnessing equilibria (`Dm:EQ:`), and minimal inconsistency explanations (`Em`):

```
Dm:EQ:({r1},{}):({allergystrong},{marker,xraypneum},{})
Dm:EQ:({r2},{}):({allergystrong},{marker,xraypneum},{pneum},{})
Dm:EQ:({r4},{}):({allergystrong},{marker,xraypneum},{atyppneum,pneum},{})
Dm:EQ:({},{r5}):({allergystrong},{marker,xraypneum},{atyppneum,pneum},{})
Em:({r1,r2,r4},{r5})
```

Accordingly, deactivating r_1, or r_2, or r_4, or adding r_5 unconditionally, will restore consistency, and there is a single inconsistency involving all rules except for r_3.

4 Conclusions

The MCS-IE system may serve as a valuable tool for analyzing inconsistencies in MCSs. It is geared towards functionality, not (yet) towards efficiency. An interactive demo is available.[1] Future work will be the implementation of further external atoms, e.g., for accessing DBMS.

References

1. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI, pp. 385–390 (2007)
2. Brewka, G., Roelofsen, F., Serafini, L.: Contextual default reasoning. In: IJCAI, pp. 268–273 (2007)
3. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in nonmonotonic multi-context systems. In: KR, pp. 329–339 (2010)
4. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: IJCAI, pp. 90–96 (2005)
5. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics, or: How we can do without modal logics. Artificial Intelligence 65(1), 29–70 (1994)

Coala: A Compiler from Action Languages to ASP

Martin Gebser[1], Torsten Grote[1], and Torsten Schaub[1,2]

[1] Universität Potsdam, Institut für Informatik, August-Bebel-Str. 89, D-14482 Potsdam
[2] Simon Fraser University, Canada, and Griffith University, Australia

Abstract. Action languages allow for compactly describing dynamic domains. They are usually implemented by compilation, e.g., to Answer Set Programming. To this end, we developed a tool, called *Coala*, offering manifold compilation techniques for several action languages. We provide an overview of the salient and distinctive features of *Coala* as well as an experimental comparison of them.

1 Introduction

Action languages provide a compact formal model for describing dynamic domains [1], being central to many applications like model checking, planning, robotics, etc. Moreover, action languages can be implemented rather efficiently through compilation to Answer Set Programming (ASP; [2]) or Satisfiability Checking (SAT; [3]). Our system *Coala* takes advantage of this by offering a variety of different compilation techniques for several action languages.

Coala originates from *al2asp*, constituting the heart of the *BioC* system [4] used for reasoning about biological models in action language $\mathcal{C}_{TAID}$ [5]: *al2asp* compiles $\mathcal{C}_{TAID}$ to $\mathcal{C}$, which is in turn mapped to ASP via the transformation in [6]. *Coala* extends the capacities of *al2asp* in several ways. First, it adds certain features of $\mathcal{C}$+ [7] and provides full support of $\mathcal{B}$ [8] (and $\mathcal{A}_L$). Second, it offers different compilation schemes. Apart from a priori bounded encodings using standard ASP systems, *Coala* furnishes incremental encodings that can be used in conjunction with the incremental ASP system *iClingo* [9]. Moreover, *Coala* distinguishes among forward and backward (incremental) encodings, depending on whether trajectories are successively extended from initial states or whether they are built backwards starting from final states. Third, *Coala* supports all action query languages, $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$, in [1]. Fourth, *Coala* allows for posing LTL-like queries, following [10]. Finally, *Coala* offers the usage of first-order variables that are treated by the underlying ASP grounder. Optionally, type checking for variables can be enabled. *Coala* is implemented in `C++` and can also be used as a library; it is freely available at [11].

Our general approach is similar to the one taken by $DLV^{\mathcal{K}}$ [12] for processing action language $\mathcal{K}$. Similarly, $CCalc$ [7] addresses $\mathcal{C}$+ in its generality and maps it to SAT. We provide in [4] an empirical evaluation comparing *al2asp* (aka *Coala*'s bounded encoding of $\mathcal{C}$ following [6]) in combination with *Gringo* and *clasp* to $CCalc$ and $DLV^{\mathcal{K}}$ on benchmarks expressed in $\mathcal{C}$. Among the three systems, this rudimentary version of *Coala* had an edge in terms of robustness, having by far the fewest number of timeouts. Hence, in what follows we concentrate on the novel features of *Coala* going well beyond existing systems.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 360–364, 2010.
© Springer-Verlag Berlin Heidelberg 2010

2 Coala at Work

Coala starts with parsing an action description by means of an easily adaptable `bison++`-based parser before compiling it into a non-ground logic program. This program is grounded by *Gringo* and optionally extended by further ground facts before trajectories are generated via *clasp*. In what follows, we sketch the major compilation features of *Coala* illustrated by constructs of $\mathcal{C}$ [1].

To begin with, *Coala* generates via option `-e` either instance-based or direct encodings. For instance, the dynamic causal law

```
<caused> -alive <if> hit <after> shoot.
```

can be mapped onto either the fact `caused(neg(alive),hit,true,shoot).` or the rule `-fluent_alive(T) :- not -fluent_hit(T), action_shoot(T-1).` While a direct encoding is executable without further additions (cf. [6]), an instance-based encoding relies on meta-interpretation through an accompanying encoding. Although we do not detail it here, such meta-interpretation is very flexible and thus an easy way to implement different strategies.

Another major feature of *Coala* is the usage of incremental ASP solving techniques (via option `-i`), as provided by *iClingo*.[1] In this case, an action description is mapped onto a parametrized threefold logic program $B \cup P[k] \cup Q[k]$, among which $P[k]$ and $Q[k]$ contain a parameter k ranging over positive integers. Program B describes static knowledge, independent of k. The role of $P[k]$ is to capture knowledge accumulating with increasing k, whereas $Q[k]$ is specific for each value of k. The goal is then to compute answer sets of program $B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$ for some (minimum) integer $i \geq 1$. In an incremental setting, the above dynamic law is mapped onto

```
#cumulative t.
-fluent_alive(t) :- not -fluent_hit(t),action_shoot(t-1).
```

indicating that the rule belongs to $P[\mathtt{t}]$; its ground instances are successively produced and accumulated in the solver. Similarly, declarations `#base.` and `#volatile t.` indicate whether a rule belongs to B or $Q[k]$, respectively. Unlike this, a non-incremental setting is guarded by a fixed number of time steps `t`, provoking repetitive grounding of rules during iterative deepening search.

A third major feature is *Coala*'s distinction between forward and backward (incremental) encodings (via option `-r`), depending on whether trajectories are successively extended from initial states or whether they are built backwards starting from final states. This is implemented by means of meta-interpretation. To get an impression, consider the following three "meta-rules":

```
1 { holds(F,-t), holds(neg(F),-t) } 1 :- fluent(F).
fire(F,G,P,A,-t) :- caused(F,G,P,A), occurs(A,-t),
                    holds(P,-t), holds(G,-t+1).
:- fire(F,G,P,A,-t), not holds(F,-t+1).
```

The first rule aims at guessing a predecessor state (time stamp `-t`). The second one determines firing dynamic laws. Third, the integrity constraint ensures that the effects of firing causal laws are consistent with the successor state (time stamp `-t+1`).

[1] Note that *iClingo* relies on *Gringo* and *clasp*.

Moreover, *Coala* supports LTL-like queries, using *next*, *finally*, *globally*, *until*, *weak until*, and *release*, viz. `X, F, G, U, W, R`, and aims at generating counterexamples. For instance, the simple LTL query `LTL: X alive.` asks whether the fluent `alive` is true in the next step in all trajectories. Following [10], this is translated to

```
ltl_counter_example :- ltl_f_2(0).
ltl_f_2(0) :- -fluent_alive(1).
```

producing counterexamples in which the complement of `alive` holds. More complex LTL formulas require additional auxiliary rules and are omitted here for brevity.

Finally, a typical call of *Coala* looks as follows:

```
coala -l b -i bw.alb | cat - bw.stat | iclingo 0
```

The options '`-l b -i`' tell *Coala* that `bw.alb` is written in $\mathcal{B}$ and that it should be compiled into an incremental ASP program. The latter is then augmented with static domain knowledge in `bw.stat` before *iClingo* is invoked to compute all answer sets for a minimum number of time steps. The interested reader is directed to [11] for more details on the language and usage of *Coala*.

3 Experiments

We conducted experiments in order to evaluate the different compilation techniques furnished by *Coala*. To this end, we confined ourselves to action language $\mathcal{C}$ and concentrate on combinations of several *Coala* options: '`-n`' enables dedicated handling of classical negation; '`-i`' produces an incremental encoding for *iClingo*; '`-e`' uses meta-interpretation (rather than direct encoding); '`-r`' uses backward encoding. The `default` setting includes none of these features. All experiments were run with *iClingo* (2.0.5), using *clasp* (1.3.2) in its default settings on an Intel Core 2 Duo CPU at 2.66 GHz running Ubuntu GNU/Linux 9.10 with RAM usage limited to 1.5 GB. All programs were run sequentially as single threads on one CPU core.

Table 1. Experiments comparing different target compilations of *Coala*

		`default`		`-n`		`-i`		`-e`		`-e -i`		`-e -i -r`	
Benchmark	#	time	stime	time	stime	time	stime	time	stime	time	stime	time	stime
blocksworld/b08	16	44.19	20.90	44.31	21.84	4.30	4.23	323.43	69.98	76.83	74.83	112.95	109.78
blocksworld/b09	16	53.56	9.27	55.78	9.58	11.68	11.58	TO	TO	283.90	280.33	338.64	332.95
blocksworld/b10	17	88.69	20.53	77.38	11.74	14.69	14.56	TO	TO	366.34	361.42	MO	MO
blocksworld/b11	18	403.31	276.31	403.78	247.84	117.53	117.37	TO	TO	TO	TO	MO	MO
blocksworld/b12	19	228.47	160.37	290.91	163.54	69.99	69.81	MO	MO	MO	MO	MO	MO
ferryman/f03	15	29.38	6.80	29.79	7.26	2.67	2.63	49.84	7.43	7.40	6.99	13.09	12.31
ferryman/f04	17	65.40	14.48	64.00	13.26	3.48	3.44	96.89	14.20	11.84	11.17	26.96	25.71
ferryman/f05	19	132.22	26.67	100.30	26.83	4.36	4.32	170.14	23.46	19.35	18.31	46.27	44.35
ferryman/f06	17	122.92	28.63	120.93	26.29	6.03	5.97	202.99	28.12	26.57	25.26	67.57	65.13
ferryman/f07	19	243.05	54.12	257.18	50.53	18.04	17.96	356.48	48.50	40.12	38.19	138.62	135.09
hanoi/h03	31	85.89	9.77	88.48	10.68	1.83	1.80	50.80	5.01	4.66	4.50	281.18	280.86
hanoi/h04	63	TO	TO	TO	TO	69.51	69.41	TO	TO	54.74	54.19	TO	TO
yale/y04	18	6.22	0.16	6.14	0.20	10.18	10.14	11.35	0.07	5.13	5.10	16.11	16.08
yale/y05	20	40.91	0.14	35.52	0.79	16.83	16.80	64.19	0.86	40.59	40.56	69.95	69.91
yale/y06	22	132.51	5.88	165.62	4.50	79.05	79.02	87.20	0.34	177.14	177.10	TO	TO
yale/y07	24	547.40	0.45	TO	TO	205.35	205.31	499.40	22.02	TO	TO	TO	TO
Average (Outs)		176.51 (1)		183.76 (2)		39.72 (0)		307.04 (5)		182.16 (3)		294.46 (6)	

Our results are summarized in Table 1. The first two columns give the respective benchmark along with its horizon (#). Note that the next three columns use direct encodings, while the last three rely upon meta-interpretation. Column *time* is average CPU time from three runs per benchmark; *stime* is average time needed by the solver (in the final successful run during iterative deepening search; and in total for `-i`). An entry TO indicates timeout after 600 seconds, while MO means that the processes were aborted at 1.5 GB RAM consumption. The last row shows the average CPU time (and number of timeouts) over all benchmarks. In case of timeout, a time of 600 seconds was assumed.

Looking at the global outcome in the last row, we observe that incremental direct encodings (`-i`) perform best over all benchmarks (except for *h04* and *y04*). Although worse, the incremental non-direct counterpart (`-e -i`) performs best on average among the meta-interpreted encodings. Changing to a more complex backward encoding (`-e -i -r`) does not lead to an improvement and yields two more memory exhaustions than the other meta-interpreted encodings. Pure meta-interpretation (`-e`), suffering from a grounding overhead, performs worst, despite of solving one more instance than the backward encoding. No clear difference was observable on the usage of built-in classical negation (`-n`), producing more integrity constraints than a dedicated treatment (`default`).

All in all, we observe that an incremental approach to action languages is largely beneficial. The usage of backward encodings may make a difference on particular problem classes. Although meta-interpretation appears to lead to less efficient encodings, it offers an easy way to experiment with different strategies.

Acknowledgments. This work was partially funded by BMBF project GoFORSYS and DFG grant SCHA 550/8-1.

References

1. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on Artificial Intelligence 3(6), 193–210 (1998)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University, Cambridge (2003)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS, Amsterdam (2009)
4. Dworschak, S., Grote, T., König, A., Schaub, T., Veber, P.: The system BioC for reasoning about biological models in $\mathcal{C}$. In: ICTAI 2008, pp. 11–18. IEEE, Los Alamitos (2008)
5. Dworschak, S., Grell, S., Nikiforova, V., Schaub, T., Selbig, J.: Modeling biological networks by action languages via ASP. Constraints 13(1-2), 21–65 (2008)
6. Lifschitz, V., Turner, H.: Representing transition systems by logic programs. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) LPNMR 1999. LNCS (LNAI), vol. 1730, pp. 92–106. Springer, Heidelberg (1999)
7. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artificial Intelligence 153(1-2), 49–104 (2004)
8. Son, T., Baral, C., Nam, T., McIlraith, S.: Domain-dependent knowledge in answer set planning. ACM Transactions on Computational Logic 7(4), 613–657 (2006)

9. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 190–205. Springer, Heidelberg (2008)
10. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. Theory and Practice of Logic Programming 3(4-5), 519–550 (2003)
11. http://potassco.sourceforge.net
12. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning. Artificial Intelligence 144(1-2), 157–211 (2003)

DLV^{MC}: Enhanced Model Checking in DLV

Marco Maratea[2], Francesco Ricca[1], and Pierfrancesco Veltri[1]

[1] Department of Mathematics, University of Calabria, 87036 Rende (CS), Italy
{veltri,ricca}@mat.unical.it
[2] DIST, University of Genova, 16145 Genova, Italy
marco@dist.unige.it

Abstract. Stable Model Checking (MC) in Answer Set Programming systems is, in general, a co-NP task for disjunctive programs. Thus, implementing an efficient strategy is very important for the performance of ASP systems. In DLV, MC is carried out by exploiting the SAT solver SATZ, and the result of this operation also returns (in case the check fails) an "unfounded set", as by-product, which is also used for pruning the search space during answer set computation.

In this paper we report on the integration of a "modern" SAT solver, MINISAT, in DLV. The integration poses not only technological issues, but also challenges w.r.t. the "quality" of the returned unfounded set and w.r.t. the interplay with the existing DLV techniques.

1 Introduction

Disjunctive Logic Programming under the answer set semantics, a.k.a., Answer Set Programming (ASP for short, [1,2]), is a powerful declarative formalism for knowledge representation and reasoning [3]. ASP is expressive in a precise sense: it allows to solve any problem belonging to the second level of the polynomial hierarchy. The idea of ASP is to represent a given computational problem by a logic program, the answer sets of which correspond to solutions; and, then, to use an answer set solver to find such solutions [4]. Answer set computation of propositional (i.e., without variables) ASP programs is carried out in DLV [5], as well as in most ASP systems (like e.g., GnT [6], Cmodels [7] and CLASPD [8]) by exploiting two main modules: model generator and a model checker. The first, which is similar to the DPLL procedure [9] for SAT, generates "candidate" answer sets; the second verifies if such candidates are indeed answer sets. On the class of non Head-Cycle-Free (HCF) [10] disjunctive programs, stable model checking is co-NP-complete. To handle this case, DLV exploits a reduction to a satisfiability (SAT) problem [11]: verifying if the candidate solution is "stable", i.e., if it is an answer set, is reduced to checking the unsatisfiability of a SAT formula ϕ, built from the ASP program and the candidate solution. If the formula is satisfiable, then the MC module of DLV also returns a set of "unfounded" atoms, i.e., a set of atoms which can be freely assigned to false. Such unfounded set corresponds to the set of atoms assigned to true in a satisfying interpretation of ϕ returned by the SAT solver (thus, it is available for free after a MC call) and is exploited in the model generator of DLV for enhancing the search together with other techniques like partial checking [11,12,13,14]. In particular, once a (total) model check fails during the search DLV goes back in the search

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 365–368, 2010.
© Springer-Verlag Berlin Heidelberg 2010

and performs partial model checks while backtracking, in order to unroll the choices in current partial interpretations causing the original stability failure. This technique is combined with another approach, in which DLV speculatively performs partial model checks while moving forwards (as opposed to backtracking) in the search tree. Those checks allows to detect in advance (i.e., before reaching a complete assignment) that the current branch of the search tree is actually inconsistent.

As a matter of fact, DLV performance on non-HCF programs depends on both the efficiency of the SAT checker employed and the "quality" of the returned "unfounded set" (in case the candidate solution is not stable).

In this paper, we report on the integration of a "modern" SAT solver, MINISAT, in DLV, which results in the enhanced system DLV^{MC}. The integration poses not only technological issues; indeed, modern SAT solvers are significantly different from previous propositional checkers such as SATZ (which is the SAT solver employed in the standard version of DLV). The difference is not only in the data structures and/or in the optimization techniques implemented but also in the termination conditions. A CNF formula is declared satisfiable if either all variables of the problem have been assigned (without any conflict) as in MINISAT; or when all clauses have been satisfied, as in SATZ. The choice of the termination condition clearly affects the nature of the returned assignment, and thus the "quality" of the unfounded set. The branching heuristic we used within MINISAT by default assigns all variables to false, i.e., it guarantees to return subset-minimal "minimal" unfounded sets (see, e.g., [15]). Theoretically, imposing an ordering to be followed while branching can have significant degradation in performance [16], but some preliminary experiments shows that this approach can pay off when exploited in DLV, sometimes by orders of magnitude.

2 System Usage and Options

DLV^{MC} has to be invoked as follows:

*./*DLV^{MC} *-solver <> [-heurm <>] [filename [filename [...]]]*

The command line of DLV^{MC} inherits all the options of DLV, and adds some new options. In more detail: (i) "-solver" indicates the SAT solver to be used for stable model checking. The two SAT solvers available are SATZ and MINISAT ("satz" and "minisat" are the specific strings to be specified in place of <>). DLV^{MC} relies on MINISAT, but the default setting of DLV (i.e., SATZ) can be also selected; and (ii) "-heurm" specifies the type of heuristic used inside MINISAT, by allowing to employ heuristics where variables are first assigned negatively, or positively, or randomly ("neg", "pos", "ran" are the specific strings to be specified in place of <>). Negative first heuristic is the default setting. The option has no effect in case of SATZ.

System Availability. The home page of the system, together with the Linux executables and the QBF benchmarks used are available at: *http://www.mat.unical.it/ricca/DLVMC*.

3 Preliminary Analysis

We have performed a preliminary experimental analysis involving both hard QBF benchmarks coming from the QBF evaluations, and StrategicCompanies benchmarks.

Table 1. Experimental analysis: DLV and DLV^{MC} with (-PC) and without (-noPC), and CLASPD, on QBF and StrategicCompanies benchmarks

instance	DLV -PC	DLV^{MC} -PC	#MC	DLV -noPC	DLV^{MC} -noPC	CLASPD
qbf1	0.85	0.88	65	0.92	0.96	2.31
qbf2	3.95	3.60	129	4.15	3.86	12.93
qbf3	14.53	14.14	257	15.43	15.17	63.47
qbf4	77.39	70.80	513	82.76	75.65	315.68
x100.0q	1.71	0.97	3	28.12	10.14	4.71
x110.0q	13.79	10.96	12/7	95.26	60.19	279.89
x135.02q	80.00	113.24	12/9	TIME	TIME	TIME
x145.0q	137.82	51.86	12/7	TIME	992.80	TIME
x150.02q	233.73	49.89	15/8	TIME	876.94	TIME
x150.04q	208.11	76.74	12/7	TIME	984.08	TIME
x165.03q	376.58	842.01	13/8	TIME	TIME	TIME
x165.04q	286.10	1084.68	12/8	TIME	TIME	TIME
x170.0q	TIME	503.42	–/8	TIME	TIME	TIME
x170.02q	919.03	475.11	15/7	TIME	TIME	TIME
x175.01q	TIME	763.78	–/8	TIME	TIME	TIME
x175.04q	816.79	689.10	15/9	TIME	TIME	TIME
x185.0q	TIME	527.99	–/9	TIME	TIME	TIME

The second ones being the only "hard" benchmarks submitted to the last ASP competition. All experiments were run on a machine equipped with two Intel Xeon "Woodcrest" (quad core) processors clocked at 3.GHz with 4MB of Level 2 Cache and 4GB of RAM, running Debian GNU Linux 4.0. Time measurements have been done using the `time` command provided by the system, counting total CPU time for the respective process. We report the results in terms of execution time for finding one answer set, if any, within 20 minutes ("TIME" otherwise). The virtual memory available to the solvers has been limited to 512MB. qbf1-qbf4 instances are translation of the bigger MutexP QBF instances. Few StrategicCompanies benchmarks are non solved by any system.

Results are presented in Table 1, where the first column is the instance, the second and third columns contain results for DLV with partial check (-PC) employing SATZ and MINISAT, respectively, the fourth column contains the number of stability checks performed (if "DLV -PC" and "DLV^{MC} -PC" perform the same number of checks there is one number), the fifth and sixth columns are the same as the second and third, but with partial check disabled (-noPC), and the last column contain results for CLASPD [8], as reference. "DLV -PC" and "DLV^{MC} -PC" always perform the same number of stability checks on these benchmarks.

On QBF benchmarks, the advantages of DLV^{MC} over DLV is of around a 10% on both configurations, i.e., with and without PC: the number of stability checks is here always the same, and the advantages of DLV^{MC} seem to be due to the efficiency of MINISAT. All DLV-based versions are much faster than CLASPD. On the StrategicCompanies benchmarks, the impact of MINISAT is instead much higher: on all but two instances, "DLV^{MC} -PC" performs much better than "DLV -PC", solving three instances more among the biggest showed. Considering the related number of stability

checks, here is very important both the efficiency of MINISAT *and* the quality of the returned unfounded sets. As a matter of fact, PC is of fundamental importance for the efficiency of DLV; nonetheless, the results without MC show an even more significant and consistent gain for the version with MINISAT, even if the number of stability checks is the same. CLASPD solves only the two smallest instances of this set.

Acknowledgements. We thank Nicola Leone for fruitful discussion about the topic of the paper. This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009. Part of this work has been done while the first author has been with the Università degli Studi e-Campus.

References

1. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP 1988, pp. 1070–1080. MIT Press, Cambridge (1988)
2. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC 9, 365–385 (1991)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. CUP (2003)
4. Lifschitz, V.: Action Languages, Answer Sets and Planning. In: The Logic Programming Paradigm – A 25-Year Perspective, pp. 357–373 (1999)
5. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL 7(3), 499–562 (2006)
6. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL 7(1), 1–37 (2006)
7. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 447–451. Springer, Heidelberg (2005)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007, pp. 386–392 (2007)
9. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. Communications of the ACM 5, 394–397 (1962)
10. Ben-Eliyahu, R., Dechter, R.: Propositional Semantics for Disjunctive Logic Programs. AMAI 12, 53–87 (1994)
11. Koch, C., Leone, N., Pfeifer, G.: Enhancing Disjunctive Logic Programming Systems by SAT Checkers. AI 15(1-2), 177–212 (2003)
12. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Inf.Comp. 135(2), 69–112 (1997)
13. Pfeifer, G.: Improving the Model Generation/Checking Interplay to Enhance the Evaluation of Disjunctive Programs. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 220–233. Springer, Heidelberg (2003)
14. Janhunen, T., Niemelä, I., Simons, P., You, J.H.: Partiality and Disjunctions in Stable Model Semantics. In: KR 2000, vol. 12, 15, pp. 411–419 (2000)
15. Giunchiglia, E., Maratea, M.: Solving optimization problems with DLL. In: Proc. of the 17th European Conference on Artificial Intelligence (ECAI 2006). Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 377–381. IOS Press, Amsterdam (2006)
16. Järvisalo, M., Junttila, T.A., Niemelä, I.: Unrestricted vs restricted cut in a tableau method for boolean circuits. Annals of Mathemathics and Artificial Intelligence 44(4), 373–399 (2005)

A Dynamic-Programming Based ASP-Solver*

Michael Morak, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran

Institut für Informationssysteme E184/2, Technische Universität Wien
Favoritenstr. 9–11, 1040 Wien, Austria

Abstract. We present a novel system for propositional Answer-Set Programming (ASP). This system, called dynASP, is based on dynamic programming and thus significantly differs from standard ASP-solvers which implement techniques stemming from SAT or CSP.

1 Introduction

Answer-Set Programming (ASP, for short) [6,7] is nowadays a well acknowledged paradigm for declarative problem solving as witnessed by many successful applications in the areas of AI and KR. Evaluating ASP-programs relies on two steps, the grounding (which instantiates the variables in the program's rules) and the solving process itself which works on ground (i.e. propositional) programs. For the latter task, many different solvers (see [1] for an overview) exist nowadays, and also the system presented here falls into this category.

Solving ground programs still is an intractable problem. More precisely, decision problems for disjunctive programs (DLPs) are located on the second level of the polynomial hierarchy, but also decision problems defined over disjunction-free programs — usually called normal programs (NLPs) — are NP- or coNP-complete. The same complexity as for NLPs holds if a certain restriction on the usage of disjunction (head-cycle free programs, HCFPs) is assumed. Due to these intractability results, standard-ASP solvers use techniques stemming from SAT or CSP, where intractability has been successfully tackled in practice.

Our solver, which we present here, relies on a more theoretical approach to deal with intractable problems, namely *parameterized complexity theory* where the idea is to bound a certain parameter by a constant in order to obtain tractable fragments for the problems under consideration. One important such parameter is *treewidth*, which measures the "tree-likeness" of a graph. For instance, the problem of deciding ASP consistency (i.e. whether a disjunctive logic program has at least one answer set) has been shown tractable [3] for programs having an incidence graph of bounded treewidth. Treewidth is defined over so-called tree decompositions which in turn can be used by dynamic programming (DP) methods to solve the considered problem. One such algorithm for disjunctive ASP has been presented recently [4]. We refer to [4] also for details how concepts as incidence graphs, treewidth, tree decomposition, etc. are defined in terms of ASP programs. It is however important to note that such DP algorithms can

* Supported by the Austrian Science Fund (FWF), project P20704-N18.

T. Janhunen and I. Niemelä (Eds.): JELIA 2010, LNAI 6341, pp. 369–372, 2010.
© Springer-Verlag Berlin Heidelberg 2010

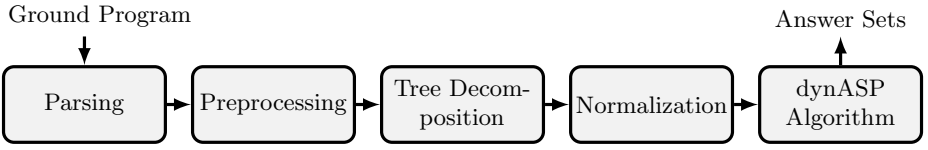


Fig. 1. Architecture of dynASP

be applied to an arbitrary program as soon as a tree decomposition is found for that program. The running time of the DP algorithms however heavily depends on the width of the supplied tree decomposition.

To evaluate this new DP-based approach for ASP, we have implemented two such methods: one for DLPs, cf. [4]; and another one applicable to HCFPs which relies on a rather different idea[1]. Thus, we call our solver dynASP, which first finds a tree decomposition for the given program, and then uses the aforementioned DP methods to evaluate the program. This entire process underlying our approach is hidden from the user. In fact, dynASP presents itself like a standard-ASP solver.

The main aim of this paper is to announce the release of the first prototype of our ASP-solver and to present some implementation details as well as some preliminary experiments. Ongoing and future work will carefully evaluate the potential of this novel approach for solving ASP programs.

2 Architecture

Figure 1 gives an overview of the overall architecture of the system. Generally the system works in five discrete steps:

- **Parsing** DLV-style programs is done via a lex/yacc parser.
- **Preprocessing** is done after successfully reading the input. This task is twofold: Firstly, it performs equivalence-preserving transformations for the provided program (for the moment, this just takes care of some special cases like tautological and empty rules in order to circumvent problems which might arise if such rules are present in the later steps; for future versions this task can be extended to find simplifications in terms of reducing the tree-width of the program without changing its semantics), second it constructs the incidence graph of the program which is then passed to the Tree Decomposition module.
- For computing the tree decomposition, we use an algorithm based on heuristics [2]. This algorithm does not guarantee to find a tree decomposition of minimal width. But it usually finds a tree decomposition of width close to the minimum at comparatively low computational cost. An implementation

[1] Basically, the HCFP algorithm follows the ideas underlying a DP algorithm for weight-constraints programs introduced in [8].

of this algorithm is freely available from `http://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html`.
- **Normalization** guarantees that the difference between a node and its predecessor in the tree decomposition is at most one variable or rule. Our algorithms require tree decompositions of this particular form.
- The actual **algorithm** is run. Depending on the supplied program options, this is either the ASP algorithm for general DLPs or the algorithm for HCFPs, as described earlier.

3 System Specifics

An executable version of dynASP is available under

`http://www.dbai.tuwien.ac.at/research/project/tractability/dynasp/`

dynASP is invoked via command line and provides the following options:

```
dynasp [-b] [-t] [-s <seed>] [-f <file>] -a <alg> -o <output>
```

`-b` print benchmark information
`-t` perform only tree decomposition step
`-s seed` initialize random number generator with "seed"
`-f file` the file to read from
`-a alg` the algorithm to use, one of {`sat`, `minsat`, `asp`, `hcfasp`}
`-o output` the output-type, one of {`enum`, `count`, `yesno`}

The benchmark information consists of timing information being printed to the screen containing information about CPU time used for the tree decomposition, normalization, algorithm and evaluation steps and of course the overall time.

Depending on whether a file option is given, input is either read from a file or from standard input. Depending on the algorithm, the input has to be in a certain format. For the DLP algorithm and its HCF pendant the file has to come in the core language of dlv [5], i.e. restricted to ground disjunctive programs. The algorithm and output options specify which algorithm to use and what the output should be (i.e. enumeration, counting or consistency checking).

dynASP is written in C++ using lex/yacc for parsing the input. In its current version dynASP has nearly 6700 lines of code written in C++ (including the tree decomposition functionality). The core algorithm for DLPs has around 600 lines of C++ code, the algorithm for head-cycle-free programs has about 200 more. Certain auxiliary functionality used by both algorithms is implemented in another 200 lines.

dynASP uses an extensible class structure, allowing for code re-use and easy implementation of various algorithms based on tree decompositions. Both the DLP algorithm and the one for head-cycle-free programs are implemented using this class structure, with the two implementations sharing much of the code used for consistency checking and answer-set enumeration. To illustrate the ability to implement other types of algorithms based on tree decompositions, DP algorithms for SAT and MINSAT have also been implemented using the same framework, however reading files in DIMACS CNF format.

4 Discussion

First experiments with our system on logic programs of low treewidth have resulted in competitive performance compared with state-of-the-art ASP solvers. The performance of our system is particularly favorable in situations where only one pass of the tree decomposition is required (i.e., to check consistency or for counting the answer sets). The evaluation of HCF programs with the dedicated HCFP algorithm tends to display a better performance than the algorithm for general programs. However, in its current implementation, our HCFP algorithm is quite sensitive to the particular form of the tree decomposition. In contrast, the algorithm for general DLPs is rather robust in this respect. Its performance is mainly determined by the treewidth. Up to treewidth $5 - 7$, this performance is comparable to that of the DLV system.

Work on our system is ongoing. Major goals for the near future are further performance improvements, e.g., by introducing better "data structures" or by simplifying the programs to reduce the treewidth (without changing the semantics of the programs). Eliminating the sensitivity of the HCFP algorithm to the particular form of the tree decomposition also falls into this category. Moreover, we plan to extend this framework by implementing further algorithms like e.g., for programs with weight/cardinality constraints, as well as support for input in the SMODELS solver syntax.

References

1. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczynski, M.: The second answer set programming competition. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 637–654. Springer, Heidelberg (2009)
2. Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B.J., Musliu, N., Samer, M.: Heuristic methods for hypertree decomposition. In: Gelbukh, A., Morales, E.F. (eds.) MICAI 2008. LNCS (LNAI), vol. 5317, pp. 1–11. Springer, Heidelberg (2008)
3. Gottlob, G., Pichler, R., Wei, F.: Bounded treewidth as a key to tractability of knowledge representation and reasoning. In: Proc. AAAI 2006, pp. 250–256. AAAI Press, Menlo Park (2006)
4. Jakl, M., Pichler, R., Woltran, S.: Answer-set programming with bounded treewidth. In: Proc. IJCAI 2009, pp. 816–822. AAAI Press, Menlo Park (2009)
5. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Log. 7(3), 499–562 (2006)
6. Marek, V.W., Truszczyński, M.: Stable Models and an Alternative Logic Programming Paradigm. In: The Logic Programming Paradigm – A 25-Year Perspective, pp. 375–398. Springer, Heidelberg (1999)
7. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. 25(3-4), 241–273 (1999)
8. Pichler, R., Rümmele, S., Szeider, S., Woltran, S.: Tractable answer-set programming with weight constraints: Bounded treewidth is not enough. In: Proc. KR 2010, pp. 508–517. AAAI Press, Menlo Park (2010)

Author Index